University of Gothenburg

Chalmers University of Technology

Software Engineering, BSc

DIT-022: Mathematical Foundations for Software Engineering

Course Script

Created by:
Katja Tuma, PhD Student,
katja.tuma@cse.gu.se
Sergio García, PhD Student,
sergio.garcia@cse.gu.se

Teachers:
Dr. Christian Berger, Associate Professor, christian.berger@gu.se
Dr. Richard Torkar, Full Professor, richard.torkar@cse.gu.se

CHALMERS

UNIVERSITY OF
GOTHENBURG

October 6, 2020

Disclaimer

This document is an extension of the topics covered in the videos of this course. It is largely based on previous work, notably the literature mentioned in the course PM and other work, referenced accordingly.

More specifically, the chapters summarize the literature as follows:

1. Chapter 1: Chapters 1.1–1.5 in [4],

2. Chapter 2: Chapters 13.1–13.4 in [4],

3. Chapter 3: Chapter 10 in [4],

4. Chapter 4: Chapters 3, 8.3, 5.3 and 5.4 in [4] and Wikipedia (sorting algorithms),

5. Chapter 5: Chapters 1.7–1.8 and 5.1 in [4],

6. Chapter 6: Chapters 1.1–1,4, 2.1–2.3, 2.5, 2.6, 3.1, 3.2, 3.4–3.7, 4.1–4.6, 5.1–5.6, 6.1–6.3, 6.5, 8.1–8.3, 8.4, 8.5, 12.1, 12.4, 11.1, 11.2, 9.1–9.3, 10.1–10.3 in [5] and Wikipedia.

Contents

Chapter 1

Logic

1.1 Logic

The rules of logic specify the meaning of mathematical statements. Such rules help to understand and reason with statements like "There exists an integer that is not the sum of two squares" or "For every positive integer n, the sum of the positive integers not exceeding n is $\frac{n\times(n+1)}{2}$."

Logic is the basis of all mathematical and automated reasoning. It has practical applications to the design of computing machines, to the specification of systems, to artificial intelligence, to computer programming, to programming languages, and other areas of computer science and many other fields of study.

To understand mathematics, we must understand what constitutes a correct mathematical argument, i.e., a *proof*. A mathematical statement that we have proven to be true is called a *theorem*. A collection of theorems on a topic organize what we know about this topic. Following and understanding the proof of a theorem is essential to apply its idea to other problems and hence, make it fit new situations.

Proofs are important throughout mathematics, but yet, proofs are also essential for computer science. For example, proofs are used to (a) verify that computer programs produce correct output for all possible input values to a function demonstrating that algorithms always produce the correct result, to (b) establish the security of a system, or to (c) create artificial intelligence; furthermore, (d) automated reasoning systems have been created to allow computers to construct their proofs.

In this chapter, we will introduce the ingredients for a correct mathematical argument and show instruments to construct such arguments. We will show different proof methods that will enable us to prove different types of results. Afterward, we will introduce strategies for constructing proofs. We will provide the notion of conjecture and explain the process of developing mathematics by studying conjectures.

1.1.1 Propositional Logic

A **proposition** is a declarative sentence that is either *true* or *false*, but not both.

All the following declarative sentences are propositions:

(a) Washington, D.C., is the capital of the United States of America.

(b) Toronto is the capital of Canada.

(c) 1+1 = 2.

(d) 2+2 = 3.

Propositions (a) and (c) are true, whereas (b) and (d) are false.

Why are the following examples not propositions:

(a) What time is it?

(b) Read this carefully.

(c) $x + 1 = 2$.

(d) $x + y = z$.

Sentences (a) and (b) are not propositions because they are not declarative sentences. Sentences (c) and (d) are not propositions because they are neither true nor false. However, each of the sentences (c) and (d) can be turned into a proposition if we assign values to the variables.

We use letters to denote propositional variables (or statement variables), i.e., variables that represent propositions. The conventional letters used for propositional variables are $p, q, r, s, \ldots$. The **truth value** of a proposition is true, denoted by T, if it is a true proposition, and the truth value of a proposition is false, denoted by F if it is a false proposition.

The field of logic that deals with propositions is called **propositional logic**. It was first developed systematically by the Greek philosopher Aristotle more than 2,300 years ago.

Let's look at methods for producing new propositions from the ones that we already have. These methods were discussed by the English mathematician George Boole in 1854 in his book *The Laws of Thought*. Many mathematical statements are constructed by combining one or more existing propositions; new propositions, called **compound propositions**, are formed from existing propositions using *logical operators*.

Definition 1. *Let p be a proposition. The negation of p, denoted by $\neg p$ (also $\sim p$), is the statement "It is not the case that p."*

The proposition $\neg p$ is read "not p". The truth value of the negation of p, $\neg p$, is the opposite of the truth value of p.

Definition 2. *Let p and q be propositions. The conjunction of p and q, denoted by $p \wedge q$, is the proposition "p and q". The conjunction $p \wedge q$ is true when both p and q are true and is false otherwise. The $\wedge$ operator for the two propositions p and q can also be understood as $min(p, q)$ where true is represented with 1 and false is represented with 0.*

p	q	$p \wedge q$	$\neg(p \wedge q)$
T	T	T	F
T	F	F	T
F	T	F	T
F	F	F	T

Example 1.1.1. Find the conjunction of the propositions p and q where p is the proposition "Rebecca's PC has more than 16 GB free hard disk space" and q is the proposition "The processor in Rebecca's PC runs faster than 1 GHz."

Solution: The conjunction of these propositions, $p \wedge q$, is the proposition"Rebecca's PC has more than 16 GB free hard disk space, and the processor in Rebeccas PC runs faster than 1 GHz." This conjunction can be expressed more simply as "Rebeccas PC has more than 16 GB free hard disk space, and its processor runs faster than 1 GHz." For this conjunction to be true, both conditions given must be true. It is false when one or both of these conditions is false.

Definition 3. *Let p and q be propositions. The disjunction of p and q, denoted by $p \vee q$, is the proposition "p or q." The disjunction $p \vee q$ is false when both p and q are false and is true otherwise. The $\vee$ operator for the two propositions p and q can also be understood as $max(p, q)$ where true is represented with 1 and false is represented with 0.*

p	q	$p \vee q$	$\neg(p \vee q)$
T	T	T	F
T	F	T	F
F	T	T	F
F	F	F	T

The usage of the connective *or* in a disjunction corresponds to one of the two ways the word *or* is used in English, namely, in an *inclusive* way. Thus, a disjunction is true when at least one of the two propositions therein is true. However, sometimes we need to use *or* in an *exclusive* sense. When the exclusive *or* is used to connect the propositions p and q, the proposition "p or q (but not both)" is obtained. This proposition is true when p is true and q is false, and when p is false and q is true. It is false when both p and q are false and when both are true.

Definition 4. *Let p and q be propositions. The* exclusive or *of p and q, denoted by $p \oplus q$, is the proposition that is true when exactly one of p and q is true and is false otherwise. The $\oplus$ operator for the two propositions p and q can also be understood as $sum(p, q) == 1$ where true is represented with* 1 *and false is represented with* 0.

p	q	$p \oplus q$	$\neg(p \oplus q)$
T	T	F	T
T	F	T	F
F	T	T	F
F	F	F	T

Definition 5. *Let p and q be propositions. The* ***conditional statement*** *$p \implies q$ is the proposition "if p, then q." The conditional statement $p \implies q$ is false when p is true and q is false, and true otherwise. In the conditional statement $p \implies q$, p is called the hypothesis (or antecedent or premise), and q is called the conclusion (or consequence). The $\implies$ operator for the two propositions p and q can also be understood as $max(\neg(p), q)$ where true is represented with* 1 *and false is represented with* 0.

p	q	$p \implies q$	$\neg(p \implies q)$
T	T	T	F
T	F	F	T
F	T	T	F
F	F	T	F

The statement $p \implies q$ is called a conditional statement because $p \implies q$ asserts that q is true on the condition that p holds. A conditional statement is also called an implication. The truth table for the conditional statement $p \implies q$ is shown in table above. Note that the statement $p \implies q$ is true when both p and q are true and when p is false (no matter what the truth value of q is).

Example 1.1.2. Let p be the statement "Maria learns discrete mathematics" and q be the statement "Maria will find a good job." Express the statement $p \implies q$ as a statement in English.

Solution: From the definition of conditional statements, we see that when p is the statement "Maria learns discrete mathematics" and q is the statement "Maria will find a good job," $p \implies q$ represents the statement
"If Maria learns discrete mathematics, then she will find a good job."

There are many other ways to express this conditional statement in English. Among the most natural of these are:
"Maria will find a good job when she learns discrete mathematics."
"For Maria to get a good job, it is sufficient for her to learn discrete mathematics."
and
"Maria will find a good job unless she does not learn discrete mathematics."

CONVERSE, CONTRAPOSITIVE, AND INVERSE Now, we can form new conditional statements starting with a conditional statement $p \implies q$. In particular, there are three related

conditional statements that occur so often that they have special names.

The proposition $q \implies p$ is called the **converse** of $p \implies q$.

The **contrapositive** of $p \implies q$ is the proposition $\neg q \implies \neg p$.

The proposition $\neg p \implies \neg q$ is called the **inverse** of $p \implies q$.

We will see that of these three conditional statements formed from $p \implies q$, only the contrapositive always has the same truth value as $p \implies q$.

Let's look at the contrapositive, $\neg q \implies \neg p$, of a conditional statement $p \implies q$ to investigate its truth value. The contrapositive is false only when $\neg p$ is false and $\neg q$ is true, that is, only when p is true and q is false.

We now show that neither the converse, $q \implies p$, nor the inverse, $\neg p \implies \neg q$, has the same truth value as $p \implies q$ for all possible truth values of p and q. Note that when p is true and q is false, the original conditional statement is false, but the converse and the inverse are both true.

As additional exercise, the reader is advised to calculate the truth tables for **converse**, **contrapositive**, and **inverse** to compare them with the truthtable in Definition 5.

Example 1.1.3. What are the contrapositive, the converse, and the inverse of the conditional statement "The home team wins whenever it is raining?"

Solution: Because "q whenever p" is one of the ways to express the conditional statement $p \implies q$, the original statement can be rewritten as
"If it is raining, then the home team wins."
Consequently, the contrapositive of this conditional statement is
"If the home team does not win, then it is not raining."
The converse is
"If the home team wins, then it is raining."
The inverse is
"If it is not raining, then the home team does not win."
Only the contrapositive is equivalent to the original statement.

Definition 6. *Let p and q be propositions. The biconditional statement $p \iff q$ is the proposition "p if and only if q." The biconditional statement $p \iff q$ is true when p and q have the same truth values, and is false otherwise. Biconditional statements are also called bi-implications. The $\iff$ operator for the two propositions p and q can also be understood as $|(p - q)| == 0$ where true is represented with 1 and false is represented with 0.*

p	q	$p \iff q$	$\sim (p \iff q)$
T	T	T	F
T	F	F	T
F	T	F	T
F	F	T	F

Example 1.1.4. Let p be the statement "You can take the flight," and let q be the statement "You buy a ticket." Then $p \iff q$ is the statement
"You can take the flight if and only if you buy a ticket."
This statement is true if p and q are either both true or both false, that is, if you buy a ticket and can take the flight or if you do not buy a ticket and you cannot take the flight. It is false when p and q have opposite truth values, that is, when you do not buy a ticket, but you can take the flight (such as when you get a free trip) and when you buy a ticket but you cannot take the flight (such as when the airline denies boarding due to overbooked flight).

We have now introduced four important logical connectives – conjunctions, disjunctions, conditional statements, and biconditional statements – as well as negations. We can use these connectives to construct complicated compound propositions involving any number of propositional variables. We can use truth tables to determine the truth values of these compound propositions and logical equivalences as shown in the following.

Definition 7. *A compound proposition that is always true, no matter what the truth values of the propositional variables that occur in it, is called a* ***tautology****. A compound proposition that is always false is called a* ***contradiction****. A compound proposition that is neither a tautology nor a contradiction is called a* ***contingency****.*

Example 1.1.5. We can construct examples of tautologies and contradictions using just one propositional variable. Consider the truth tables of $p \wedge \neg p$ and $p \vee \neg p$: Because $p \vee \neg p$ is always true, it is a tautology; and because $p \wedge \neg p$ is always false, it is a contradiction.

Definition 8. *The compound propositions p and q are called* ***logically equivalent*** *if $p \iff q$ is a tautology. The notation $p \equiv q$ denotes that p and q are logically equivalent.*

Please find a list of truth tables for logical equivalences in Appendix A. The two logical equivalences

known as De Morgan's laws are particularly important. They tell us how to negate conjunctions and how to negate disjunctions. In particular, the equivalence $\neg(p \vee q) \equiv \neg p \wedge \neg q$ tells us that the negation of a disjunction is formed by taking the conjunction of the negations of the component propositions. Similarly, the equivalence $\neg(p \wedge q) \equiv \neg p \vee \neg q$ tells us that the negation of a conjunction is formed by taking the disjunction of the negations of the component propositions.

Example 1.1.6. Use De Morgan's laws to express the negations of "Miguel has a cellphone and he has a laptop computer" and "Heather will go to the concert or Steve will go to the concert."

Solution: Let p be "Miguel has a cellphone" and q be "Miguel has a laptop computer." Then "Miguel has a cellphone and he has a laptop computer" can be represented by $p \wedge q$. Because of the first of De Morgan's laws, we can express the negation of our original statement as
"Miguel does not have a cellphone or he does not have a laptop computer."

Let r be "Heather will go to the concert" and s be "Steve will go to the concert." Then "Heather will go to the concert or Steve will go to the concert" can be represented by $r \vee s$. Because of the second of De Morgans laws, we can express the negation of our original statement as
"Heather will not go to the concert and Steve will not go to the concert."

Example 1.1.7. Show that $\neg(p \vee (\neg p \wedge q))$ and $\neg p \wedge \neg q$ are logically equivalent by developing a series of logical equivalences.

Solution:

$$\begin{aligned}
\neg(p \vee (\neg p \wedge q)) &\equiv \neg p \wedge \neg(\neg p \wedge q) \, by\, the\, second\, De\, Morgan\, law && (1.1)\\
&\equiv \neg p \wedge [\neg(\neg p) \vee \neg q] \, by\, the\, first\, De\, Morgan\, law && (1.2)\\
&\equiv \neg p \wedge (p \vee \neg q) \, by\, the\, double\, negation\, law && (1.3)\\
&\equiv (\neg p \wedge p) \vee (\neg p \wedge \neg q) \, by\, the\, second\, distributive\, law && (1.4)\\
&\equiv F \vee (\neg p \wedge \neg q) \, because\, \neg p \wedge p \equiv F && (1.5)\\
&\equiv (\neg p \wedge \neg q) \vee F \, by\, the\, commutative\, law\, for\, disjunction && (1.6)\\
&\equiv \neg p \wedge \neg q \, by\, the\, identity\, law\, for\, F && (1.7)
\end{aligned}$$

Example 1.1.8. Show that $(p \wedge q) \implies (p \vee q)$ is a tautology.

Solution:
$(p \wedge q) \implies (p \vee q) \equiv \neg(p \wedge q) \vee (p \vee q)$ by logical equivalence (see Appendix A)
$(p \wedge q) \implies (p \vee q) \equiv (\neg p \vee \neg q) \vee (p \vee q)$ by the first De Morgan law)
$(p \wedge q) \implies (p \vee q) \equiv (\neg p \vee p) \vee (\neg q \vee q)$ by the associative and commutative laws for disjunction
$(p \wedge q) \implies (p \vee q) \equiv T \vee T$ by logical equivalence (see Appendix A)
$(p \wedge q) \implies (p \vee q) \equiv T$ by the domination law

A compound proposition is **satisfiable** if there is an assignment of truth values to its variables that makes it true. When no such assignments exists, i.e., when the compound proposition is false for all assignments of truth values to its variables, the compound proposition is **unsatisfiable**. Note that a compound proposition is unsatisfiable if and only if its negation is true for all assignments of truth values to the variables, i.e., if and only if its negation is a tautology. When we find a particular assignment of truth values that makes a compound proposition true, we have shown that it is satisfiable; such an assignment is called a solution of this particular satisfiability problem.

However, to show that a compound proposition is unsatisfiable, we need to show that every assignment of truth values to its variables makes it false. Although we can always use a truth table to determine whether a compound proposition is satisfiable, it is often more efficient not to.

1.1.2 Predicate Logic

Propositional logic cannot adequately express the meaning of all statements in mathematics and in natural language. For example, suppose that we know that
"Every computer connected to the university network is functioning properly"
No rules of propositional logic allow us to conclude the truth of the statement
"MATH3 is functioning properly,"
where MATH3 is one of the computers connected to the university network.

Likewise, we cannot use the rules of propositional logic to conclude from the statement
"CS2 is under attack by an intruder,"
where CS2 is a computer on the university network, to conclude the truth of
"There is a computer on the university network that is under attack by an intruder."

In this section we will introduce a more expressive type of logic called **predicate logic**. We will see how predicate logic can be used to express the meaning of a wide range of statements in mathematics and computer science in ways that permit us to reason and explore relationships between objects. To understand predicate logic, we first need to introduce the concept of a predicate.

Afterwards, we will introduce the notion of quantifiers, which enable us to reason with statements that assert that a certain property holds for all objects of a certain type and with statements that assert the existence of an object with a particular property. The statement $x > 3$ has two parts. The first

part, the variable x, is the subject of the statement. The second part, the predicate, "is greater than 3" refers to a property that the subject of the statement can have. We can denote the statement "x is greater than 3" by $P(x)$, where P denotes the predicate "is greater than 3" and x is the variable. The statement $P(x)$ is also referred to as the value of the propositional function P at x. Once a value has been assigned to the variable x, the statement $P(x)$ becomes a proposition and has a truth value.

Example 1.1.9. Let $Q(x, y)$ denote the statement "$x = y + 3$". What are the truth values of the propositions $Q(1, 2)$ and $Q(3, 0)$?

Solution: To obtain $Q(1, 2)$, set $x = 1$ and $y = 2$ in the statement $Q(x, y)$. Hence, $Q(1, 2)$ is the statement "$1 = 2 + 3$", which is false. The statement $Q(3, 0)$ is the proposition "$3 = 0 + 3$", which is true.

A statement of the form $P(x_1, x_2, ..., x_n)$ is the value of the **propositional function** P at the n-tuple $(x_1, x_2, ..., x_n)$, and P is also called an n-place predicate or a **n-ary predicate**. When the variables in a

propositional function are assigned values, the resulting statement becomes a proposition with a certain truth value. However, there is another important way, called **quantification**, to create a proposition from a propositional function. Quantification expresses the extent, to which a predicate is true over a range of elements. In English, the words *all*, *some*, *many*, *none*, and *few* are used in quantifications. We will focus on two types of quantification here: *universal quantification*, which tells us that a predicate

is true for *every* element under consideration, and *existential quantification*, which tells us that there is *one or more* element under consideration, for which the predicate is true. The field of logic that deals with predicates and quantifiers is called the **predicate calculus**.

Definition 9. *The universal quantification of $P(x)$ is the statement*

"$P(x)$ for all values of x in the domain."

The notation $\forall x P(x)$ denotes the universal quantification of $P(x)$. Here $\forall$ is called the universal quantifier. We read $\forall x P(x)$ as "for all $xP(x)$". An element, for which $P(x)$ is false, is called a counterexample of $\forall x P(x)$.

Example 1.1.10. (a) What is the truth value of $\forall x(x^2 \geq x)$ if the domain consists of all real numbers? (b) What is the truth value of this statement if the domain consists of all integers?

Solution: (a) The universal quantification $\forall x(x^2 \geq x)$, where the domain consists of all real numbers, is false. In this case, a counterexample is sufficient to falsify the claim: For example, $(\frac{1}{2})^2 \not\geq \frac{1}{2}$. Note that $x^2 \geq x$ if and only if $x^2 - x = x * (x-1) \geq 0$. Consequently, $x^2 \geq x$ if and only if $x \leq 0$ or $x \geq 1$. It follows that $\forall x * (x^2 \geq x)$ is false if the domain consists of all real numbers because the inequality is false for all real numbers x within the interval $0 < x < 1$.

(b) However, if the domain consists of the integers, $\forall x(x^2 \geq x)$ is true because there are no integers x within the interval $0 < x < 1$.

Definition 10. *The existential quantification of P(x) is the proposition*

"There exists an element x in the domain such that P(x) evaluates to true."

We use the notation $\exists$ xP(x) for the existential quantification of P(x). Here $\exists$ is called the existential quantifier.

Example 1.1.11. What is the truth value of $\exists x P(x)$, where $P(x)$ is the statement "$x^2 > 10$" and the universe of discourse consists of the positive integers not exceeding 4?

Solution: Because the domain is $\{1, 2, 3, 4\}$, the proposition $\exists x P(x)$ is the same as the disjunction $P(1) \vee P(2) \vee P(3) \vee P(4)$. Because $P(4)$, which is the statement "$4^2 > 10$", is true, it follows that $\exists x P(x)$ is true.

Example 1.1.12. Consider these statements, of which the first three are premises and the fourth is a valid conclusion. "All hummingbirds are richly colored."
"No large birds live on honey."
"Birds that do not live on honey are dull in color."
"Hummingbirds are small."
Let $P(x), Q(x), R(x)$, and $S(x)$ be the statements "x is a hummingbird," "x is large," "x lives on honey," and "x is richly colored," respectively. Assuming that the domain consists of all birds, express the statements in the argument using quantifiers and $P(x), Q(x), R(x)$, and $S(x)$.

Solution: We can express the statements in the argument as
$\forall x(P(x) \implies S(x))$.
$\neg\exists x(Q(x) \wedge R(x))$.
$\forall x(\neg R(x) \implies \neg S(x))$.
$\forall x(P(x) \implies \neg Q(x))$.

Note we have assumed that "small" is the same as "not large" and that "dull in color" is the same as "not richly colored".

ARGUMENTS, RULES OF INFERENCE An **argument** in propositional logic is a sequence of propositions. All but the final proposition in the argument are called **premises** and the final proposition is called the **conclusion**. An argument is **valid** if the truth of all its premises implies that the conclusion is true.

We can always use a truth table to show that an argument is valid. We do this by showing that whenever the premises are true, the conclusion must also be true. However, this can be a tedious approach. For example, when an argument involves ten different propositional variables, using a truth table to show this argument is valid requires $2^{10} = 1,024$ different rows.

Fortunately, we do not have to resort to truth tables. Instead, we can first establish the validity of some relatively simple arguments, called **rules of inference**. These rules of inference can be used as building blocks to construct more complicated valid argument.

1.2 Supervision session exercises

This Section includes all exercises that will be focused on during this week's supervision session.

Exercise 1.1. Suppose that during the most recent fiscal year, the annual revenue of Acme Computer was 138 billion dollars and its net profit was 8 billion dollars, the annual revenue of Nadir Software was 87 billion dollars and its net profit was 5 billion dollars, and the annual revenue of Quixote Media was 111 billion dollars and its net profit was 13 billion dollars. Determine the truth value of each of these propositions for the most recent fiscal year.

a) Quixote Media had the largest annual revenue.

b) Nadir Software had the lowest net profit and Acme Computer had the largest annual revenue.

c) Acme Computer had the largest net profit or Quixote Media had the largest net profit.

d) If Quixote Media had the smallest net profit, then Acme Computer had the largest annual revenue.

e) Nadir Software had the smallest net profit if and only if Acme Computer had the largest annual revenue.

Exercise 1.2. Let p and q be the propositions
p: You drive over 65 miles per hour.
q: You get a speeding ticket.
Write these propositions using p and q and logical connectives (including negations).

a) You do not drive over 65 miles per hour.

b) You drive over 65 miles per hour, but you do not get a speeding ticket.

c) You will get a speeding ticket if you drive over 65 miles per hour.

d) If you do not drive over 65 miles per hour, then you will not get a speeding ticket.

e) Driving over 65 miles per hour is sufficient for getting a speeding ticket.

f) You get a speeding ticket, but you do not drive over 65 miles per hour.

g) Whenever you get a speeding ticket, you are driving over 65 miles per hour.

Exercise 1.3. Construct a truth table for each of the following compound propositions.

a) $p \to (\neg q \vee r)$

b) $\neg p \to (q \to r)$

c) $(p \to q) \vee (\neg p \to r)$

d) $(p \to q) \wedge (\neg p \to r)$

e) $(p \leftrightarrow q) \vee (\neg q \leftrightarrow r)$

f) $(\neg p \leftrightarrow \neg q) \leftrightarrow (q \leftrightarrow r)$

Exercise 1.4. Explain, without using a truth table, why $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ is true when at least one of p, q, and r is true and at least one is false, but is false when all three variables have the same truth value.

Exercise 1.5. Show that $(p \to q) \to r$ and $p \to (q \to r)$ are not logically equivalent.

Exercise 1.6. How many of the disjunctions $p \vee \neg q \vee s$, $\neg p \vee \neg r \vee s$, $\neg p \vee \neg r \vee \neg s$, $\neg p \vee q \vee \neg s$, $q \vee r \vee \neg s$, $q \vee \neg r \vee \neg s$, $\neg p \vee \neg q \vee \neg s$, $p \vee r \vee s$, and $p \vee r \vee \neg s$ can be made simultaneously true by an assignment of truth values to p, q, r, and s?

Solution 1.1. a) F

b) T

c) T

d) T

e) T

Solution 1.2. a) $\neg p$

b) $p \wedge \neg q$

c) $p \rightarrow q$

d) $\neg p \rightarrow \neg q$

e) $p \rightarrow q$

f) $q \wedge \neg p$

g) $q \rightarrow p$

Solution 1.3. Truth tables in Figure 1.1.

37.

p	q	r	$p \rightarrow (\neg q \vee r)$	$\neg p \rightarrow (q \rightarrow r)$	$(p \rightarrow q) \vee (\neg p \rightarrow r)$	$(p \rightarrow q) \wedge (\neg p \rightarrow r)$	$(p \leftrightarrow q) \vee (\neg q \leftrightarrow r)$	$(\neg p \leftrightarrow \neg q) \leftrightarrow (q \leftrightarrow r)$
T	T	T	T	T	T	T	T	T
T	T	F	F	T	T	T	T	F
T	F	T	T	T	T	F	T	T
T	F	F	T	T	T	F	F	F
F	T	T	T	T	T	T	F	F
F	T	F	T	F	T	F	T	T
F	F	T	T	T	T	T	T	F
F	F	F	T	T	T	F	T	T

Figure 1.1: Truth tables for Exercise 1.3.

Solution 1.4. The first clause is true if and only if at least one of p, q, and r is true. The second clause is true if and only if at least one of the three variables is false. Therefore the entire statement is true if and only if there is at least one T and one F among the truth values of the variables, in other words, that they dont all have the same truth value.

Solution 1.5. These are not logically equivalent because when p, q, and r are all false, $(p \rightarrow q) \rightarrow r$ is false, but $p \rightarrow (q \rightarrow r)$ is true.

Solution 1.6. All 9. Draw truth tables for all possible combinations of p,q,r,s to see it.

Chapter 2

Languages and Grammars, Introduction to Automata

2.1 Languages and Grammars

Words in the English language can be combined in various ways. The grammar of English tells us whether a combination of words is a valid sentence. For instance, the frog writes neatly is a valid sentence because it is formed from a noun phrase, *the frog*, made up of the article *the* and the noun *frog*, followed by a verb phrase, *writes neatly*, made up of the verb *writes* and the adverb *neatly*. We do not care that this is a nonsensical statement, because we are concerned only with the syntax (or the form) of the sentence, and not its semantics (or its meaning). We also note that the combination of words "swims quickly mathematics" is not a valid sentence because it does not follow the rules of English grammar.

The syntax of a natural language, i.e., a spoken language such as English, French, German, or Spanish is extremely complicated. In fact, it does not seem possible to specify all rules of syntax for a natural language. Research in the automatic translation of one language to another has led to the concept of a formal language, which, unlike a natural language, is specified by a well-defined set of rules of syntax. Rules of syntax are important not only in linguistics (the study of natural languages) but also in the study of programming languages.

We will describe the sentences of a formal language using a grammar. The use of grammars helps when we consider the two classes of problems that arise most frequently in applications to programming languages: (a) How can we determine whether a combination of words is a valid sentence in a formal language? (b) How can we generate valid sentences of a formal language?

Before giving a technical definition of a grammar, we will describe an example of a grammar that generates a subset of English. This subset of English is defined using a list of rules that describe how a valid sentence can be produced as follows

1. a **sentence** is made up of a **noun phrase** followed by a **verb phrase**;
2. a **noun phrase** is made up of an **article** followed by an **adjective** followed by a **noun**, or
3. a **noun phrase** is made up of an **article** followed by a **noun**;
4. a **verb phrase** is made up of a **verb** followed by an **adverb**, or
5. a **verb phrase** is made up of a **verb**;
6. an **article** is *a*, or
7. an **article** is *the*;
8. an **adjective** is *large*, or

9. an **adjective** is *hungry*;
10. a **noun** is *rabbit*, or
11. a **noun** is *mathematician*;
12. a **verb** is *eats*, or
13. a **verb** is *hops*;
14. an **adverb** is *quickly*, or
15. an **adverb** is *wildly*.

From these rules we can form valid sentences using a series of replacements until no more rules can be used. For instance, we can follow the sequence of replacements: **sentence**
noun phrase verb phrase
article adjective noun verb phrase
article adjective noun verb adverb
the **adjective noun verb adverb**
the large **noun verb adverb**
the large rabbit **verb adverb**
the large rabbit hops **adverb**
the large rabbit hops quickly
to obtain a valid sentence. It is also easy to see that some other valid sentences are: *a hungry mathematician eats wildly, a large mathematician hops, the rabbit eats quickly,* and so on. Also, we can see that *the quickly eats mathematician* is not a valid sentence.

2.1.1 Phrase-structure Grammars

Before we give a formal definition of a grammar, we introduce the necessary terminology.

Definition 11. *A **vocabulary** (or **alphabet**) V is a finite, nonempty set of elements called **symbols**. A word (or **sentence**) over V is a string of finite length of elements of V. The **empty string** or **null string**, denoted by $\boldsymbol{\lambda}$, is the string containing no symbols. The set of all words over V is denoted by $\boldsymbol{V^*}$. A language over V is a subset of $\boldsymbol{V^*}$.*

Note that λ is the string containing no symbols. It is different from ϕ, the empty set. It follows that $\{\lambda\}$ is the set containing exactly one string, namely, the empty string.

Languages can be specified in various ways. One way is to list all the words in the language. Another is to give some criteria that a word must satisfy to be in the language. In this section, we describe another important way to specify a language, namely, through the use of a grammar, similar to the set of rules we gave in the introduction to this section.

A grammar provides a set of symbols of various types and a set of rules for producing words: A grammar has a **vocabulary** V, which is a set of symbols used to derive members of the language. Some of the elements of the vocabulary cannot be replaced further by other symbols; these elements are called **terminals**. Other members of the vocabulary, which can be replaced further by other symbols (**terminals** or **nonterminals**), are called **nonterminals**. The sets of terminals and nonterminals are usually denoted by T and N, respectively. In the example given in the introduction of the section, the set of terminals is $a, the, rabbit, mathematician, hops, eats, quickly, wildly$, and the set of nonterminals is {**sentence, noun phrase, verb phrase, adjective, article, noun, verb, adverb**}.

There is a special member of the vocabulary called the **start symbol**, denoted by S, which is the element of the vocabulary that we always begin with our replacements. In the example in the introduction, the start symbol is **sentence**.

The rules that specify when we can replace a string from V^* (the set of all strings of elements in the vocabulary) with another string are called the **productions** of the grammar. We denote by $z_0 \to z_1$ the production that specifies that z_0 can be replaced by z_1 within a string. The productions in the grammar

given in the introduction of this section were listed; the first production, written using this notation, is **sentence → noun phrase verb phrase**.

Definition 12. *A phrase-structure grammar G = (V, T, S, P) consists of a* vocabulary V*, a subset* T *of V consisting of* terminal symbols*, a* start symbol S ∈ *V, and a finite set of* productions P*. The set* V\T *is denoted by* N*. Elements of N are called* nonterminal symbols*. Every production in P* must *contain at least one nonterminal on its left side.*

Definition 13. *Let $G = (V, T, S, P)$ be a phrase-structure grammar. Let $w_0 = lz_0r$, i.e., the concatenation of l, z_0, and r, and $w_1 = lz_1r$ be strings over V. If $z_0 \to z_1$ is a production of G, we say that w_1 is directly derivable from w_0 and we write $w_0 \Rightarrow w_1$.*

If $w_0, w_1, \ldots, w_n$ are strings over V such that $w_0 \Rightarrow w_1, w_1 \Rightarrow w_2, \ldots, w_{n-1} \Rightarrow w_n$, then we say that w_n is derivable from w_0, and we write $w_0 \overset{*}{\Rightarrow} w_n$. The sequence of steps used to obtain w_n from w_0 is called a derivation*.*

Example 2.1.1. Let $G = (V, T, S, P)$, where $V = \{a, b, A, B, S\}, T = \{a, b\}, withS$ as the start symbol, and $P = \{S \to ABa, A \to BB, B \to ab, AB \to b\}$.

G is an example of a phrase-structure grammar. We are interested in the words that can be generated by the productions of such a phrase structure grammar. The string $Aaba$ is directly derivable from ABa because $B \to ab$ is a production in the grammar. The string $abababa$ is derivable from ABa because $ABa \Rightarrow Aaba \Rightarrow BBaba \Rightarrow Bababa \Rightarrow abababa$, using the productions $B \to ab, A \to BB, B \to ab, andB \to ab$ in sequence.

Definition 14. *Let $G = (V, T, S, P)$ be a phrase-structure grammar. The language generated by G, denoted by $L(G)$, is the set of all strings of terminals that are derivable from the starting state S; in short: $L(G) = \{w \in T^* | S \overset{*}{\Rightarrow} w\}$.*

In the following examples 2.1.2 and 2.1.3 we will define such a language.

Example 2.1.2. Let G be the grammar with vocabulary $V = \{S, A, a, b\}$, set of terminals $T = \{a, b\}$, starting symbol S, and productions $P = \{S \to aA, S \to b, A \to aa\}$. What is the language of the grammar $L(G)$?

Solution: From the start state S we can derive aA using the production $S \to aA$. We can also use the production $S \to b$ to derive b. From aA the production $A \to aa$ can be used to derive aaa. No additional words can be derived. Hence, $L(G) = \{b, aaa\}$.

Example 2.1.3. Let G be the grammar with vocabulary $V = \{S, 0, 1\}$, set of terminals $T = \{0, 1\}$, starting symbol S, and productions $P = \{S \to 11S, S \to 0\}$. What is $L(G)$, the language of this grammar?

Solution: From S we can derive 0 using $S \to 0$, or $11S$ using $S \to 11S$. From $11S$ we can derive either 110 or $1111S$. From $1111S$ we can derive 11110 and $111111S$. At any stage of a derivation we can either add 11 at the end of the string followed by S or terminate the derivation by adding a final 0. We note that $L(G) = \{0, 110, 11110, 1111110, \ldots\}$, the set of all strings that begin with an even number of 1 and end with a 0.

This can even be proven using an inductive argument that shows that after n productions have been used, the only strings of terminals generated are those consisting of $n - 1$ concatenations of 11 followed by 0; this is left as an exercise for the reader.

The problem of constructing a grammar that generates a given language often arises. Examples 2.1.4, 2.1.5 and 2.1.6 describe problems of this kind.

Example 2.1.4. Provide a phrase-structure grammar that generates the set $\{0^n1^n|n = 0, 1, 2, \dots\}$.

Solution: Two productions can be used to generate all strings consisting of a string of 0 followed by a string of the same number of 1, including the null string. The first production builds up successively longer strings in the language by adding a 0 at the start of the string and a 1 at its end. The second production replaces S with the empty string. The solution is the grammar $G = (V, T, S, P)$, where $V = \{0, 1, S\}, T = \{0, 1\}, S$ is the starting symbol, and the productions are
$S \to 0S1$
$S \to \lambda$
The verification that this grammar generates the correct set is left as an exercise for the reader.

Example 2.1.5. Find a phrase-structure grammar to generate the set $\{0^m1^n|m$ and n are non-negative integers$\}$.

Solution: We will give two grammars $G1$ and $G2$ that generate this set. This will illustrate that two grammars can generate the same language.

The grammar $G1$ has the alphabet $V = \{S, 0, 1\}$; terminals $T = \{0, 1\}$; and productions $S \to 0S$, $S \to S1$, and $S \to \lambda$. $G1$ generates the correct set, because using the first production m times puts $m \times 0$ at the beginning of the string, and using the second production n times puts $n \times 1$ at the end of the string. The details of this verification are left to the reader.

The grammar $G2$ has alphabet $V = \{S, A, 0, 1\}$; terminals $T = \{0, 1\}$; and productions $S \to 0S, S \to 1A, S \to 1, A \to 1A, A \to 1$, and $S \to \lambda$. The details that this grammar generates the correct set are left as an exercise for the reader.

Sometimes a set that is easy to describe can be generated only by a complicated grammar, as illustrated by the following example.

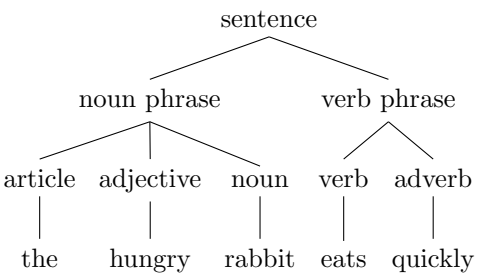
Example 2.1.6. One grammar that generates the set $\{0^n1^n2^n|n = 0, 1, 2, 3, \dots\}$ is $G = (V, T, S, P)$ with $V = \{0, 1, 2, S, A, B, C\}; T = \{0, 1, 2\}$; starting state S; and productions $S \to C, C \to 0CAB, C \to \lambda, BA \to AB, 0A \to 01, 1A \to 11, 1B \to 12$, and $2B \to 22$. We leave it as an exercise for the reader to show that this statement is correct.

2.1.2 Derivation Trees

A derivation in the language generated by a context-free grammar can be represented graphically using an ordered rooted tree, called a **derivation** or **parse tree**. The root of this tree represents the starting symbol. The internal vertices of the tree represent the nonterminal symbols that arise in the derivation. The leaves of the tree represent the terminal symbols that arise. If the production $A \to w$ arises in the derivation, where w is a word, the vertex that represents A has as children vertices that represent each symbol in w, in order from left to right.

Example 2.1.7. Construct a derivation tree for the derivation of the hungry rabbit eats quickly, given in the introduction of this chapter.

Solution: The derivation tree is shown in Figure 2.1.2.



Example 2.1.8. Determine whether the word *cbab* belongs to the language generated by the grammar $G = (V, T, S, P)$, where $V = \{a, b, c, A, B, C, S\}, T = \{a, b, c\}, S$ is the starting symbol, and the productions are
$S \to AB$
$A \to Ca$
$B \to Ba$
$B \to Cb$
$B \to b$
$C \to cb$
$C \to b$
Solution: One way to approach this problem is to begin with S and attempt to derive *cbab* using a series of productions. Because there is only one production with S on its left-hand side, we must start with $S \Rightarrow AB$. Next we use the only production that has A on its left-hand side, namely, $A \to Ca$, to obtain $S \Rightarrow AB \Rightarrow CaB$. Because *cbab* begins with the symbols *cb*, we use the production $C \to cb$. This provides us $S \Rightarrow AB \Rightarrow CaB \Rightarrow cbaB$. We finish by using the production $B \to b$, to obtain $S \Rightarrow AB \Rightarrow CaB \Rightarrow cbaB \Rightarrow cbab$. The approach that we have used is called *top-down parsing*, because it begins with the starting symbol and proceeds by successively applying productions.

There is another approach to this problem, called bottom-up parsing. In this approach, we work backward. Because *cbab* is the string to be derived, we can use the production $C \to cb$, so that $Cab \Rightarrow cbab$. Then, we can use the production $A \to Ca$, so that $Ab \Rightarrow Cab \Rightarrow cbab$. Using the production $B \to b$ gives $AB \Rightarrow Ab \Rightarrow Cab \Rightarrow cbab$. Finally, using $S \to AB$ shows that a complete derivation for *cbab* is $S \Rightarrow AB \Rightarrow Ab \Rightarrow Cab \Rightarrow cbab$.

2.2 Finite-State Machine with Output

Many kinds of machines, including components in computers, can be modeled using a structure called a finite-state machine. Several types of finite-state machines are commonly used in models. All versions of finite-state machines include a finite set of states, with a designated starting state, an input alphabet, and a transition function that assigns a next state to every state and input pair. Finite-state machines are used extensively in applications in computer science and data networking. For example, finite-state machines are the basis for spelling checkers, grammar checking, indexing or searching large bodies of text, recognizing speech, transforming text using markup languages such as XML and HTML, and network protocols that specify how computers communicate.

In this section, we will study finite-state machines that produce output. We will show how finite-state machines can be used to model a vending machine, a machine that delays input, a machine that adds integers, and a machine that determines whether a bit string contains a specified pattern.

Before giving formal definitions, we will show how a vending machine can be modeled. A vending machine accepts nickels (5 cents), dimes (10 cents), and quarters (25 cents). When a total of 30 cents or more has been deposited, the machine immediately returns the amount in excess of 30 cents. When 30 cents has been deposited and any excess refunded, the customer can push an orange button and receive an orange juice or push a red button and receive an apple juice. We can describe how the machine works by specifying its states, how it changes states when input is received, and the output that is produced for every combination of input and current state.

The machine can be in any of seven different states $s_i, i = 0, 1, 2, \ldots, 6$, where s_i is the state where the machine has collected $5 \times i$ cents. The machine starts in state s_0, with 0 cents received. The possible inputs are 5 cents, 10 cents, 25 cents, the orange button (O), and the red button (R). The possible outputs are nothing (n), 5 cents, 10 cents, 15 cents, 20 cents, 25 cents, an orange juice, and an apple juice.

We illustrate how this model of the machine works with this example. Suppose that a student puts in a dime followed by a quarter, receives 5 cents back, and then pushes the orange button for an orange juice. The machine starts in state s_0. The first input is 10 cents, which changes the state of the machine to s_2 and gives no output. The second input is 25 cents. This changes the state from s_2 to s_6, and gives

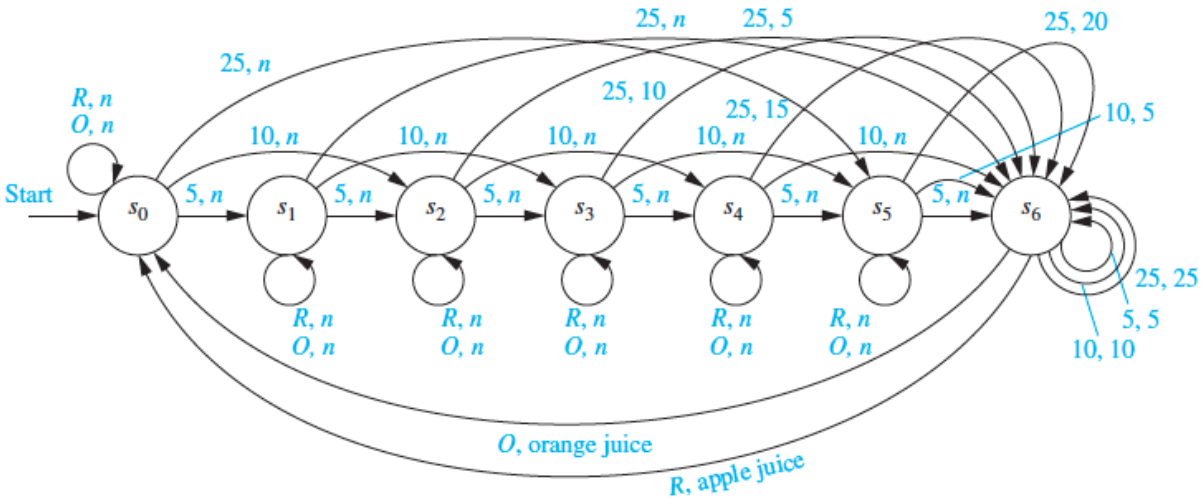


Figure 2.1: A directed graph for the vending machine.

5 cents as output. The next input is the orange button, which changes the state from s_6 back to s_0 (because the machine returns to the start state) and gives an orange juice as its output.

We can describe all state changes and output of this machine in a table. To do this we need to specify for each combination of state and input the next state and the output obtained. Tab. 2.1 shows the transitions and outputs for each pair of a state and an input.

	Next State					Output				
	Input					*Input*				
State	5	10	25	O	R	5	10	25	O	R
s_0	s_1	s_2	s_5	s_0	s_0	n	n	n	n	n
s_1	s_2	s_3	s_6	s_1	s_1	n	n	n	n	n
s_2	s_3	s_4	s_6	s_2	s_2	n	n	5	n	n
s_3	s_4	s_5	s_6	s_3	s_3	n	n	10	n	n
s_4	s_5	s_6	s_6	s_4	s_4	n	n	15	n	n
s_5	s_6	s_6	s_6	s_5	s_5	n	5	20	n	n
s_6	s_6	s_6	s_6	s_0	s_0	5	10	25	OJ	AJ

Table 2.1: State Table for a Vending Machine.

Another way to show the actions of a machine is to use a directed graph with labeled edges, where each state is represented by a circle, edges represent the transitions, and edges are labeled with the input and the output for that transition. Figure 2.1 shows such a directed graph for the vending machine.

Definition 15. *A finite-state machine $M = (S, I, O, f, g, s_0)$ consists of a finite set S of states, a finite input alphabet I, a finite output alphabet O, a transition function f that assigns to each pair of state and input a new state, an output function g that assigns to each pair of state and input a corresponding output, and an initial state s_0.*

We can use a **state table** to represent the values of the transition function f and the output function g for all pairs of states and input. We have already seen a state table for the vending machine.

Example 2.2.1. The state table shown in Table 2.3 describes a finite-state machine with $S = \{s_0, s_1, s_2, s_3\}$, $I = \{0, 1\}$, and $O = \{0, 1\}$. The values of the transition function f are displayed in the first two columns, and the values of the output function g are displayed in the last two columns.

Another way to represent a finite-state machine is to use a **state diagram**, which is a directed graph with labeled edges. In this diagram, each state is represented by a circle. Arrows labeled with the

Table 2.2: State tables for two state machines.

	f		g	
	Input		*Input*	
State	0	1	0	1
s_0	s_1	s_0	1	0
s_1	s_3	s_0	1	1
s_2	s_1	s_2	0	1
s_3	s_2	s_1	0	0

Table 2.3: State Table for the state diagram in Figure 2.2.

	f		g	
	Input		*Input*	
State	0	1	0	1
s_0	s_1	s_3	1	0
s_1	s_1	s_2	1	1
s_2	s_3	s_4	0	0
s_3	s_1	s_0	0	0
s_4	s_3	s_4	0	0

Table 2.4: State Table for the state diagram in Figure 2.3.

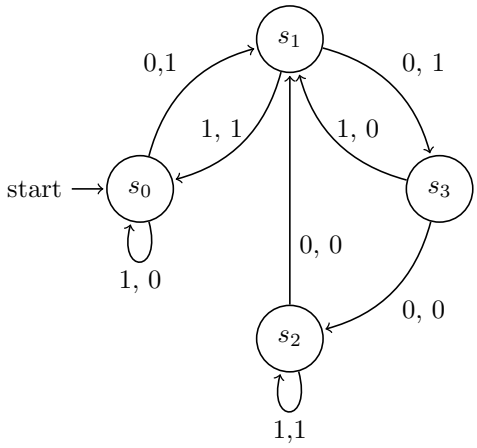


Figure 2.2: State diagram for the state table 2.3.

input and output pair are shown for each transition. Figure 2.2 shows the state diagram corresponding with the state table 2.3. Similarly, the diagram in Figure 2.3 represents the state machine described in Table 2.4.

An input string takes the starting state through a sequence of states, as determined by the transition function. As we read the input string symbol by symbol (from left to right), each input symbol takes the machine from one state to another. Because each transition produces an output, an input string also produces an output string.

Suppose that the input string is $x = x_1x_2 \dots x_k$. Then, reading this input takes the machine from state s_0 to state s_1, where $s_1 = f(s_0, x_1)$, then to state s_2, where $s_2 = f(s_1, x_2)$, and so on, with $s_j = f(s_{j-1}, x_j)$ for $j = 1, 2, \dots, k$, ending at state $s_k = f(s_{k-1}, x_k)$. This sequence of transitions produces an output string $y_1y_2 \dots y_k$, where $y_1 = g(s_0, x_1)$ is the output corresponding to the transition from s_0 to s_1, $y_2 = g(s_1, x_2)$ is the output corresponding to the transition from s_1 to s_2, and so on. In general, $y_j = g(s_{j-1}, x_j)$ for $j = 1, 2, \dots, k$. Hence, we can extend the definition of the output function g to input strings so that $g(x) = y$, where y is the output corresponding to the input string x. This notation is useful in many applications.

Example 2.2.2. Find the output string generated by the finite-state machine in Figure 2.3 if the input string is 101011.

Solution: The output obtained is 001000. The successive states and outputs are shown in Table 2.4.

We can now look at some examples of useful finite-state machines. Examples 2.2.3, 2.2.4, and 2.2.5

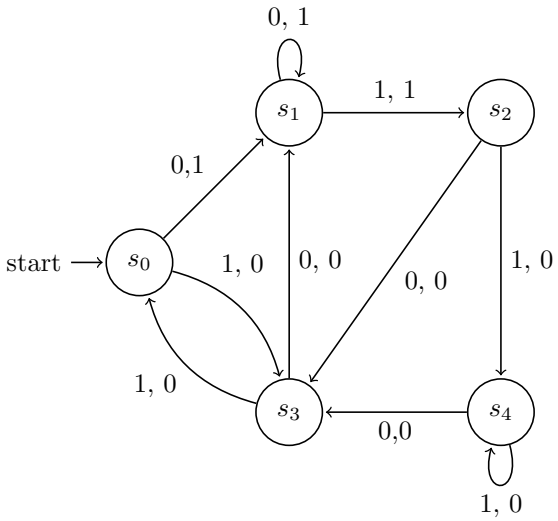


Figure 2.3: State diagram for the state table 2.4.

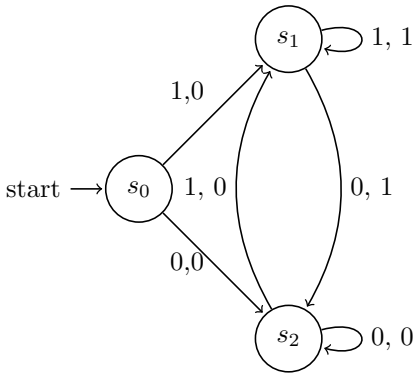


Figure 2.4: State diagram for the unit-delay machine.

illustrate that the states of a finite-state machine give it limited memory capabilities. The states can be used to remember the properties of the symbols that have been read by the machine. However, because there are only finitely many different states, finite-state machines cannot be used for some important purposes.

Example 2.2.3. An important element in many electronic devices is a **unit-delay machine**, which produces as output the input string delayed by a specified amount of time. How can a finite-state machine be constructed that delays an input string by one unit of time, that is, produces as output the bit string $0x_1x_2 \dots x_{k1}$ given the input bit string $x_1x_2...x_k$?

Solution: A delay machine can be constructed that has two possible inputs, namely, 0 and 1. The machine must have a start state s_0. Because the machine has to remember whether the previous input was a 0 or a 1, two other states s_1 and s_2 are needed, where the machine is in state s_1 if the previous input was 1 and in state s_2 if the previous input was 0. An output of 0 is produced for the initial transition from s_0. Each transition from s_1 gives an output of 1, and each transition from s_2 gives an output of 0. The output corresponding to the input of a string $x_1 \dots x_k$ is the string that begins with 0, followed by x_1, followed by $x_2, \dots$, ending with x_{k-1}. The state diagram for this machine is shown in Figure 2.4.

Example 2.2.4. Create a finite-state machine that adds two positive integers using their binary expansions.

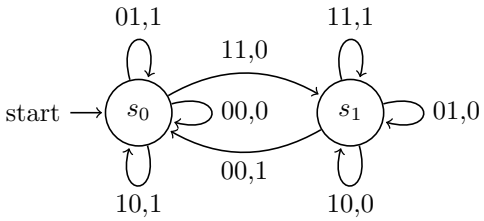


Figure 2.5: A finite-state machine for addition.

Solution: When $(x_n \dots x_1 x_0)_2$ and $(y_n \dots y_1 y_0)_2$ are added, the following procedure is followed. First, the bits x_0 and y_0 are added, producing a sum bit z_0 and a carry bit c_0. This carry bit is either 0 or 1. Then, the bits x_1 and y_1 are added, together with the carry c_0. This gives a sum bit z_1 and a carry bit c_1. This procedure is continued until the n-th stage, where x_n, y_n, and the previous carry c_{n1} are added to produce the sum bit z_n and the carry bit c_n, which is equal to the sum bit z_{n+1}.

A finite-state machine to carry out this addition can be constructed using just two states. For simplicity we assume that both the initial bits x_n and y_n are 0 (otherwise we have to make special arrangements concerning the sum bit z_{n+1}). The start state s_0 is used to remember that the previous carry is 0 (or for the addition of the rightmost bits). The other state, s_1, is used to remember that the previous carry is 1.

Because the inputs to the machine are pairs of bits, there are four possible inputs. We represent these possibilities by 00 (when both bits are 0), 01 (when the first bit is 0 and the second is 1), 10 (when the first bit is 1 and the second is 0), and 11 (when both bits are 1).

The transitions and the outputs are constructed from the sum of the two bits represented by the input and the carry represented by the state. For instance, when the machine is in state s_1 and receives 01 as input, the next state is s_1 and the output is 0, because the sum that arises is $0 + 1 + 1 = (10)_2$. The state diagram for this machine is shown in Figure 2.5.

Example 2.2.5. In a certain coding scheme, when three consecutive 1 appear in a message, the receiver of the message knows that there has been a transmission error. Construct a finite-state machine that gives a 1 as its current output bit if and only if the last three bits received are all 1.

Solution: Three states are needed in this machine. The start state s_0 remembers that the previous input value, if it exists, was not a 1. The state s_1 remembers that the previous input was a 1, but the input before the previous input, if it exists, was not a 1. The state s_2 remembers that the previous two inputs were $1s$.

An input of 1 takes s_0 to s_1, because now a 1, and not two consecutive 1, has been read; it takes s_1 to s_2, because now two consecutive 1 have been read; and it takes s_2 to itself, because at least two consecutive 1 have been read. An input of 0 takes every state to s_0, because this breaks up any string of consecutive 1. The output for the transition from s_2 to itself when a 1 is read is 1, because this combination of input and state shows that three consecutive 1 have been read. All other outputs are 0. The state diagram of this machine is shown in Figure 2.6.

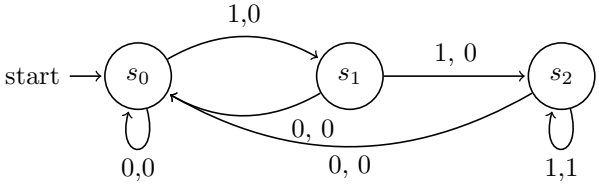


Figure 2.6: A finite-state machine that gives an output of 1 if and only if the input string read so far ends with 111.

The final output bit of the finite-state machine we constructed in this example is 1 if and only if the input string ends with 111. Because of this, we say that this finite-state machine **recognizes** the set of bit strings that end with 111. This leads us to Definition 16.

Definition 16. *Let $M = (S, I, O, f, g, s_0)$ be a finite-state machine and $L \subseteq I^*$. We say that M recognizes (or accepts) L if an input string x belongs to L if and only if the last output bit produced by M when given x as input is a* 1.

TYPES OF FINITE-STATE MACHINES Many different kinds of finite-state machines have been developed to model computing machines. In this section we have given a definition of one type of finite-state machine. In the type of machine introduced in this section, outputs correspond to transitions between states. Machines of this type are known as **Mealy machines**, because they were first studied by G. H. Mealy in 1955.

There is another important type of finite-state machine with output, where the output is determined only by the state. This type of finite-state machine is known as a **Moore machine**, because E. F. Moore introduced this type of machine in 1956. A Moore machine $M = (S, I, O, f, g, s_0)$ consists of a finite set of states, an input alphabet I, an output alphabet O, a transition function f that assigns a next state to every pair of a state and an input, an output function g that assigns an output to every state, and a starting state s_0. A Moore machine can be represented either by a table listing the transitions for each pair of state and input and the outputs for each state, or by a state diagram that shows the states, the transitions between states, and the output for each state. In the diagram, transitions are indicated with arrows labeled with the input, and the outputs are shown next to the states.

In Example 2.2.5 we showed how a Mealy machine can be used for language recognition. However, another type of finite-state machine, giving no output, is usually used for this purpose. Finite-state machines with no output, also known as finite-state automata, have a set of final states and recognize a string if and only if it takes the start state to a final state.

2.3 Finite-State Machine with No Output

One of the most important applications of finite-state machines is in language recognition. This application plays a fundamental role in the design and construction of compilers for programming languages. In the previous section we showed that a finite-state machine with output can be used to recognize a language, by giving an output of 1 when a string from the language has been read and a 0 otherwise.

However, there are other types of finite-state machines that are specially designed for recognizing languages. Instead of producing output, these machines have final states. A string is recognized if and only if it takes the starting state to one of these final states.

Definition 17. *A finite-state automaton $M = (S, I, f, s_0, F)$ consists of a finite set S of states, a finite input alphabet I, a transition function f that assigns a next state to every pair of state and input (so that $f : S \times I \to S$), an initial or start state s_0, and a subset F of S consisting of final state, which are also called* accepting states.

We can represent finite-state automata using either state tables or state diagrams. Final states are indicated in state diagrams by using double circles.

Example 2.3.1. Construct the state diagram for the finite-state automaton $M = (S, I, f, s_0, F)$, where $S = \{s_0, s_1, s_2, s_3\}$, $I = \{0, 1\}$, $F = \{s_0, s_3\}$, and the transition function f is given in Table 2.5.

Solution: The state diagram is shown next to the state table below. Note that because both inputs 0 and 1 take s_2 to s_0, we write 0,1 at the edge from s_2 to s_0.

	f	
	Input	
State	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_0
s_3	s_2	s_1

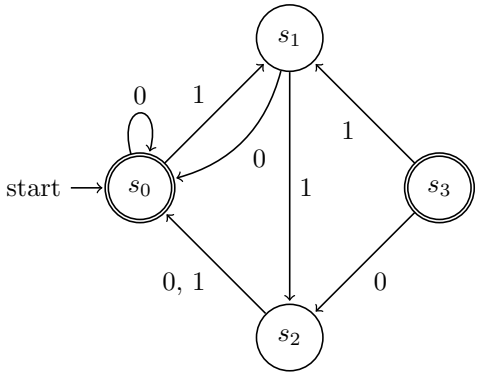


Table 2.5: State table and corresponding state diagram for Example 2.3.1.

2.4 Language Recognition

Definition 18. *A string x is said to be recognized or accepted by the machine $M = (S, I, f, s_0, F)$ if it takes the initial state s_0 to a final state, i.e., $f(s_0, x)$ is a state in F. The language accepted by the machine M as denoted by $L(M)$ is the set of all strings that are recognized by M. Two finite-state automata are called equivalent if they recognize the same language.*

In Example 2.4.1 we will find the languages recognized by several finite-state automata.

Example 2.4.1. Determine the languages recognized by the finite-state automata M_1, M_2, and M_3 in Figure 2.7, 2.8, and 2.9.

Solution: The only final state of M_1 is s_0. The strings that take s_0 to itself are those consisting of zero or more consecutive 1. Hence, $L(M_1) = \{1^n | n = 0, 1, 2, \dots\}$.

The only final state of M_2 is s_2. The only strings that take s_0 to s_2 are 1 and 01. Hence, $L(M_2) = \{1, 01\}$.

The final states of M_3 are s_0 and s_3.The only strings that take s_0 to itself are $\lambda, 0, 00, 000, \dots$, i.e., any string of zero or more consecutive 0. The only strings that take s_0 to s_3 are a string of zero or more consecutive 0, followed by 10, followed by any string. Hence, $L(M_3) = \{0^n, 0^n 10x | n = 0, 1, 2, \dots,$ and x is any string$\}$.

DESIGNING FINITE-STATE AUTOMATA We can often construct a finite-state automaton that recognizes a given set of strings by carefully adding states and transitions and determining, which of these states should be final states. When appropriate we include states that can keep track of some of the properties of the input string, providing the finite-state automaton with limited memory. The following examples illustrate some of the techniques that can be used to construct finite-state automata that recognize particular types of sets of strings.

Example 2.4.2. Construct deterministic finite-state automata that recognize each of these languages:

(a) the set of bit strings that begin with two 0

(b) the set of bit strings that contain two consecutive 0

(c) the set of bit strings that do not contain two consecutive 0

(d) the set of bit strings that end with two 0

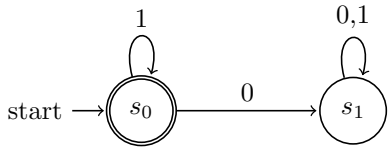


Figure 2.7: State diagram for M_1.

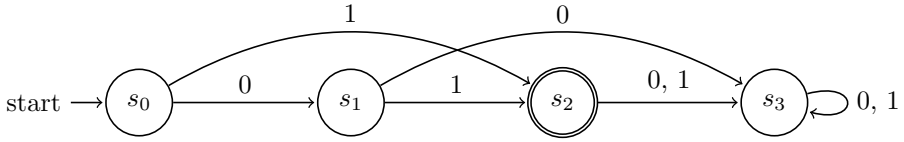


Figure 2.8: State diagram for M_2.

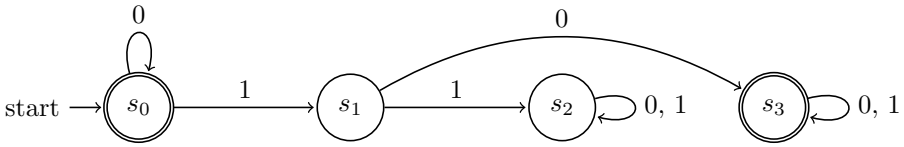


Figure 2.9: State diagram for M_3.

(e) the set of bit strings that contain at least two 0

Solutions:

(a) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that begin with two 0. Besides the start state s_0, we include a non-final state s_1; we move to s_1 from s_0 if the first bit is a 0. Next, we add a final state s_2, which we move to from s_1 if the second bit is a 0. When we have reached s_2 we know that the first two input bits are both 0, so we stay in the state s_2 no matter what the succeeding bits (if any) are.

We move to a non-final state s_3 from s_0 if the first bit is a 1 and from s_1 if the second bit is a 1. The reader should verify that the finite-state automaton in Figure 2.10 recognizes the set of bit strings that begin with two 0*s*.

(b) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that contain two consecutive 0. Besides the start state s_0, we include a non-final state s_1, which tells us that the last input bit seen is a 0, but either the bit before it was a 1, or this bit was the initial bit of the string. We include a final state s_2 that we move to from s_1 when the next input bit after a 0 is also a 0. If a 1 follows a 0 in the string (before we encounter two consecutive 0), we return to s_0 and begin looking for consecutive 0 all over again. The reader should verify that the finite-state automaton in Figure 2.11 recognizes the set of bit strings that contain two consecutive 0s.

(c) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that do not contain two consecutive 0s. Besides the start state s_0, which should be a final state, we include a final state s_1, which we move to from s_0 when 0 is the first input bit. When an input bit is a 1, we return to, or stay in, state s_0.

We add a state s_2, to which we move from s_1 when the input bit is a 0. Reaching s_2 tells us that we have seen two consecutive 0 as input bits. We stay in state s_2 once we have reached it; this state is not final. The reader should verify that the finite-state automaton in Figure 2.12 recognizes the set of bit strings that do not contain two consecutive 0.

(d) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings

that end with two 0s. Besides the start state s_0, we include a non-final state s_1, to which we move if the first bit is 0. We include a final state s_2, which we move to from s_1 if the next input bit after a 0 is also a 0.

If an input of 0 follows a previous 0, we stay in state s_2 because the last two input bits are still 0. Once we are in state s_2, an input bit of 1 sends us back to s_0, and we begin looking for consecutive 0 all over again. We also return to s_0 if the next input is a 1 when we are in state s_1. The reader should verify that the finite-state automaton in Figure 2.13 recognizes the set of bit strings that end with two 0.

(e) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that contain two 0. Besides the start state, we include a state s_1, which is not final; we stay in s_0 until an input bit is a 0 and we move to s_1 when we encounter the first 0 in the input.

We add a final state s_2, to which we move from s_1 once we encounter a second 0 bit. Whenever we encounter a 1 as input, we stay in the current state. Once we have reached s_2, we remain there. Here, s_1 and s_2 are used to tell us that we have already seen one or two 0 in the input string so far, respectively. The reader should verify that the finite-state automaton in Figure 2.14 recognizes the set of bit strings that contain two 0s.

Example 2.4.3. Construct a deterministic finite-state automaton that recognizes the set of bit strings that contain an odd number of 1 and that end with at least two consecutive 0.

Solution: We can construct a deterministic finite-state automaton that recognizes the specified set by including states that keep track of both the parity of the number of 1 bits and whether we have seen no, one, or at least two 0 at the end of the input string.

The start state s_0 can be used to tell us that the input read so far contains an even number of 1 and ends with no 0 (i.e., is empty or ends with a 1). Besides the start state, we include five more states. We move to states s_1, s_2, s_3, s_4, and s_5, respectively, when the input string read so far contains an even number of 1 and ends with one 0; when it contains an even number of 1 and ends with at least two 0; when it contains an odd number of 1 and ends with no 0; when it contains an odd number of 1 and ends with one 0; and when it contains an odd number of 1 and ends with two 0. The state s_5 is a final state. The reader should verify that the finite-state automaton in Figure 2.15 recognizes the set of bit strings that contain an odd number of 1 and end with at least two consecutive 0.

In Definition 18 we specified that two finite-state automata are equivalent if they recognize the same language. Example 2.4.4 provides an example of two equivalent deterministic finite-state machines.

Example 2.4.4. Show that the two finite-state automata M_0(Figure 2.16) and M_1(Figure 2.17) are equivalent.

Solution: For a string x to be recognized by M_0, x must take us from s_0 to the final state s_1 or the final state s_4. The only string that takes us from s_0 to s_1 is the string 1. The strings that take us from s_0 to s_4 are those strings that begin with a 0, which takes us from s_0 to s_2, followed by zero or more additional 0, which keep the machine in state s_2, followed by a 1, which takes us from state s_2 to the final state s_4. All other strings take us from s_0 to a state that is not final, which we leave to the reader to fill in the details. We conclude that $L(M_0)$ is the set of strings of zero or more 0 bits followed by a final 1.

For a string x to be recognized by M_1, x must take us from s_0 to the final state s_1. So, for x to be recognized, it must begin with some number of 0, which leaves us in state s_0, followed by a 1, which takes us to the final state s_1. A string of all zeros is not recognized because it leaves us in state s_0, which is not final. All strings that contain a 0 after 1 are not recognized because they take us to state s_2, which is not final. It follows that $L(M_1)$ is the same as $L(M_0)$. Hence, we conclude that M_0 and M_1 are equivalent.

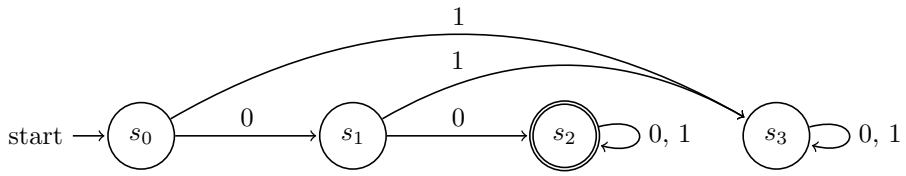


Figure 2.10: State diagram for (a).

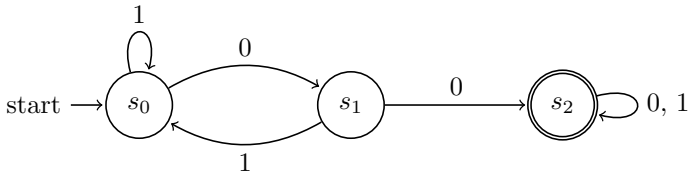


Figure 2.11: State diagram for (b).

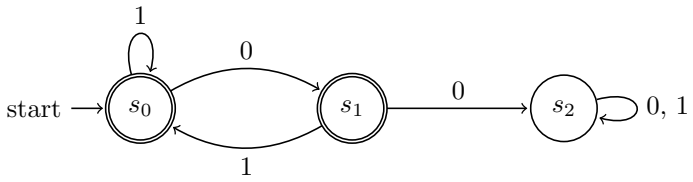


Figure 2.12: State diagram for (c).

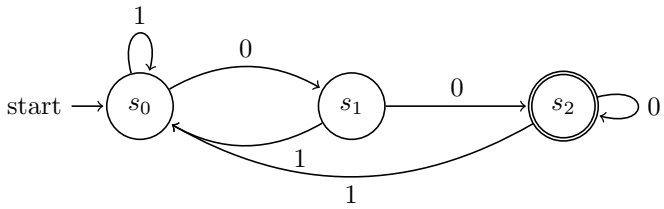


Figure 2.13: State diagram for (d).

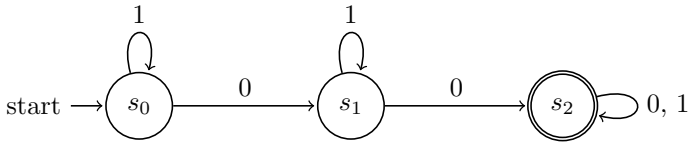


Figure 2.14: State diagram for (e).

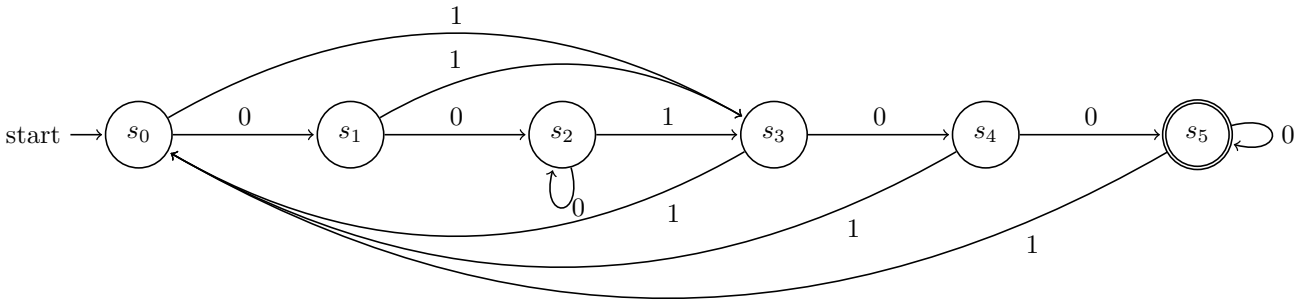


Figure 2.15: State diagram for Example 2.4.3.

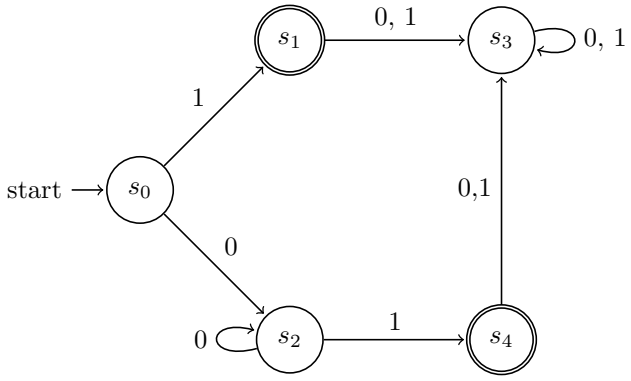


Figure 2.16: Diagram of FSA M_0.

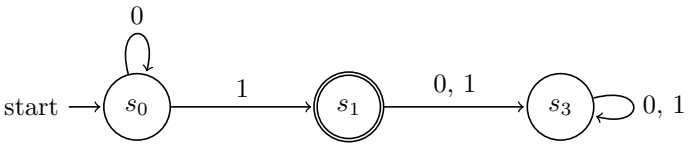


Figure 2.17: Diagram of FSA M_1.

Note that the finite-state machine M_1 only has three states. No finite state machine with fewer than three states can be used to recognize the set of all strings of zero or more 0 bits followed by a 1.

NONDETERMINISTIC FINITE-STATE AUTOMATA The finite-state automata discussed so far are deterministic because for each pair of state and input value there is a unique next state given by the transition function. There is another important type of finite-state automata, in which there may be several possible next states for each pair of input value and state. Such machines are called nondeterministic. Nondeterministic finite-state automata are important in determining which languages can be recognized by a finite-state automaton.

A nondeterministic finite-state automaton $M = (S, I, f, s_0, F)$ consists of a set S of states, an input alphabet I, a transition function f that assigns a set of states to each pair of state and input (i.e., $f : S \times I \to P(S)$), a starting state s_0, and a subset F of S consisting of the final states. The reader can study the behavior of the nondeterministic state machine in the following.

	f	
	Input	
State	0	1
s_0	s_0, s_1	s_3
s_1	s_0	s_1, s_3
s_2		s_0, s_2
s_3	s_0, s_1, s_2	s_1

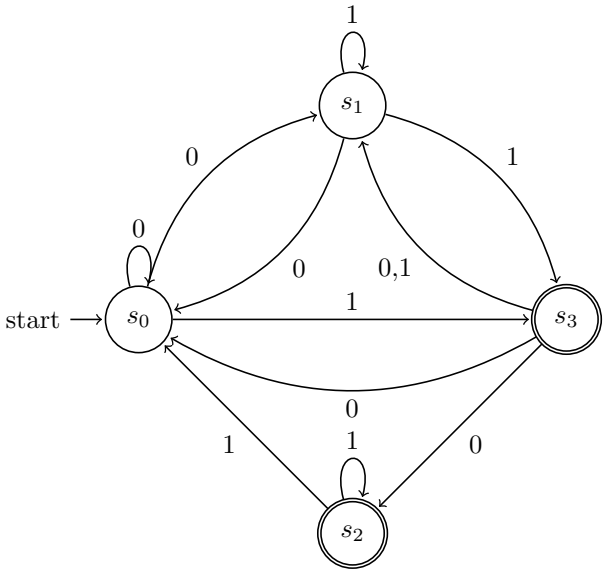


Table 2.6: State table and corresponding state diagram for a nondeterministic finite-state automaton.

2.5 Supervision session exercises

This Section includes all the exercises that will be focused on during this week's supervision session.

Languages and Grammars

Exercise 2.1. Let G be the grammar with $V = a, b, c, S$; $T = a, b, c$; starting symbol S; and productions $S \to abS$, $S \to bcS$, $S \to bbS$, $S \to a$, and $S \to cb$. Construct derivation trees for:

a) bcbba

b) bbbcbba

c) bcabbbbbcb

Exercise 2.2. Find a phrase-structure grammar for each of these languages.

a) the set of all bit strings containing an even number of 0s and no 1s

b) the set of all bit strings made up of a 1 followed by an odd number of 0s

c) the set of all bit strings containing an even number of 0s and an even number of 1s

d) the set of all strings containing 10 or more 0s and no 1s

e) the set of all strings containing more 0s than 1s

f) the set of all strings containing an equal number of 0s and 1s

g) the set of all strings containing an unequal number of 0s and 1s

Exercise 2.3. Give production rules in BackusNaur form for an identifier if it can consist of:

a) one or more lowercase letters.

b) at least three but no more than six lowercase letters.

c) one to six uppercase or lowercase letters beginning with an uppercase letter.

d) a lowercase letter, followed by a digit or an underscore, followed by three or four alphanumeric characters (lower or uppercase letters and digits).

Finite State Machines

Exercise 2.4. Determine whether all the strings in each of these sets are recognized by the deterministic finite-state automaton in Figure below.

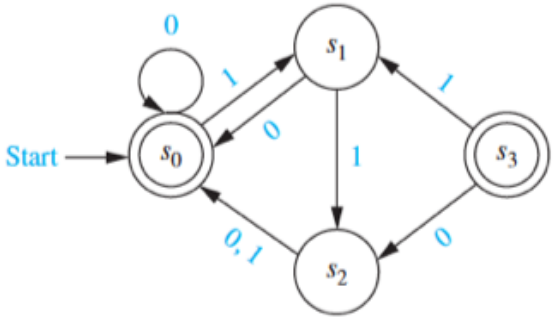


Figure 2.18: Finite State Machine for Exercise 2.4.

a) $\{0\}^*$

b) $\{0\}\{0\}^*$

c) $\{1\}\{0\}^*$

d) $\{01\}^*$

e) $\{0\}^*\{1\}^*$

f) $\{1\}\{0,1\}^*$

Exercise 2.5. Build an electronic combination lock with a reset button, two number buttons (0 and 1), and an unlock output. The combination should be **01011**. Write down the transition table.

Exercise 2.6. Construct a regular grammar $G = (V, T, S, P)$ that generates the language recognized by the given finite-state machines.

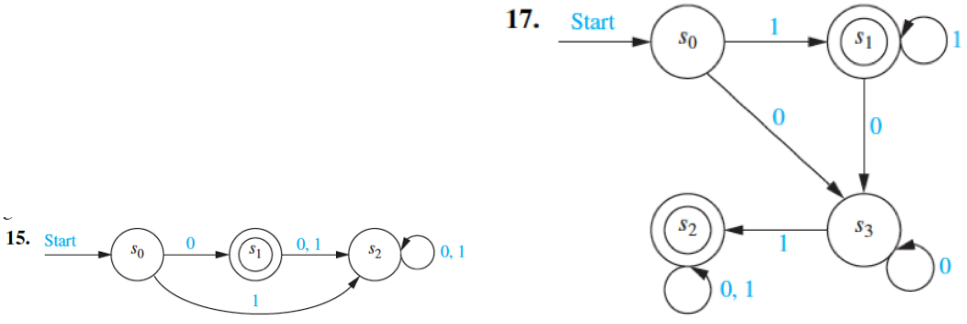


Figure 2.19: Finite State Machines for Exercise 2.6.

Solution 2.1. See Figure 2.20.

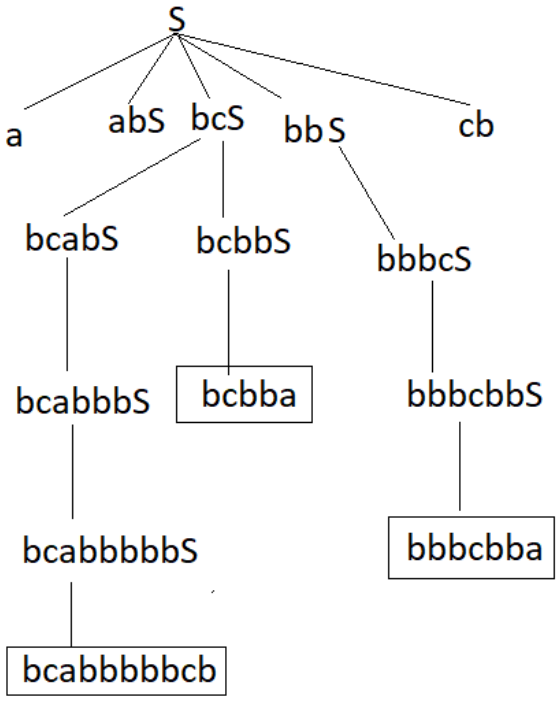


Figure 2.20: Derivation tree for Exercise 2.1.

Solution 2.2. a) $S \to 00S, S \to \lambda$

b) $S \to 10A, A \to 00A, A \to \lambda$

c) $S \to AAS, S \to BBS, AB \to BA, BA \to AB, S \to \lambda, A \to 0, B \to 1$

d) $S \to 0000000000A, A \to 0A, A \to \lambda$

e) $S \to AS, S \to ABS, S \to A, AB \to BA, BA \to AB, A \to 0, B \to 1$

f) $S \to ABS, S \to \lambda, AB \to BA, BA \to AB, A \to 0, B \to 1$

g) $S \to ABS, S \to T, S \to U, T \to AT, T \to A, U \to BU, U \to B, AB \to BA, BA \to AB, A \to 0, B \to 1$

Solution 2.3. See Figure 2.21.

Solution 2.4. a) Yes

b) Yes

c) No

d) No

e) No

f) No

Solution 2.5. See Figure 2.5 and Table 2.7.

Solution 2.6. 15. $S \to 0A, S \to 1B, S \to 0, A \to 0B, A \to 1B, B \to 0B, B \to 1B$

[1]Exercise source: `http://web.mit.edu/6.111/www/f2017/handouts/L06.pdf`.

31. a) ⟨*identifier*⟩ ::= ⟨*lcletter*⟩ | ⟨*identifier*⟩⟨*lcletter*⟩
⟨*lcletter*⟩ ::= *a* | *b* | *c* | ··· | *z*

b) ⟨*identifier*⟩ ::= ⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩ | ⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩ |
⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩ |
⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩
⟨*lcletter*⟩ ::= *a* | *b* | *c* | ··· | *z*

c) ⟨*identifier*⟩ ::= ⟨*ucletter*⟩ | ⟨*ucletter*⟩⟨*letter*⟩ | ⟨*ucletter*⟩⟨*letter*⟩⟨*letter*⟩ |
⟨*ucletter*⟩⟨*letter*⟩⟨*letter*⟩⟨*letter*⟩ | ⟨*ucletter*⟩⟨*letter*⟩⟨*letter*⟩⟨*letter*⟩⟨*letter*⟩ |
⟨*ucletter*⟩⟨*letter*⟩⟨*letter*⟩⟨*letter*⟩⟨*letter*⟩⟨*letter*⟩
⟨*letter*⟩ ::= ⟨*lcletter*⟩ | ⟨*ucletter*⟩
⟨*lcletter*⟩ ::= *a* | *b* | *c* | ··· | *z*
⟨*ucletter*⟩ ::= *A* | *B* | *C* | ··· | *Z*

d) ⟨*identifier*⟩ ::= ⟨*lcletter*⟩⟨*digitorus*⟩⟨*alphanumeric*⟩⟨*alphanumeric*⟩⟨*alphanumeric*⟩ |
⟨*lcletter*⟩⟨*digitorus*⟩⟨*alphanumeric*⟩⟨*alphanumeric*⟩⟨*alphanumeric*⟩⟨*alphanumeric*⟩
⟨*digitorus*⟩ ::= ⟨*digit*⟩ | _
⟨*alphanumeric*⟩ ::= ⟨*letter*⟩ | ⟨*digit*⟩
⟨*letter*⟩ ::= ⟨*lcletter*⟩ | ⟨*ucletter*⟩
⟨*lcletter*⟩ ::= *a* | *b* | *c* | ··· | *z*
⟨*ucletter*⟩ ::= *A* | *B* | *C* | ··· | *Z*
⟨*digit*⟩ ::= 0 | 1 | 2 | ··· | 9

Figure 2.21: Production rules in Bachus-Naur form for Exercise 2.3.

	f		g	
Current state	Input = 1	Input = 0	Input = 1	Input = 0
RESET	RESET	”0”	0	0
”0”	”01”	”0”	0	0
”01”	RESET	”010”	0	0
”010”	”0101”	”0”	0	0
”0101”	”01011”	”010”	0	0
”01011”	RESET	”0”	1	0

Table 2.7: Transition table for Exercise 2.5.

17. $S \to 0C, S \to 1A, S \to 1, A \to 1A, A \to 0C, A \to 1, B \to 0B, B \to 1B, B \to 0, B \to 1, C \to 0C, C \to 1B, C \to 1$.

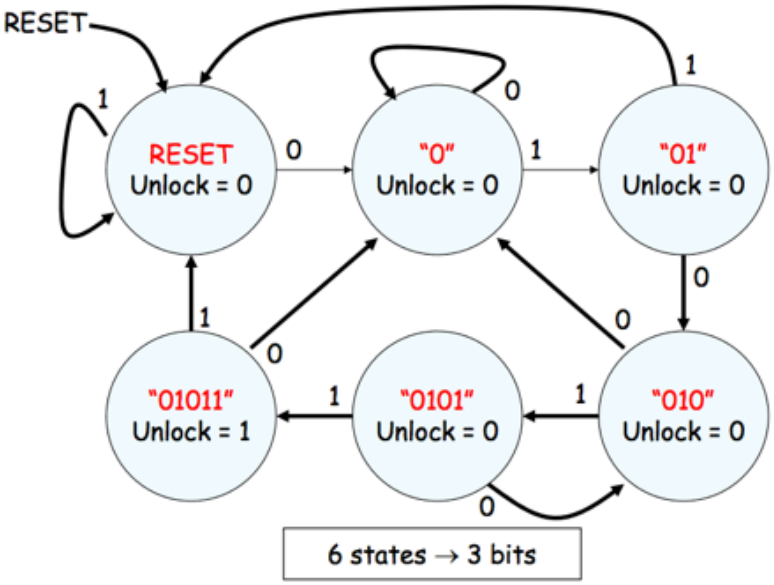


Figure 2.22: Diagram for state machine from Exercise 2.5 [1].

Chapter 3

Introduction to Graph Theory

3.1 Graphs and Terminology

Graphs are discrete structures consisting of vertices and edges that connect these vertices. There are different kinds of graphs, depending on whether edges have directions, whether multiple edges can connect the same pair of vertices, and whether loops are allowed. Problems in almost every conceivable discipline can be solved using graph models. We will give examples to illustrate how graphs are used as models in a variety of areas. For instance, we will show how graphs are used to represent the competition of different species in an ecological niche, how graphs are used to represent who influences whom in an organization, and how graphs are used to represent the outcomes of round-robin tournaments. We will describe how graphs can be used to model acquaintanceships between people, collaboration between researchers, telephone calls between telephone numbers, and links between websites. We will show how graphs can be used to model roadmaps and the assignment of jobs to employees of an organization.

Using graph models, we can determine whether it is possible to walk down all the streets in a city without going down a street twice, and we can find the number of colors needed to color the regions of a map. Graphs can be used to determine whether a circuit can be implemented on a planar circuit board. We can distinguish between two chemical compounds with the same molecular formula but different structures using graphs. We can determine whether two computers are connected by a communications link using graph models of computer networks. Graphs with weights assigned to their edges can be used to solve problems such as finding the shortest path between two cities in a transportation network.We can also use graphs to schedule exams and assign channels to television stations. This chapter will introduce the basic concepts of graph theory and present many different graph models. To solve the wide variety of problems that can be studied using graphs, we will introduce many different graph algorithms. We will also study the complexity of these algorithms.

Definition 19. *A **graph** G = (V, E) consists of V, a nonempty set of vertices (or nodes) and E, a set of edges. Each edge has either one or two vertices associated with it, called its endpoints. An edge is said to connect its endpoints.*

A graph in which each edge connects two different vertices and where no two edges connect the same pair of vertices is called a **simple graph**. Note that in a simple graph, each edge is associated to an unordered pair of vertices, and no other edge is associated to this same pair. Consequently, when there is an edge of a simple graph associated to an unordered pair of vertices noted $\{u, v\}$, we can also say, without possible confusion, that $\{u, v\}$ is an edge of the graph. Graphs that may have multiple edges connecting the same vertices are called multigraphs. When there are m different edges associated to the same unordered pair of vertices $\{u, v\}$, we also say that $\{u, v\}$ is an edge of multiplicity m. That is, we can think of this set of edges as m different copies of an edge $\{u, v\}$.

Graphs may include edges that connect a vertex to itself. Such edges are called **loops**, and sometimes we may even have more than one loop at a vertex. Graphs that may include loops, and possibly multiple

edges connecting the same pair of vertices or a vertex to itself, are sometimes called pseudographs.

So far the graphs we have introduced are **undirected graphs**. Their edges are also said to be undirected. However, to construct a graph model, we may find it necessary to assign directions to the edges of a graph.

Definition 20. *A **directed graph** (or digraph) (V, E) consists of a nonempty set of vertices V and a set of directed edges (or arcs) E. Each directed edge is associated with an ordered pair of vertices. The directed edge associated with the ordered pair (u, v) is said to start at u and end at v.*

When we depict a directed graph with a line drawing, we use an arrow pointing from u to v to indicate the direction of an edge that starts at u and ends at v. A directed graph may contain loops as one or more directed edges that start and end at the same vertices. A directed graph may also contain directed edges that connect vertices u and v in both directions; that is, when a directed graph contains an edge from u to v, it may also contain one or more edges from v to u. Note that we obtain a directed graph when we assign a direction to each edge in an undirected graph. When a directed graph has no loops and has no multiple directed edges, it is called a **simple directed graph**. Because a simple directed graph has at most one edge associated to each ordered pair of vertices (u, v), we call (u, v) an edge if there is an edge associated to it in the graph.

Directed graphs that may have **multiple directed edges** from a vertex to a second (possibly the same) vertex are used to model such networks.We called such graphs **directed multigraphs**. When there are m directed edges, each associated to an ordered pair of vertices (u, v), we say that (u, v) is an edge of **multiplicity** m.

Definition 21. *Two vertices u and v in an undirected graph G are called **adjacent** (or neighbors) in G if u and v are endpoints of one edge e of G. Such an edge e is called incident with the vertices u and v and e is said to connect u and v.*

Definition 22. *The set of all neighbors of a vertex v of G = (V, E), denoted by N(v), is called the **neighborhood** of v. If A is a subset of V , we denote by N(A) the set of all vertices in G that are adjacent to at least one vertex in A. So, $N(A) = \bigcup_{v \in A} N(v)$.*

Definition 23. *The **degree** of a vertex in an undirected graph is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex v is denoted by deg(v).*

Example 3.1.1. What are the degrees and what are the neighborhoods of the vertices in the graphs H and G displayed in Figure 3.1?

Solution:
In H, deg(a) = 4, deg(b) = deg(e) = 6, deg(c) = 1, and deg(d) = 5. The neighborhoods of these vertices are N(a) = {b, d, e}, N(b) = {a, b, c, d, e}, N(c) = {b}, N(d) = {a, b, e} and N(e) = {a, b, d}.
In G, deg(a) = 2, deg(b) = deg(c) = deg(f) = 4, deg(d) = 1, deg(e) = 3, and deg(g) = 0. The neighborhoods of these vertices are N(a) = {b, f}, N(b) = {a, c, e, f}, N(c) = {b, d, e, f}, N(d) = {c}, N(e) = {b, c, f}, N(f) = {a, b, c, e} and N(g) = $\varnothing$.

A vertex of degree zero is called **isolated**. It follows that an isolated vertex is not adjacent to any vertex. The vertex g in graph G in Example 3.1.1 is isolated. A vertex is **pendant** if and only if it has degree one. Consequently, a pendant vertex is adjacent to exactly one other vertex. Vertex d in graph G in Example 1 is pendant.

Definition 24. ***The Handshaking Theorem.*** *Let G = (V, E) be an undirected graph with m edges. Then*

$$2m = \sum_{v \in V} deg(v). \tag{3.1}$$

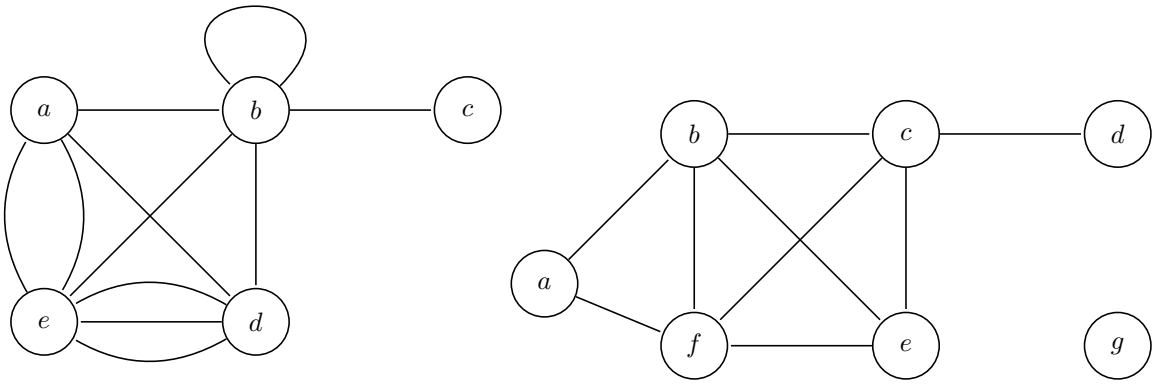


Figure 3.1: The undirected graphs H (left) and G (right).

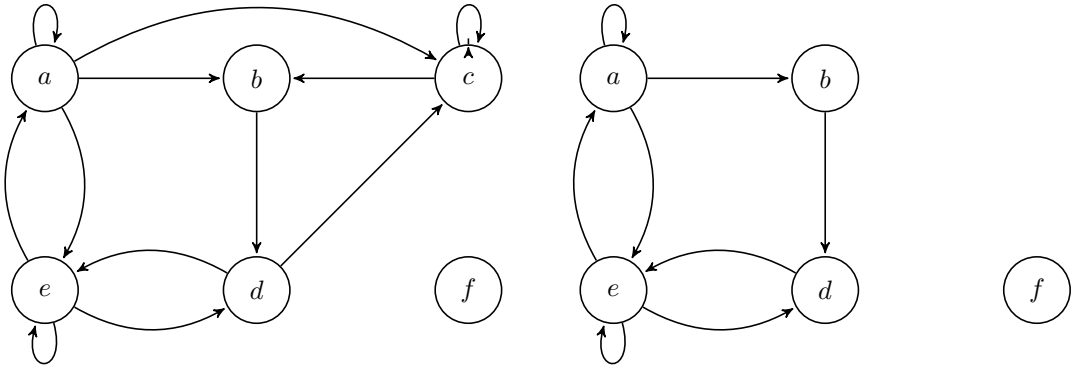


Figure 3.2: The directed graph G (left) and its subgraph G_1 (right).

(Note that this applies even if multiple edges and loops are present.)

Because of this theorem, the following statement is true: An undirected graph has an ***even*** *number of vertices of* ***odd*** *degree.*

Example 3.1.2. How many edges are there in a graph with 10 vertices each of degree 6?

Solution: Because the sum of the degrees of the vertices is $6 * 10 = 60$, it follows that $2m = 60$ where m is the number of edges. Therefore, $m = 30$.

Definition 25. *In a directed graph, the* ***in-degree*** *of a vertex v, denoted by $deg^-(v)$, is the number of edges with v as their terminal vertex. The* ***out-degree*** *of v, denoted by $deg^+(v)$, is the number of edges with v as their initial vertex. Note that a loop at a vertex contributes 1 to* ***both*** *the in-degree and the out-degree of this vertex.*

Example 3.1.3. Find the in-degree and out-degree of each vertex in the graph G with directed edges shown in Figure 3.2.

Solution: The in-degrees in G are $deg^-(a) = 2, deg^-(b) = 2, deg^-(c) = 3, deg^-(d) = 2, deg^-(e) = 3$ and $deg^-(f) = 0$. The out-degrees are $deg^+(a) = 4, deg^+(b) = 1, deg^+(c) = 2, deg^+(d) = 2, deg^+(e) = 3$, and $deg^+(f) = 0$.

Because each edge has an initial vertex and a terminal vertex, the sum of the in-degrees and the sum of the out-degrees of all vertices in a graph with directed edges are the same. Both of these sums are the number of edges in the graph.

Definition 26. *Let G = (V, E) be a directed graph with m edges*

$$\sum_{v \in V} deg^-(v) = \sum_{v \in V} deg^+(v) = |E| = m. \tag{3.2}$$

Sometimes we need only part of a graph to solve a problem. For instance, we may care only about the part of a large computer network that involves the computer centers in New York, Denver, Detroit, and Atlanta. Then we can ignore the other computer centers and all telephone lines not linking two of these specific four computer centers. In the graph model for the large network, we can remove the vertices corresponding to the computer centers other than the four of interest, and we can remove all edges incident with a vertex that was removed. When edges and vertices are removed from a graph, without removing endpoints of any remaining edges, a smaller graph is obtained. Such a graph is called a **subgraph** of the original graph (as can be shown in Figure 3.2).

Definition 27. *A subgraph of a graph $G = (V, E)$ is a graph $H = (W, F)$, where $W \subseteq V$ and $F \subseteq E$. A subgraph H of G is a proper subgraph of G if $H = G$.*

Let G = (V ,E) be a simple graph. The subgraph induced by a subset W of the vertex set V is the graph (W, F), where the edge set F contains an edge in E if and only if both endpoints of this edge are in W.

3.2 Graph Isomorphism

There are many useful ways to represent graphs. Carrying out graph algorithms using the representation of graphs by lists of edges, or by adjacency lists, can be cumbersome if there are many edges in the graph. To simplify computation, graphs can be represented using matrices. Two types of matrices commonly used to represent graphs will be presented here. One is based on the adjacency of vertices, and the other is based on incidence of vertices and edges.

Suppose that $G = (V, E)$ is a simple graph where $|V| = n$. Suppose that the vertices of G are listed arbitrarily as $v_1, v_2, ..., v_n$. The **adjacency matrix** A (or A_G) of G, with respect to this listing of the vertices, is the $n \times n$ zeroone matrix with 1 as its (i, j)th entry when v_i and v_j are adjacent, and 0 as its (i, j)th entry when they are not adjacent. In other words, if its adjacency matrix is $A = [a_{ij}]$, then

$$a_{ij} = \begin{cases} 1 \text{ if } \{v_i, v_j\} \text{ is an edge of G,} \\ 0 \text{ otherwise.} \end{cases}$$

Note that an adjacency matrix of a graph is based on the ordering chosen for the vertices. Hence, there may be as many as $n!$ different adjacency matrices for a graph with n vertices, because there are $n!$ different orderings of n vertices.

The adjacency matrix of a simple graph is symmetric, that is, $a_{ij} = a_{ji}$, because both of these entries are 1 when v_i and v_j are adjacent, and both are 0 otherwise. Furthermore, because a simple graph has no loops, each entry $a_{ii} i, i = 1, 2, 3, ..., n$, is 0.

Adjacency matrices can also be used to represent undirected graphs with loops and with multiple edges. A loop at the vertex v_i is represented by a 1 at the (i, i)th position of the adjacency matrix. When multiple edges connecting the same pair of vertices v_i and v_j, or multiple loops at the same vertex, are present, the adjacency matrix is no longer a zeroone matrix, because the (i, j)th entry of this matrix equals the number of edges that are associated to $\{v_i, v_j\}$. All undirected graphs, including multigraphs and pseudographs, have symmetric adjacency matrices.

Another common way to represent graphs is to use incidence matrices. Let $G = (V, E)$ be an undirected graph. Suppose that $v_1, v_2, ..., v_n$ are the vertices and $e_1, e_2, ..., e_m$ are the edges of G. Then the incidence matrix with respect to this ordering of V and E is the $n \times m$ matrix $M = [m_{ij}]$, where

$$m_{ij} = \begin{cases} 1 \text{ if edge } e_j \text{ is incident with } v_i, \\ 0 \text{ otherwise.} \end{cases}$$

Incidence matrices can also be used to represent multiple edges and loops. Multiple edges are represented in the incidence matrix using columns with identical entries, because these edges are incident with the same pair of vertices. Loops are represented using a column with exactly one entry equal to 1, corresponding to the vertex that is incident with this loop.

We often need to know whether it is possible to draw two graphs in the same way. That is, do the graphs have the same structure when we ignore the identities of their vertices? For instance, in chemistry, graphs are used to model chemical compounds (in a way we will describe later). Different compounds can have the same molecular formula but can differ in structure. Such compounds can be represented by graphs that cannot be drawn in the same way. The graphs representing previously known compounds can be used to determine whether a supposedly new compound has been studied before.

Definition 28. *The simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there exists a oneto- one and onto function f from V_1 to V_2 with the property that a and b are adjacent in G_1 if and only if $f(a)$ and $f(b)$ are adjacent in G_2, for all a and b in V_1. Such a function f is called an isomorphism. Two simple graphs that are not isomorphic are called nonisomorphic.*

In other words, when two simple graphs are isomorphic, there is a one-to-one correspondence between vertices of the two graphs that preserves the adjacency relationship. Isomorphism of simple graphs is an equivalence relation.

It is often difficult to determine whether two simple graphs are isomorphic. There are $n!$ possible one-to-one correspondences between the vertex sets of two simple graphs with n vertices.Testing each such correspondence to see whether it preserves adjacency and nonadjacency is impractical if n is at all large.

Sometimes it is not hard to show that two graphs are not isomorphic. In particular, we can show that two graphs are not isomorphic if we can find a property only one of the two graphs has, but that is preserved by isomorphism. A property preserved by isomorphism of graphs is called a **graph invariant**. For instance, isomorphic simple graphs must have the same number of vertices, because there is a one-to-one correspondence between the sets of vertices of the graphs.

Isomorphic simple graphs also must have the same number of edges, because the one-to-one correspondence between vertices establishes a one-to-one correspondence between edges. In addition, the degrees of the vertices in isomorphic simple graphs must be the same. That is, a vertex v of degree d in G must correspond to a vertex $f(v)$ of degree d in H, because a vertex w in G is adjacent to v if and only if $f(v)$ and $f(w)$ are adjacent in H. The number of vertices, the number of edges, and the number of vertices of each degree are all invariants under isomorphism. If any of these quantities differ in two simple graphs, these graphs cannot be isomorphic. However, when these invariants are the same, it does not necessarily mean that the two graphs are isomorphic. There are no useful sets of invariants currently known that can be used to determine whether simple graphs are isomorphic.

To show that a function f from the vertex set of a graph G to the vertex set of a graph H is an isomorphism, we need to show that f preserves the presence and absence of edges. One helpful way to do this is to use adjacency matrices. In particular, to show that f is an isomorphism, we can show that the adjacency matrix of G is the same as the adjacency matrix of H, when rows and columns are labeled to correspond to the images under f of the vertices in G that are the labels of these rows and columns in the adjacency matrix of G. We illustrate this with Example 3.3.

Example 3.2.1. Determine whether the graphs G and H displayed in Figure 3.3 are isomorphic.

Solution: Both G and H have six vertices and seven edges. Both have four vertices of degree two and two vertices of degree three. It is also easy to see that the subgraphs of G and H consisting of all vertices of degree two and the edges connecting them are isomorphic (as the reader should verify). Because G and H agree with respect to these invariants, it is reasonable to try to find an isomorphism f. We now will define a function f and then determine whether it is an isomorphism. Because $deg(u_1) = 2$

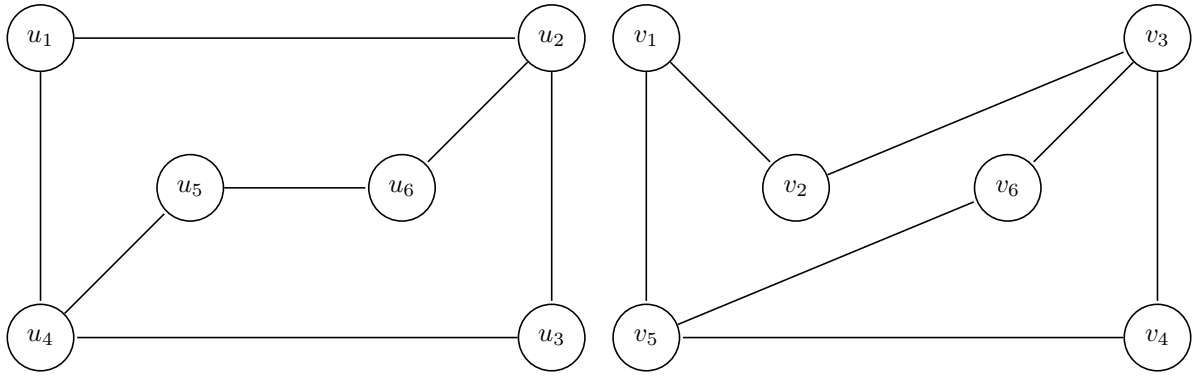


Figure 3.3: Graphs G (left) and H (right).

and because u_1 is not adjacent to any other vertex of degree two, the image of u_1 must be either v_4 or v_6, the only vertices of degree two in H not adjacent to a vertex of degree two. We arbitrarily set $f(u_1) = v_6$. (If we found that this choice did not lead to isomorphism, we would then try $f(u_1) = v_4$.) Because u_2 is adjacent to u_1, the possible images of u_2 are v_3 andv_5. We arbitrarily set $f(u_2) = v_3$. Continuing in this way, using adjacency of vertices and degrees as a guide, we set $f(u_3) = v_4, f(u_4) = v_5, f(u_5) = v_1$, and $f(u_6) = v_2$.We now have a one-to-one correspondence between the vertex set of G and the vertex set of H, namely, $f(u_1) = v_6, f(u_2) = v_3, f(u_3) = v_4, f(u_4) = v_5, f(u_5) = v_1, f(u_6) = v_2$. To see whether f preserves edges, we examine the adjacency matrix of G

$$A_G = \begin{array}{c} \begin{matrix} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{matrix} \tag{3.3}$$

and the adjacency matrix of H with the rows and columns labeled by the images of the corresponding vertices in G,

$$A_H = \begin{array}{c} \begin{matrix} v_6 & v_3 & v_4 & v_5 & v_1 & v_2 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} v_6 \\ v_3 \\ v_4 \\ v_5 \\ v_1 \\ v_2 \end{matrix} \tag{3.4}$$

Because $A_G = A_H$, it follows that f preserves edges. We conclude that f is an isomorphism, so G and H are isomorphic. Note that if f turned out not to be an isomorphism, we would not have established that G and H are not isomorphic, because another correspondence of the vertices in G and H may be an isomorphism.

3.3 Connectivity

Many problems can be modeled with paths formed by traveling along the edges of graphs. For instance, the problem of determining whether a message can be sent between two computers using intermediate links can be studied with a graph model. Problems of efficiently planning routes for mail delivery, garbage

pickup, diagnostics in computer networks and so on, can be solved using models that involve paths in graphs.

Informally, a **path** is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along edges of the graph. As the path travels along its edges, it visits the vertices along this path, that is, the endpoints of these edges.

Definition 29. *Let n be a nonnegative integer and G an undirected graph, in which u and v are two vertices. A **path** of length n from u to v in G is a sequence of n edges $e_1, ..., e_n$ of G for which there exists a sequence $x_0 = u, x_1, ..., x_{n1}, x_n = v$ of vertices such that e_i has, for $i = 1, ..., n$, the endpoints x_{i1} and x_i. When the graph is simple, we denote this path by its vertex sequence $x_0, x_1, ..., x_n$ (because listing these vertices uniquely determines the path). The path is a **circuit** if it begins and ends at the same vertex, that is, if $u = v$, and has length greater than zero. The path or circuit is said to pass through the vertices $x_1, x_2, ..., x_{n1}$ or traverse the edges $e_1, e_2, ..., e_n$. A path or circuit is **simple** if it does not contain the same edge more than once.*

When does a computer network have the property that every pair of computers can share information, if messages can be sent through one or more intermediate computers? When a graph is used to represent this computer network, where vertices represent the computers and edges represent the communication links, this question becomes: When is there always a path between two vertices in the graph?

Definition 30. *An undirected graph is called **connected** if there is a path between every pair of distinct vertices of the graph. An undirected graph that is not **connected** is called **disconnected**. We say that we disconnect a graph when we remove vertices or edges, or both, to produce a disconnected subgraph.*

There is a simple path between every pair of distinct vertices of a connected undirected graph (see proof in [4]).

Thus, any two computers in the network can communicate if and only if the graph of this network is connected.

Suppose that a graph represents a computer network. Knowing that this graph is connected tells us that any two computers on the network can communicate. However, we would also like to understand how reliable this network is. For instance, will it still be possible for all computers to communicate after a router or a communications link fails? To answer this and similar questions, we now develop some new concepts.

Sometimes the removal from a graph of a vertex and all incident edges produces a subgraph with more connected components. Such vertices are called **cut vertices** (or articulation points). The removal of a cut vertex from a connected graph produces a subgraph that is not connected. Analogously, an edge whose removal produces a graph with more connected components than in the original graph is called a **cut edge** or **bridge**. Note that in a graph representing a computer network, a cut vertex and a cut edge represent an essential router and an essential link that cannot fail for all computers to be able to communicate.

VERTEX CONNECTIVITY Not all graphs have cut vertices. For example, the complete graph K_n, where $n \geq 3$, has no cut vertices. When you remove a vertex from K_n and all edges incident to it, the resulting subgraph is the complete graph K_{n-1}, a connected graph. Connected graphs without cut vertices are called **nonseparable** graphs, and can be thought of as more connected than those with a cut vertex. We can extend this notion by defining a more granulated measure of graph connectivity based on the minimum number of vertices that can be removed to disconnect a graph.

We define the **vertex connectivity** of a noncomplete graph G, denoted by $\kappa(G)$, as the minimum number of vertices in a vertex cut.

EDGE CONNECTIVITY We can also measure the connectivity of a connected graph $G = (V, E)$ in terms of the minimum number of edges that we can remove to disconnect it. If a graph has a cut edge, then we need only remove it to disconnect G. If G does not have a cut edge, we look for the smallest set of edges that can be removed to disconnect it. A set of edges E is called an edge cut of G if the subgraph $G - E$ is disconnected. The edge connectivity of a graph G, denoted by $\lambda(G)$, is the minimum number of edges in an edge cut of G.

When $G = (V, E)$ is a non-complete connected graph with at least three vertices, the minimum degree of a vertex of G is an upper bound for both the vertex connectivity of G and the edge connectivity of G. That is, $\kappa(G) \leq min_{v \in V} deg(v)$ and $\lambda(G) \leq min_{v \in V} deg(v)$. To see this, observe that deleting all the neighbors of a fixed vertex of minimum degree disconnects G, and deleting all the edges that have a fixed vertex of minimum degree as an endpoint disconnects G.

It is possible to show that $\kappa(G) \leq \lambda(G)$ when G is a connected noncomplete graph. Note also that $\kappa(K_n) = \lambda(K_n) = min_{v \in V} deg(v) = n - 1$ when n is a positive integer and that $\kappa(G) = \lambda(G) = 0$ when G is a disconnected graph. Putting these facts together, establishes that for all graphs G,

$$\kappa(G) \leq \lambda(G) \leq min_{v \in V}\ deg(v). \tag{3.5}$$

Definition 31. *A **directed graph is strongly connected** if there is a path from a to b and from b to a for all pairs of vertices $\{a, b\}$ in the graph.*

For a directed graph to be strongly connected, there must be a sequence of directed edges from any vertex in the graph to any other vertex. A directed graph can fail to be strongly connected but still be in one piece.

Definition 32. *A **directed graph is weakly connected** if there is a path between every two vertices in the underlying undirected graph.*

That is, a directed graph is weakly connected if and only if there is always a path between two vertices when the directions of the edges are disregarded. Clearly, any strongly connected directed graph is also weakly connected.

There are several ways that paths and circuits can help determine whether two graphs are isomorphic. For example, the existence of a simple circuit of a particular length is a useful invariant that can be used to show that two graphs are not isomorphic. In addition, paths can be used to construct mappings that may be isomorphisms.

3.4 Euler and Hamilton Paths

The town of Königsberg, Prussia (now called Kaliningrad and part of the Russian republic), was divided into four sections by the branches of the Pregel River. These four sections included the two regions on the banks of the Pregel, Kneiphof Island, and the region between the two branches of the Pregel. In the eighteenth century seven bridges connected these regions. The left part of Figure 3.4 depicts these regions and bridges.

The townspeople took long walks through town on Sundays. They wondered whether it was possible to start at some location in the town, travel across all the bridges once without crossing any bridge twice, and return to the starting point.

The Swiss mathematician Leonhard Euler solved this problem. His solution, published in 1736, may be the first usage of graph theory. Euler studied this problem using the multigraph obtained when the four regions are represented by vertices and the bridges by edges. This multigraph is shown in the right

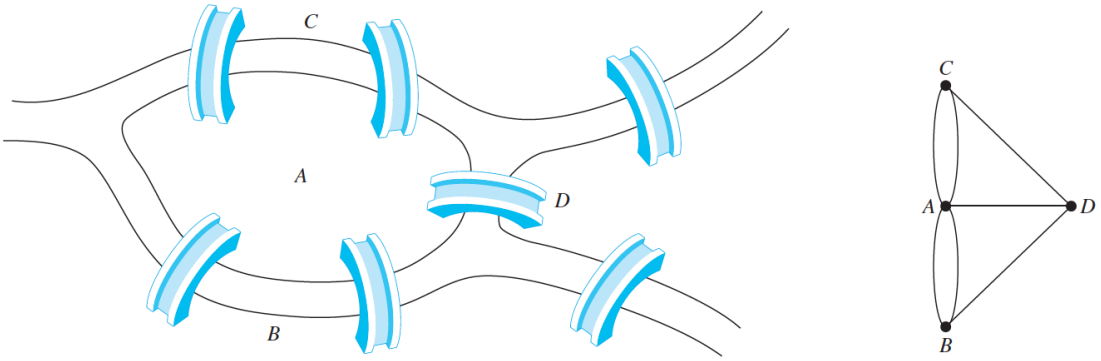


Figure 3.4: The seven bridges of Königsberg (left) and a graph model of the town (right), taken from [4].

part of Figure 3.4. The problem of traveling across every bridge without crossing any bridge more than once can be rephrased in terms of this model. The question becomes: Is there a simple circuit in this multigraph that contains every edge?

Definition 33. *An **Euler circuit** in a graph G is a simple circuit containing every edge of G. An **Euler path** in G is a simple path containing every edge of G.*

Definition 34. ***Necessary and sufficient conditions for the existence of an Euler circuit/path.*** *A connected multigraph with at least two vertices has an Euler circuit if and only if each of its vertices has even degree. A connected multigraph has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.*

Because the multigraph representing these bridges, shown in Figure 3.4, has four vertices of odd degree, it does not have an Euler circuit. There is no way to start at a given point, cross each bridge exactly once, and return to the starting point.

Fleurys algorithm, published in 1883, constructs Euler circuits by first choosing an arbitrary vertex of a connected multigraph, and then forming a circuit by choosing edges successively. Once an edge is chosen, it is removed form the original graph. Edges are chosen successively so that each edge begins where the last edge ends, and so that this edge is not a cut edge unless there is no alternative. We refer the reader to review the preparation videos for step-by-step explanation of this algorithm.

We have developed necessary and sufficient conditions for the existence of paths and circuits that contain every edge of a multigraph exactly once. Can we do the same for simple paths and circuits that contain every vertex of the graph exactly once?

This terminology comes from a game, called the Icosian puzzle, invented in 1857 by the Irish mathematician Sir William Rowan Hamilton. It consisted of a wooden dodecahedron (a polyhedron with 12 regular pentagons as faces, as shown in Figure 3.5 (left)), with a peg at each vertex of the dodecahedron, and string. The 20 vertices of the dodecahedron were labeled with different cities in the world. The object of the puzzle was to start at a city and travel along the edges of the dodecahedron, visiting each of the other 19 cities exactly once, and end back at the first city. The circuit traveled was marked off using the string and pegs.

Definition 35. *A simple path in a graph G that passes through every vertex exactly once is called a **Hamilton path**, and a simple circuit in a graph G that passes through every vertex exactly once is called a **Hamilton circuit**. That is, the simple path $x_0, x_1, ..., x_{n1}, x_n$ in the graph G = (V, E) is a Hamilton path if $V = \{x_0, x_1, ..., x_{n1}, x_n\}$ and $x_i = x_j for 0 i < j n$, and the simple circuit $x_0, x_1, ..., x_{n1}, x_n, x_0$ (with n ¿ 0) is a Hamilton circuit if $x_0, x_1, ..., x_{n1}, x_n$ is a Hamilton path.*

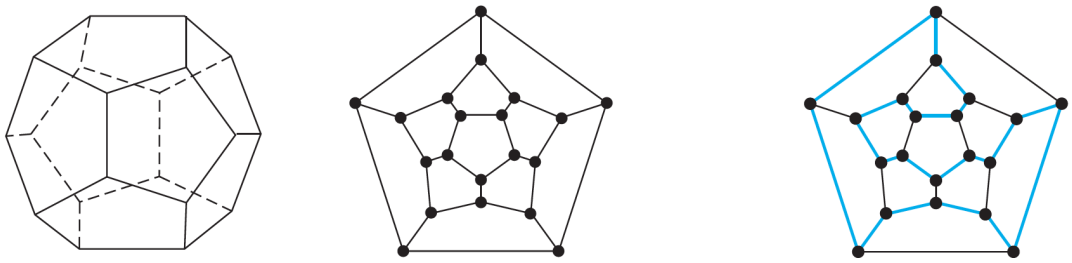


Figure 3.5: Hamilton's puzzle (left and middle) and a solution for the puzzle (right), taken from [4].

Is there a simple way to determine whether a graph has a Hamilton circuit or path? At first, it might seem that there should be an easy way to determine this, because there is a simple way to answer the similar question of whether a graph has an Euler circuit. Surprisingly, there are no known simple necessary and sufficient criteria for the existence of Hamilton circuits. However, many theorems are known that give sufficient conditions for the existence of Hamilton circuits. Also, certain properties can be used to show that a graph has no Hamilton circuit. For instance, a graph with a vertex of degree one cannot have a Hamilton circuit, because in a Hamilton circuit, each vertex is incident with two edges in the circuit. Moreover, if a vertex in the graph has degree two, then both edges that are incident with this vertex must be part of any Hamilton circuit. Also, note that when a Hamilton circuit is being constructed and this circuit has passed through a vertex, then all remaining edges incident with this vertex, other than the two used in the circuit, can be removed from consideration. Furthermore, a Hamilton circuit cannot contain a smaller circuit within it.

Although no useful necessary and sufficient conditions for the existence of Hamilton circuits are known, quite a few sufficient conditions have been found. Note that the more edges a graph has, the more likely it is to have a Hamilton circuit. Furthermore, adding edges (but not vertices) to a graph with a Hamilton circuit produces a graph with the same Hamilton circuit. So as we add edges to a graph, especially when we make sure to add edges to each vertex, we make it increasingly likely that a Hamilton circuit exists in this graph. Consequently, we would expect there to be sufficient conditions for the existence of Hamilton circuits that depend on the degrees of vertices being sufficiently large.We state two of the most important sufficient conditions here.

Definition 36. ***Dirac's Theorem*** *If G is a simple graph with n vertices with $n \geq 3$ such that the degree of every vertex in G is at least $\frac{n}{2}$, then G has a Hamilton circuit.*

Definition 37. ***Ore's Theorem*** *If G is a simple graph with n vertices with $n \geq 3$ such that $deg(u) + deg(v) \geq n$ for every pair of nonadjacent vertices u and v in G, then G has a Hamilton circuit.*

Both Ores theorem and Diracs theorem provide sufficient conditions for a connected simple graph to have a Hamilton circuit. However, these theorems **do not provide necessary conditions** for the existence of a Hamilton circuit. The best algorithms known for finding a Hamilton circuit in a graph or determining that no such circuit exists have exponential worst-case time complexity (in the number of vertices of the graph). Finding an algorithm that solves this problem with polynomial worst-case time complexity would be a major accomplishment because it has been shown that this problem is NP-complete. Consequently, the existence of such an algorithm would imply that many other seemingly intractable problems could be solved using algorithms with polynomial worst-case time complexity.

APPLICATIONS OF HAMILTON CIRCUITS Hamilton paths and circuits can be used to solve practical problems. For example, many applications ask for a path or circuit that visits each road intersection in a city, each place pipelines intersect in a utility grid, or each node in a communications network exactly once. Finding a Hamilton path or circuit in the appropriate graph model can solve such problems. The famous **traveling salesperson problem** or **TSP** (also known in older literature as the traveling salesman problem) asks for the shortest route a traveling salesperson should take to visit a set of cities. This problem reduces to finding a Hamilton circuit

in a complete graph such that the total weight of its edges is as small as possible.

3.5 Supervision session exercises

This Section includes all the exercises that will be focused on during this week's supervision session.

Exercise 3.1. Draw graph models, stating the type of graph used, to represent airline routes where every day there are four flights from Boston to Newark, two flights from Newark to Boston, three flights from Newark to Miami, two flights from Miami to Newark, one flight from Newark to Detroit, two flights from Detroit to Newark, three flights from Newark to Washington, two flights from Washington to Newark, and one flight from Washington to Miami, with

a) an edge between vertices representing cities that have a flight between them (in either direction).

b) an edge between vertices representing cities for each flight that operates between them (in either direction).

c) an edge between vertices representing cities for each flight that operates between them (in either direction), plus a loop for a special sightseeing trip that takes off and lands in Miami.

d) an edge from a vertex representing a city where a flight starts to the vertex representing the city where it ends.

e) an edge for each flight from a vertex representing a city where the flight begins to the vertex representing the city where the flight ends.

Exercise 3.2. Draw adjacency matrixes for the following graphs.

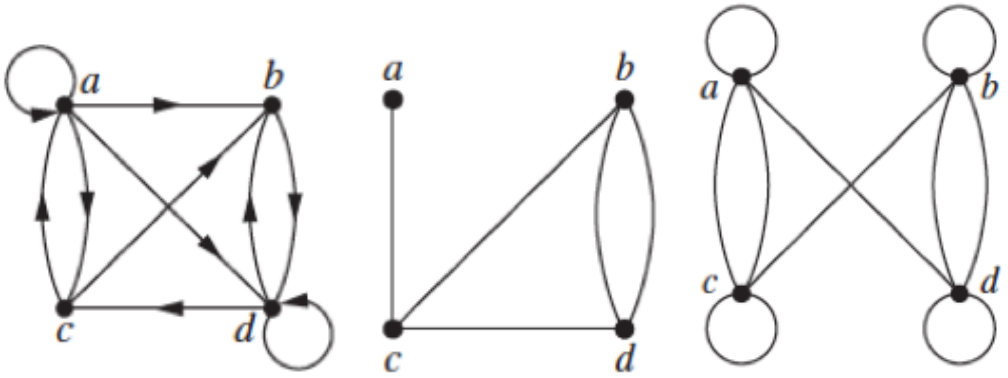


Figure 3.6: Graphs a), b) and c) (from left to right).

Exercise 3.3. Are the following graphs isomorphic?

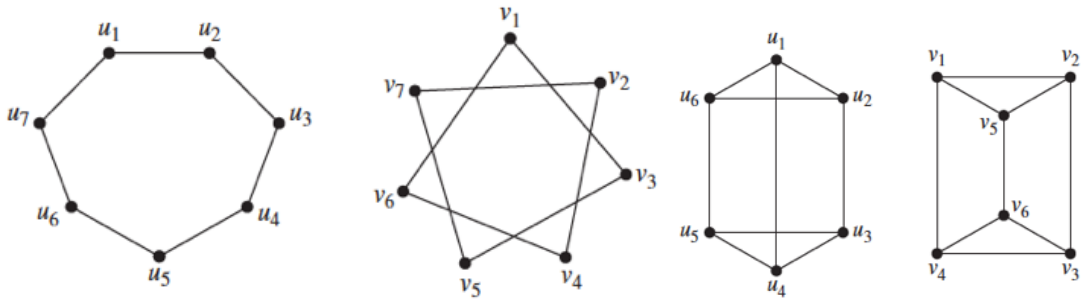


Figure 3.7: Graph pair a) and graph pair b) (from left to right).

Exercise 3.4. Are the simple graphs from the following adjacency matrices isomorphic?

b) $\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$

c) $\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$

Figure 3.8: Adjacency matrixes b) and c) (from left to right).

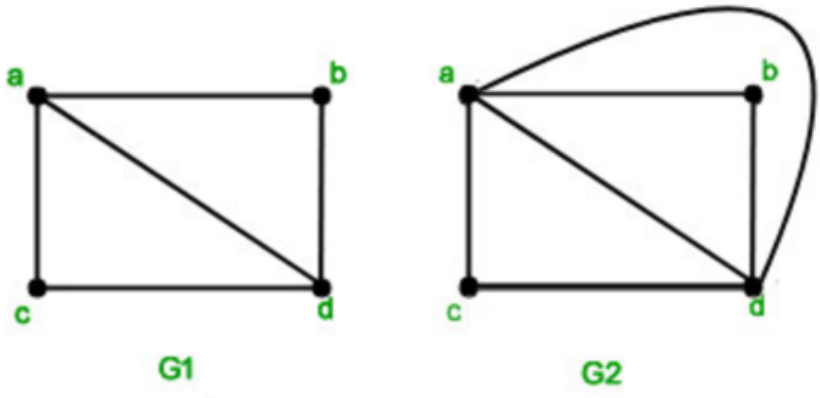


Figure 3.9: Graph from Exercise 3.5[1].

Exercise 3.5. Which graphs shown in Figure 3.9 have an Euler path or Euler circuit?

Exercise 3.6. a) Does the graph in Figure 3.10 have a Hamiltonian circuit?

b) Does the graph in Figure 3.11 have a Hamilton circuit? If so, find it.

[1]Exercise source: `https://www.geeksforgeeks.org/mathematics-euler-hamiltonian-paths/`.
[2]Exercise source: `https://www.geeksforgeeks.org/mathematics-euler-hamiltonian-paths/`.
[3]Exercise source: `https://en.wikipedia.org/wiki/Frucht_graph`.

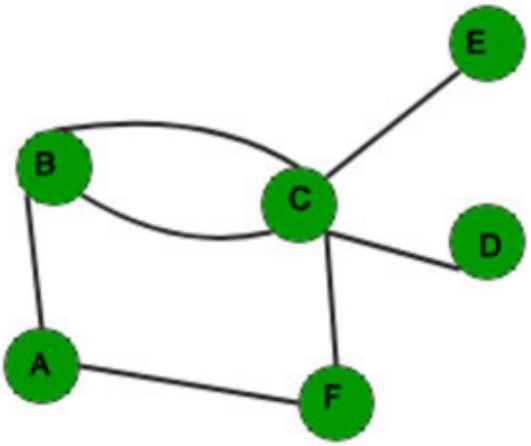


Figure 3.10: Graph from Exercise 3.5[2].

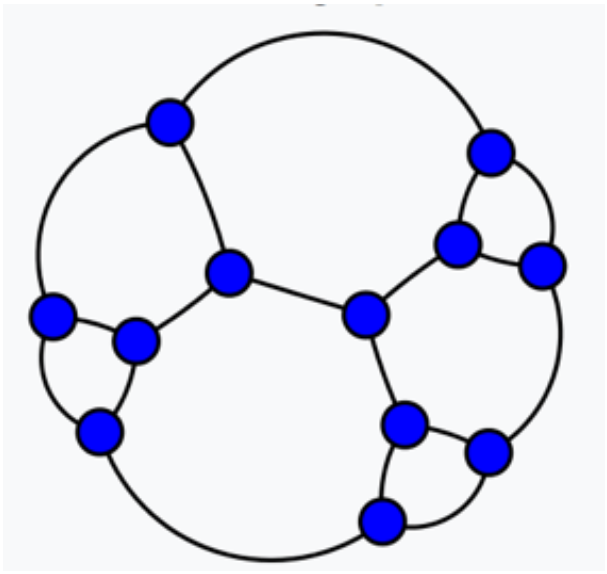


Figure 3.11: Graph from Exercise 3.5[3].

Solution 3.1. See Figure 3.12.

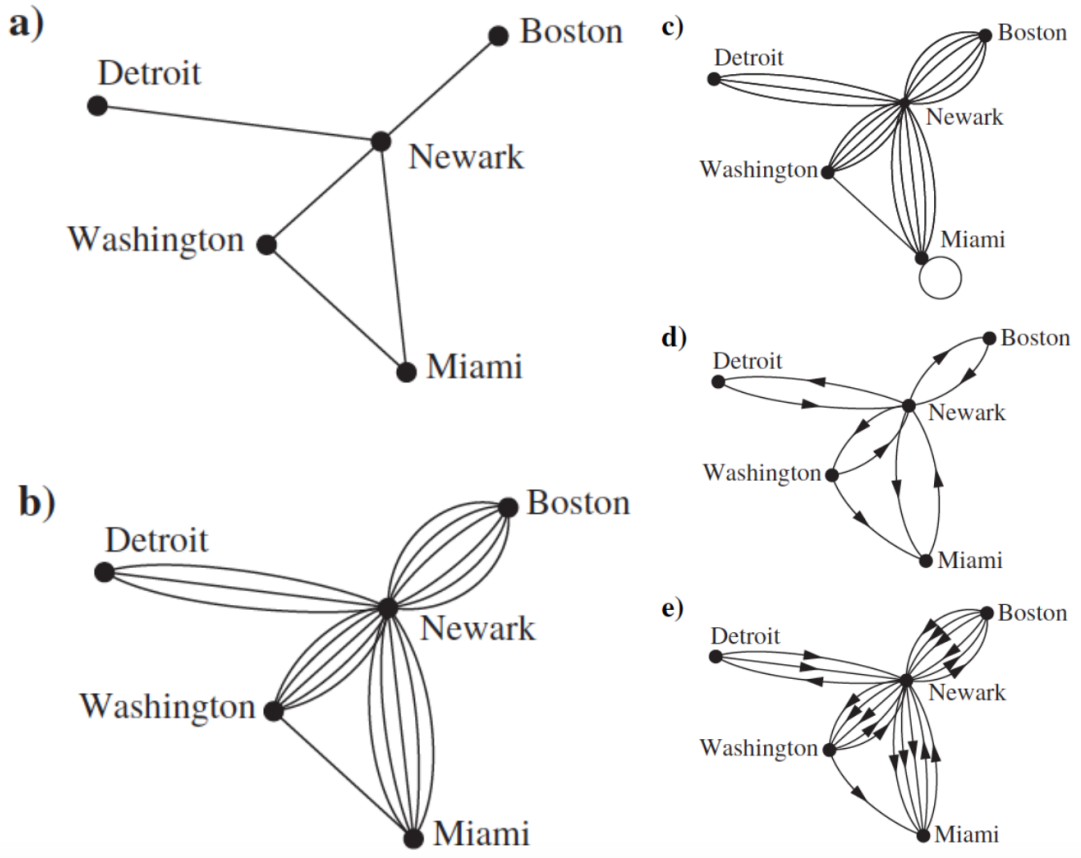


Figure 3.12: Solution for Exercise 3.1.

Solution 3.2. See Figure 3.13.

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 \\ 0 & 2 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 2 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{bmatrix}$$

Figure 3.13: Solution for graphs a), b) and c) (from left to right).

Solution 3.3. a) Yes, b) Yes.

Solution 3.4. b) No, c) No.

Solution 3.5. G_1 has two vertices of odd degree a and d and the rest of them have even degree. So this graph has an Euler path but not an Euler circuit. The path starts and ends at the vertices of odd degree. The path is $a-c-d-a-b-d$. G_2 has four vertices all of even degree, so it has a Euler circuit. The circuit is $a-d-b-a-c-d-a$.

Solution 3.6. a) No, the above graph does not have a Hamiltonian circuit as there are two vertices with degree one in the graph.

b) Yes.

Chapter 4

Complexity and Sorting

4.1 Algorithms

There are many general classes of problems that arise in discrete mathematics. For instance: given a sequence of integers, find the largest one; given a set, list all its subsets; given a set of integers, put them in increasing order; given a network, find the shortest path between two vertices. When presented with such a problem, the first thing to do is to construct a model that translates the problem into a mathematical context. Discrete structures used in such models include sets, sequences, and functions, as well as such other structures as permutations, relations, graphs, trees, networks, and finite state machines.

Setting up the appropriate mathematical model is only part of the solution. To complete the solution, a method is needed that will solve the general problem using the model. Ideally, what is required is a procedure that follows a sequence of steps that leads to the desired answer. Such a sequence of steps is called an algorithm.

Definition 38. *An algorithm is a finite sequence of precise instructions for performing a computation or for solving a problem.*

There are several properties that algorithms generally share. They are useful to keep in mind when algorithms are described. These properties are:

(a) **Input.** An algorithm has input values from a specified set.

(b) **Output.** From each set of input values an algorithm produces output values from a specified set. The output values are the solution to the problem.

(c) **Definiteness.** The steps of an algorithm must be defined precisely.

(d) **Correctness.** An algorithm should produce the correct output values for each set of input values.

(e) **Finiteness.** An algorithm should produce the desired output after a finite (but perhaps large) number of steps for any input in the set.

(f) **Effectiveness.** It must be possible to perform each step of an algorithm exactly and in a finite amount of time.

(g) **Generality.** The procedure should be applicable for all problems of the desired form, not just for a particular set of input values.

GREEDY ALGORITHM Many algorithms we will study are designed to solve optimization problems. The goal of such problems is to find a solution to the given problem that either minimizes or maximizes the value of some parameter. Surprisingly, one of the simplest approaches often leads to a solution of an optimization problem. This approach selects the best choice at each step, instead

of considering all sequences of steps that may lead to an optimal solution. Algorithms that make what seems to be the "best" choice at each step are called **greedy algorithms**. Once we know that a greedy algorithm finds a feasible solution, we need to determine whether it has found an *optimal solution*. (Note that we call the algorithm "greedy" whether or not it finds an optimal solution.) To do this, we either prove that the solution is optimal or we show that there is a counterexample where the algorithm yields a nonoptimal solution.

Greedy algorithms provide an example of an **algorithmic paradigm**, that is, a general approach based on a particular concept that can be used to construct algorithms for solving a variety of problems. When we talk about different types of algorithmic paradigms, we refer also to: brute-force algorithms, divide-and-conquer algorithms (studied in 4.3), probabilistic algorithms[1], backtracking and dynamic programming.

BRUTE-FORCE ALGORITHM Brute force is an important, and basic, algorithmic paradigm. In a brute-force algorithm, a problem is solved in the most straightforward manner based on the statement of the problem and the definitions of terms. Brute-force algorithms are designed to solve problems without regard to the computing resources required. For example, in some brute-force algorithms the solution to a problem is found by examining every possible solution, looking for the best possible. In general, brute-force algorithms are naive approaches for solving problems that do not take advantage of any special structure of the problem or clever ideas. The bubble, insertion, and selection sorts are also considered to be brute-force algorithms; all three of these sorting algorithms are straightforward approaches much less efficient than other sorting algorithms such as the merge sort and the quicksort. Although brute-force algorithms are often inefficient, they are often quite useful. A brute-force algorithm may be able to solve practical instances of problems, particularly when the input is not too large, even if it is impractical to use this algorithm for larger inputs. Furthermore, when designing new algorithms to solve a problem, the goal is often to find a new algorithm that is more efficient than a brute-force algorithm.

Nondeterminism

A **randomized algorithm** is an algorithm which employs a degree of randomness as part of its logic. The algorithm typically uses uniformly random bits as an auxiliary input to guide its behavior, in the hope of achieving good performance in the "average case" over all possible choices of random bits. Formally, the algorithm's performance will be a random variable determined by the random bits; thus either the running time, or the output (or both) are random variables.

A **nondeterministic algorithm** however, is an algorithm that can exhibit different behaviors on different runs, as opposed to a deterministic algorithm. There are several ways an algorithm may behave differently from run to run. A *concurrent algorithm* can perform differently on different runs due to a race condition. A *probabalistic algorithm's* behaviors depends on a random number generator. An algorithm that solves a problem in nondeterministic polynomial time can run in polynomial time or exponential time depending on the choices it makes during execution.

For a good explanation of problems that can be solved in polynomial time and problems that are probably not solvable in polynomial time, check out this MIT lecture: `https://www.youtube.com/watch?v=mr1FMrwi6Ew` .

4.2 Big-O Notation

The time required to solve a problem depends on more than only the number of operations it uses. The time also depends on the hardware and software used to run the program that implements the algorithm. However, when we change the hardware and software used to implement an algorithm, we can closely

[1]Note that, the relationship between probabilistic algorithms and non-deterministic algorithms is not one-to-one.

approximate the time required to solve a problem of size n by multiplying the previous time required by a constant. For example, on a supercomputer we might be able to solve a problem of size n a million times faster than we can on a PC. However, this factor of one million will not depend on n (except perhaps in some minor ways).

One of the advantages of using **big-O notation**, which we introduce in this section, is that we can estimate the growth of a function without worrying about constant multipliers or smaller order terms. This means that, using big-O notation, we do not have to worry about the hardware and software used to implement an algorithm.

Furthermore, using big-O notation, we can assume that the different operations used in an algorithm take the same time, which simplifies the analysis considerably. Big-O notation is used extensively to estimate the number of operations an algorithm uses as its input grows. With the help of this notation, we can determine whether it is practical to use a particular algorithm to solve a problem as the size of the input increases. Furthermore, using big-O notation, we can compare two algorithms to determine which is more efficient as the size of the input grows. For instance, if we have two algorithms for solving a problem, one using $100n^2 + 17n + 4$ operations and the other using n^3 operations, big-O notation can help us see that the first algorithm uses considerably fewer operations when n is large, even though it uses more operations for small values of n, such as $n = 10$. This section introduces big-O notation and the related big-Omega and big-Theta notations. We will explain how big-O, big-Omega, and big-Theta estimates are constructed and establish estimates for some important functions that are used in the analysis of algorithms.

Definition 39. *Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that f(x) is O(g(x)) (see also Figure 4.1a) if there are constants C and k such that*

$$|f(x)| \leq C|g(x)|, \tag{4.1}$$

*whenever $x > k$. [This is read as "f(x) is big-oh of g(x)."] Intuitively, this means that f(x) grows slower than some fixed multiple of g(x) as x grows without bound. The constants C and k in the definition of big-O notation are called **witnesses** to the relationship f(x) is O(g(x)). To establish that f(x) is O(g(x)) we need only one pair of witnesses to this relationship. That is, to show that f(x) is O(g(x)), we need to find only one pair of constants C and k, the witnesses, such that $|f(x)| \leq C|g(x)$ whenever $x > k$. Note that when there is one pair of witnesses to the relationship f(x) is O(g(x)), there are infinitely many pairs of witnesses.*

A useful approach for finding a pair of witnesses is to first select a value of k for which the size of $|f(x)|$ can be readily estimated when $x > k$ and to see whether we can use this estimate to find a value of C for which $|f(x)| \leq C|g(x)|$ for $x > k$. This approach is illustrated in Example 4.2.1.

Example 4.2.1. Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.

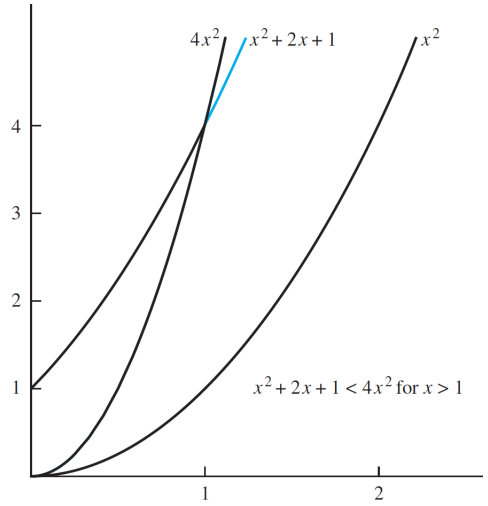
Solution: We observe that we can readily estimate the size of $f(x)$ when $x > 1$ because $x < x^2$ and $1 < x^2$ when $x > 1$. It follows that

$$0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2, \tag{4.2}$$

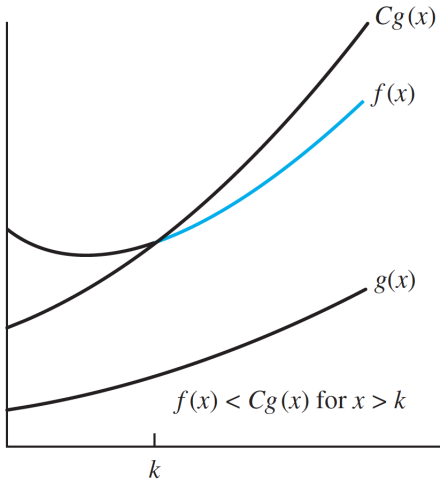
whenever $x > 1$, as shown in Figure 4.1a. Consequently, we can take $C = 4$ and $k = 1$ as witnesses to show that $f(x)$ is $O(x^2)$. That is, $f(x) = x^2 + 2x + 1 < 4x^2$ whenever $x > 1$. (Note that it is not necessary to use absolute values here because all functions in these equalities are positive when x is positive.) Alternatively, we can estimate the size of $f(x)$ when $x > 2$. When $x > 2$, we have $2x \leq x^2$ and $1 \leq x^2$. Consequently, if $x > 2$, we have

$$0 \leq x^2 + 2x + 1 \leq x^2 + x^2 + x^2 = 3x^2. \tag{4.3}$$

It follows that $C = 3$ and $k = 2$ are also witnesses to the relation $f(x)$ is $O(x^2)$. Observe that in the relationship "$f(x)$ is $O(x^2)$," x^2 can be replaced by any function with larger values than x^2. For example, $f(x)$ is $O(x^3)$, $f(x)$ is $O(x^2 + x + 7)$, and so on. It is also true that x^2 is $O(x^2 + 2x + 1)$, because $x^2 < x^2 + 2x + 1$ whenever $x > 1$. This means that $C = 1$ and $k = 1$ are witnesses to the relationship x^2 is $O(x^2 + 2x + 1)$.



(a) The function $x^2 + 2x + 1$ is $O(x^2)$, taken from [4]. The part of the graph of $f(x) = x^2+2x+1$ that satisfies $f(x) < 4x^2$ is shown in blue.



(b) The function $f(x)$ is $O(g)$, taken from [4].The part of the graph of $f(x)$ that satisfies $f(x) < Cg(x)$ is shown in blue.

Figure 4.1: Big-O complexity of function $x^2 + 2x + 1$ (left) and the general case (right).

Polynomials can often be used to estimate the growth of functions. Instead of analyzing the growth of polynomials each time they occur, we would like a result that can always be used to estimate the growth of a polynomial. Definition 40 does this. It shows that the **leading term** of a polynomial **dominates its growth** by asserting that a polynomial of degree n or less is $O(x^n)$.

Definition 40. *Let* $f(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$, *where* $a_0, a_1, ..., a_{n-1}$, *are real numbers. Then* $f(x)$ *is* $O(x^n)$.

Big-O notation is used to estimate the number of operations needed to solve a problem using a specified procedure or algorithm. The functions used in these estimates often include the following: $1, \log n, n, n \log n, n^2, 2^n, n!$

Using calculus it can be shown that each function in the list is smaller than the succeeding function, in the sense that the ratio of a function and the succeeding function tends to zero as n grows without bound. Figure 4.2b displays the graphs of these functions, using a scale for the values of the functions that doubles for each successive marking on the graph. That is, the vertical scale in this graph is logarithmic.

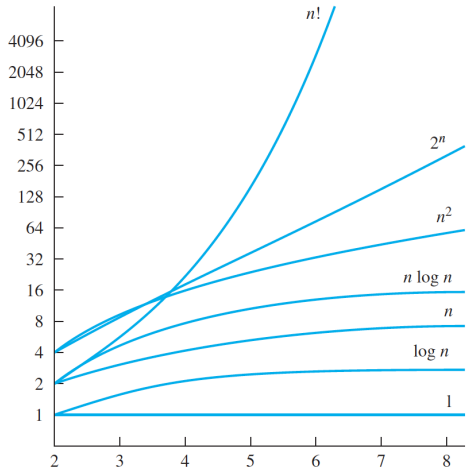
Many algorithms are made up of two or more separate subprocedures. The number of steps used by a computer to solve a problem with input of a specified size using such an algorithm is the sum of the number of steps used by these subprocedures. To give a big-O estimate for the number of steps needed, it is necessary to find big-O estimates for the number of steps used by each subprocedure and then combine these estimates.

Big-O estimates of combinations of functions can be provided if care is taken when different big-O estimates are combined. In particular, it is often necessary to estimate the growth of the sum and the product of two functions. What can be said if big-O estimates for each of two functions are known? To see what sort of estimates hold for the sum and the product of two functions, suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$. From the definition of big-O notation, there are constants C_1, C_2, k_1, and k_2 such that

$$|f_1(x)| \leq C_1|g_1(x)|, \tag{4.4}$$

Complexity	*Terminology*
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	Linearithmic complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

(a) Commonly used terminology for complexity of algorithms, taken from [4].



(b) The growth of commonly used function in Big-O estimates, taken from [4].

Figure 4.2: Different complexities (left) and the corresponding growth of such functions (right).

when $x > k_1$, and

$$|f_2(x)| \leq C_2|g_2(x)|, \tag{4.5}$$

when $x > k_2$. To estimate the sum of $f_1(x)$ and $f_2(x)$, note that

$$|(f_1 + f_2)(x)| = |f_1(x) + f_2(x)| \leq |f_1(x)| + |f_2(x)|, \tag{4.6}$$

using the triangle inequality $|a + b| \leq |a| + |b|$. Furthermore, Definitions 41 and 42 hold. Example 4.2.2 shows how we calculate (simplify) the complexity of functions.

Definition 41. *Suppose that $f_1(x)$ is $O(g_1(x))$ and that $f_2(x)$ is $O(g_2(x))$. Then $(f_1 + f_2)(x)$ is $O(max(|g_1(x)|, |g_2(x)|))$.*

Similarly, suppose that $f_1(x)$ and $f_2(x)$ are both $O(g(x))$. Then $(f_1 + f_2)(x)$ is $O(g(x))$.

Definition 42. *Suppose that $f_1(x)$ and $f_2(x)$ are both $O(g(x))$. Then $(f_1 f_2)(x)$ is $O(g_1(x)g_2(x))$.*

Example 4.2.2. Give a big-O estimate for $f(x) = (x + 1)log(x^2 + 1) + 3x^2$.

Solution: First, a big-O estimate for $(x + 1)\log(x^2 + 1)$ will be found. Note that $(x + 1)$ is $O(x)$. Furthermore, $x^2 + 1 \leq 2x^2$ when $x > 1$. Hence,

$$\log(x^2 + 1) \leq \log(2x^2) = \log 2 + \log x^2 = \log 2 + 2\log x \leq 3\log x, \tag{4.7}$$

if $x > 2$. This shows that $\log(x^2 + 1)$ is $O(\log x)$.

From Definition 42 it follows that $(x+1)\log(x^2+1)$ is $O(x \log x)$. Because $3x^2$ is $O(x^2)$, Definition 41 tells us that $f(x)$ is $O(max(x \log x, x^2))$. Because $x \log x \leq x^2$, for $x > 1$, it follows that $f(x)$ is $O(x^2)$.

Big-O notation is used extensively to describe the growth of functions, but it has limitations. In particular, when $f(x)$ is $O(g(x))$, we have an upper bound, in terms of $g(x)$, for the size of $f(x)$ for large values of x. However, big-O notation does not provide a **lower bound** for the size of $f(x)$ for large x. For this, we use **big-Omega** (big-Ω) notation. When we want to give **both an upper and a lower bound** on the size of a function $f(x)$, relative to a reference function $g(x)$,we use **big-Theta** (big-Θ) notation. Both big-Omega and big-Theta notation were introduced by Donald Knuth in the

1970s. His motivation for introducing these notations was the common misuse of big-O notation when both an upper and a lower bound on the size of a function are needed.

We now define both notations and illustrate its use. After that, we continue with explaining how we use this knowledge when determining the complexity of algorithms.

Definition 43. *Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Omega(g(x))$ if there are positive constants C and k such that*

$$|f(x)| \geq C|g(x)| \tag{4.8}$$

whenever $x > k$. [This is read as "f (x) is big-Omega of g(x)."]

Definition 44. *Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that f(x) is (g(x)) if f(x) is O(g(x)) and f(x) is (g(x)). When f(x) is (g(x)) we say that f is big-Theta of g(x), that f(x) is of order g(x), and that f(x) and g(x) are of the same order.*

There is a strong connection between big-O and big-Omega notation. In particular, $f(x)$ is $(g(x))$ if and only if $g(x)$ is $O(f(x))$.We leave the verification of this fact as a straightforward exercise for the reader.

Example 4.2.3. The function $f(x) = 8x^3 + 5x^2 + 7$ is $(g(x))$, where $g(x)$ is the function $g(x) = x^3$. This is easy to see because $f(x) = 8x^3 + 5x^2 + 7 \geq 8x^3$ for all positive real numbers x. This is equivalent to saying that $g(x) = x^3$ is $O(8x^3 + 5x^2 + 7)$, which can be established directly by turning the inequality around.

When does an algorithm provide a satisfactory solution to a problem? First, it must always produce the correct answer. This can be guaranteed by applying program verification techniques, which we won't study in this course. Second, it should be efficient. How can the efficiency of an algorithm be analyzed? One measure of efficiency is the time used by a computer to solve a problem using the algorithm, when input values are of a specified size. A second measure is the amount of computer memory required to implement the algorithm when input values are of a specified size.

Questions such as these involve the **computational complexity** of the algorithm. An analysis of the time required to solve a problem of a particular size involves the **time complexity** of the algorithm. An analysis of the computer memory required involves the **space complexity** of the algorithm. Considerations of the time and space complexity of an algorithm are essential when algorithms are implemented. It is obviously important to know whether an algorithm will produce an answer in a microsecond, a minute, or a billion years. Likewise, the required memory must be available to solve a problem, so that space complexity must be taken into account.

Considerations of space complexity are tied in with the particular data structures used to implement the algorithm. Because data structures are not dealt with in detail in this course, space complexity will not be considered. We will restrict our attention to time complexity.

The time complexity of an algorithm can be expressed in terms of the number of operations used by the algorithm when the input has a particular size. The operations used to measure time complexity can be the comparison of integers, the addition of integers, the multiplication of integers, the division of integers, or any other basic operation.

Time complexity is described in terms of the number of operations required instead of actual computer time because of the difference in time needed for different computers to perform basic operations. Moreover, it is quite complicated to break all operations down to the basic bit operations that a computer uses. Furthermore, the fastest computers in existence can perform basic bit operations (for instance, adding, multiplying, comparing, or exchanging two bits) in 10^{-11} second (10 picoseconds), but personal computers may require 10^{-8} second (10 nanoseconds), which is 1000 times as long, to do the same operations.

WORST-CASE COMPLEXITY The type of complexity analysis done in Examples 4.2.4, 4.2.5 and 4.2.6 is a worst-case analysis. By the worst-case performance of an algorithm, we mean the largest number of operations needed to solve the given problem using this algorithm on input of

specified size. Worst-case analysis tells us how many operations an algorithm requires to guarantee that it will produce a solution.

We use the following examples to demonstrate how time complexity is determined for some algorithms.

Example 4.2.4. The simplest approach, described with Procedure 4.3c to finding a maximum element in a sequence of elements performs the following:

1. Set the temporary maximum equal to the first integer in the sequence. (The temporary maximum will be the largest integer examined at any stage of the procedure.)

2. Compare the next integer in the sequence to the temporary maximum, and if it is larger than the temporary maximum, set the temporary maximum equal to this integer.

3. Repeat the previous step if there are more integers in the sequence.

4. Stop when there are no integers left in the sequence. The temporary maximum at this point is the largest integer in the sequence.

Describe the time complexity of Procedure 4.3c.

Solution: The number of comparisons will be used as the measure of the time complexity of the algorithm, because comparisons are the basic operations used. To find the maximum element of a set with n elements, listed in an arbitrary order, the temporary maximum is first set equal to the initial term in the list. Then, after a comparison $i \leq n$ has been done to determine that the end of the list has not yet been reached, the temporary maximum and second term are compared, updating the temporary maximum to the value of the second term if it is larger. This procedure is continued, using two additional comparisons for each term of the list-one $i \leq n$, to determine that the end of the list has not been reached and another $max < A[i]$, to determine whether to update the temporary maximum. Because two comparisons are used for each of the second through the nth elements and one more comparison is used to exit the loop when $i = n + 1$, exactly $2(n - 1) + 1 = 2n - 1)$ comparisons are used whenever this algorithm is applied. Hence, the algorithm for finding the maximum of a set of n elements has time complexity $\Theta(n)$, measured in terms of the number of comparisons used. Note that for this algorithm the number of comparisons is independent of particular input of n numbers.

Example 4.2.5. The problem of locating an element in an ordered list occurs in many contexts. The linear search algorithm begins by comparing x (the element we are looking for) and $A[0]$ (the first element in the list). When $x = A[0]$, the solution is the location of $A[0]$, namely, the index 0. When $x \neq A[0]$, compare x with $A[1]$. If $x = A[1]$, the solution is the location, namely, 1. Continue this process, comparing x successively with each term of the list until a match is found, where the solution is the location of that term, unless no match occurs. If the entire list has been searched without locating x, the solution is $NULL$. The pseudocode for the linear search algorithm is displayed as Procedure 4.3b.

Describe the time complexity of Procedure 4.3b.

Solution: At each step of the loop in the algorithm, two comparisons are performed: one $i \leq n$, to see whether the end of the list has been reached and one $x \leq A[1]$, to compare the element x with a term of the list. Finally, one more comparison $i \leq n$ is made outside the loop. Consequently, if $x = A[1]$, $2i + 1$ comparisons are used. The most comparisons, $2n + 2$, are required when the element is not in the list. In this case, $2n$ comparisons are used to determine that x is not $A[i]$, for $i = 1, 2, ..., n$, an additional comparison is used to exit the loop, and one comparison is made outside the loop. So when x is not in the list, a total of $2n + 2$ comparisons are used. Hence, a linear search requires $\Theta(n)$ comparisons in the worst case, because $2n + 2$ is $\Theta(n)$.

Example 4.2.6. Binary search algorithm can be used when the list has terms occurring in order of increasing size (for instance: if the terms are numbers, they are listed from smallest to largest; if they are words, they are listed in lexicographic, or alphabetic, order). It proceeds by comparing the element to be

located to the middle term of the list. The list is then split into two smaller sublists of the same size, or where one of these smaller lists has one fewer term than the other. The search continues by restricting the search to the appropriate sublist based on the comparison of the element to be located and the middle term. This way it makes less comparisons. The pseudocode is presented with Procedure 4.3a.

Describe the time complexity of Procedure 4.3a in terms of the number of comparisons used (and ignoring the time required to compute $m = \lfloor (i + j)/2 \rfloor$ in each iteration of the loop in the algorithm).

Solution: For simplicity, assume there are $n = 2^k$ elements in the list A, where k is a nonnegative integer. Note that $k = \log n$.[2]

At each stage of the algorithm, i and j, the locations of the first term and the last term of the restricted list at that stage, are compared to see whether the restricted list has more than one term. If $i < j$, a comparison is done to determine whether x is greater than the middle term of the restricted list.

At the first stage the search is restricted to a list with 2^{k-1} terms. So far, two comparisons have been used. This procedure is continued, using two comparisons at each stage to restrict the search to a list with half as many terms. In other words, two comparisons are used at the first stage of the algorithm when the list has 2^k elements, two more when the search has been reduced to a list with 2^{k-1} elements, two more when the search has been reduced to a list with 2^{k-2}elements, and so on, until two comparisons are used when the search has been reduced to a list with $2^1 = 2$ elements. Finally, when one term is left in the list, one comparison tells us that there are no additional terms left, and one more comparison is used to determine if this term is x.

Hence, at most $2^{k+2} = 2 \log n + 2$ comparisons are required to perform a binary search when the list being searched has 2^k elements.[3] It follows that in the worst case, binary search requires $O(\log n)$ comparisons. Note that in the worst case, $2 \log n + 2$ comparisons are used by the binary search. Hence, the binary search uses $\Theta(\log n)$ comparisons in the worst case, because $2 \log n + 2 = \Theta(\log n)$. From this analysis it follows that in the worst case, the binary search algorithm is more efficient than the linear search!

AVERAGE-CASE COMPLEXITY Another important type of complexity analysis, besides worst-case analysis, is called average-case analysis. The average number of operations used to solve the problem over all possible inputs of a given size is found in this type of analysis. Average-case time complexity analysis is usually much more complicated than worst-case analysis.

Figure 4.2a displays some common terminology used to describe the time complexity of algorithms. The linear search algorithm has **linear** (worst-case or average-case) **complexity** and the binary search algorithm has **logarithmic** (worst-case) **complexity**. Many important algorithms have $n \log n$, or **linearithmic** (worst-case) **complexity**, such as the merge sort.

An algorithm has **polynomial complexity** if it has complexity $\Theta(n^b)$, where b is an integer with $b \geq 1$. For example, the bubble sort algorithm is a polynomial-time algorithm because it uses $\Theta(n^2)$ comparisons in the worst case. An algorithm has **exponential complexity** if it has time complexity $\Theta(b^n)$, where $b > 1$. The algorithm that determines whether a compound proposition in n variables is satisfiable by checking all possible assignments of truth variables is an algorithm with exponential complexity, because it uses $\Theta(2^n)$ operations. Finally, an algorithm has **factorial complexity** if it has $\Theta(n!)$ time complexity. The algorithm that finds all orders that a traveling salesperson could use to visit n cities has factorial complexity.

[2]If n, the number of elements in the list, is not a power of 2, the list can be considered part of a larger list with 2^{k+1} elements, where $2k < n < 2^{k+1}$. Here 2^{k+1} is the smallest power of 2 larger than n.

[3]If n is not a power of 2, the original list is expanded to a list with 2^{k+1} terms, where $k = \log n$, and the search requires at most $2 \log n + 2$ comparisons.

```
Data: A : list of sorted items
Result: index_x or NULL
i = A[0]
j = A[length(A)-1]
Function binary search(A, x):
    while i < j do
        /* find the middle */
        m = ⌊(i + j)/2⌋
        if x > A[m] then
            /* don't consider the "lower" sublist*/
            i = m + 1
        end
        else
            /* don't consider the "top" sublist */
            j = m
        end
    end
    if x = A[i] then
        return i;
    end
    else
        return NULL
    end
```

(a) Finding the index of an element in a finite sequence.

```
Data: A : list of comparable items
Result: index_x or NULL
index_x = A[0]
Function find index of x(A, x):
    for i = 1; i < lenght(A) − 1, i++ do
        if x == A[i] then
            return i
        end
    end
    return NULL
```

(b) Finding the index of an element in a finite sequence.

```
Data: A : list of comparable items
Result: max : the maximum element of A
max = A[0]
Function find max(A):
    for i = 1; i < lenght(A) − 1, i++ do
        if max < A[i] then
            max = A[i]
        end
    end
    return max
```

(c) Finding the maximum element in a finite sequence.

Figure 4.3: Binary search algorithm (left), linear search (top right) and finding a maximum in a sequence (bottom right).

Data: n : a positive integer
Result: $n!$: factorial of n
Function `factorial(`n`):`

```
/* halting condition - border line case */
if n = 0 then
    return 1
end
else
    return n*factorial(n − 1)
end
```

(a) Recursive algorithm for computing the factorial $n!$.

Data: A, x : a list of comparable elements and one element of same type
Result: $index_x$: index of element x
Function `binary search(`i`,`j`,`x`):`

```
m = ⌊(i + j)/2⌋
/* halting condition - border line case */
if x = A[m] then
    return m
else if x < A[m] ∧ i < m then
    return binary search(i, m − 1, x)
else if x > A[m] ∧ j > m then
    return binary search(m + 1, j, x)
else
    return 0
end
```

(b) Recursive algorithm for binary search.

Figure 4.4: Recursive algorithm for the factorial (left) and the binary search (right).

4.3 Recursive algorithms

Sometimes it is difficult to define an object explicitly. However, it may be easy to define this object in terms of itself. This process is called **recursion**. Functions can be defined recursively, for instance consider the following definition:

$$\sum_{k=0}^{n+1} a_k = \left(\sum_{k=0}^{n} a_k \right) + a_{n+1} \tag{4.9}$$

Sometimes we can reduce the solution to a problem with a particular set of input values to the solution of the same problem with smaller input values.
Definition 45. *An algorithm is called* ***recursive*** *if it solves a problem by reducing it to an instance of the same problem with smaller input.*

Example 4.3.1. Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

Solution: We can build a recursive algorithm that finds $n!$, where n is a nonnegative integer, based on the recursive definition of $n!$, which specifies that $n! = n(n-1)!$ when n is a positive integer, and that $0! = 1$. To find $n!$ for a particular integer, we use the recursive step n times, each time replacing a value of the factorial function with the value of the factorial function at the next smaller integer. At this last step, we insert the value of 0!. The recursive algorithm we obtain is displayed as Procedure 4.4a. To help understand how this algorithm works, we trace the steps used by the algorithm to compute 4!. First, we use the recursive step to write $4! = 4 * 3!$. We then use the recursive step repeatedly to write $3! = 3 * 2!, 2! = 2 * 1!$, and $1! = 1 * 0!$. Inserting the value of $0! = 1$, and working back through the steps, we see that $1! = 1 * 1 = 1, 2! = 2 * 1! = 2, 3! = 3 * 2! = 3 * 2 = 6$, and $4! = 4 * 3! = 4 * 6 = 24$.

Example 4.3.2. Construct a recursive version of a binary search algorithm.

Solution: Suppose we want to locate x in the list A of integers in increasing order. To perform a binary search, we begin by comparing x with the middle term, $A[\lfloor (n+1)/2 \rfloor]$. Our algorithm will terminate if x equals this term and return the location of this term in the sequence. Otherwise, we reduce the search to a smaller search sequence, namely, the first half of the sequence if x is smaller than the middle term of the original sequence, and the second half otherwise. We have reduced the solution of the search problem to the solution of the same problem with a sequence at most half as long. If have we never encountered the search term x, our algorithm returns the value 0. We express this recursive version of a binary search algorithm as Procedure 4.4b.

Many recursive algorithms take a problem with a given input and divide it into one or more smaller problems. This reduction is successively applied until the solutions of the smaller problems can be found quickly. For instance, we perform a binary search by reducing the search for an element in a list to the search for this element in a list half as long. We successively apply this reduction until one element is left. When we sort a list of integers using the merge sort, we split the list into two halves of equal size and sort each half separately. We then merge the two sorted halves. Another example of this type of recursive algorithm is a procedure for multiplying integers that reduces the problem of the multiplication of two integers to three multiplications of pairs of integers with half as many bits. This reduction is successively applied until integers with one bit are obtained. These procedures follow an important algorithmic paradigm known as **divide-and-conquer**, and are called divide-and-conquer algorithms, because they divide a problem into one or more instances of the same problem of smaller size and they conquer the problem by using the solutions of the smaller problems to find a solution of the original problem, perhaps with some additional work.

Suppose that a recursive algorithm divides a problem of size n into a subproblems, where each subproblem is of size $\frac{n}{b}$ (for simplicity, assume that n is a multiple of b; in reality, the smaller problems are often of size equal to the nearest integers either less than or equal to, or greater than or equal to, $\frac{n}{b}$). Also, suppose that a total of $g(n)$ extra operations are required in the conquer step of the algorithm to combine the solutions of the subproblems into a solution of the original problem. Then, if $f(n)$ represents the number of operations required to solve the problem of size n, it follows that f satisfies the recurrence relation:

$$f(n) = af(\frac{n}{b}) + g(n) \tag{4.10}$$

This is called a **divide-and-conquer recurrence relation**.

Example 4.3.3. Binary Search. The binary search algorithm reduces the search for an element in a search sequence of size n to the binary search for this element in a search sequence of size $\frac{n}{2}$, when n is even. (Hence, the problem of size n has been reduced to one problem of size $\frac{n}{2}$.) Two comparisons are needed to implement this reduction (one to determine which half of the list to use and the other to determine whether any terms of the list remain). Hence, if $f(n)$ is the number of comparisons required to search for an element in a search sequence of size n, then $f(n) = f(\frac{n}{2}) + 2$ when n is even.

4.4 Sorting

Ordering the elements of a list is a problem that occurs in many contexts. For example, to produce a telephone directory it is necessary to alphabetize the names of subscribers. Similarly, producing a directory of songs available for downloading requires that their titles be put in alphabetic order. Putting addresses in order in an e-mail mailing list can determine whether there are duplicated addresses. Creating a useful dictionary requires that words be put in alphabetical order. Similarly, generating a parts list requires that we order them according to increasing part number.

Suppose that we have a list of elements of a set. Furthermore, suppose that we have a way to order elements of the set. **Sorting** is putting these elements into a list in which the elements are in increasing order. For instance, sorting the list 7, 2, 1, 4, 5, 9 produces the list 1, 2, 4, 5, 7, 9. Sorting the list d, h, c, a, f (using alphabetical order) produces the list a, c, d, f, h.

An amazingly large percentage of computing resources is devoted to sorting one thing or another. Hence, much effort has been devoted to the development of sorting algorithms. A surprisingly large number of sorting algorithms have been devised using distinct strategies, with new ones introduced regularly. In his fundamental work, The Art of Computer Programming, Donald Knuth devotes close to 400 pages to sorting, covering around 15 different sorting algorithms in depth! More than 100 sorting algorithms have been devised, and it is surprising how often new sorting algorithms are developed. Among the newest sorting algorithms that have caught on is the the library sort, also known as the gapped insertion sort, invented as recently as 2006. There are many reasons why sorting algorithms interest computer scientists and mathematicians. Among these reasons are that some algorithms are

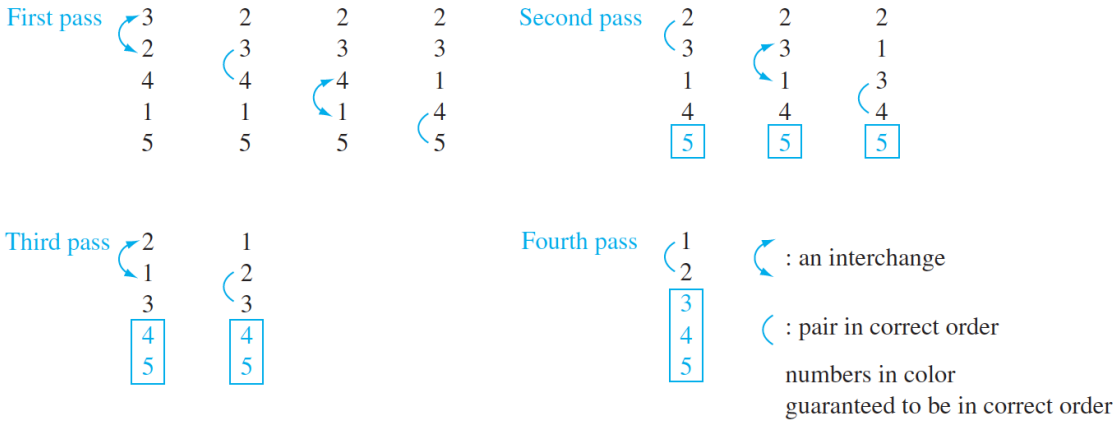


Figure 4.5: Steps of bubble sort, taken from [4].

easier to implement, some algorithms are more efficient (either in general, or when given input with certain characteristics, such as lists slightly out of order), some algorithms take advantage of particular computer architectures, and some algorithms are particularly clever. In this section we will introduce four sorting algorithms, the bubble sort, selection sort, mergesort and quicksort. We cover sorting algorithms both because sorting is an important problem and because these algorithms can serve as examples for many important concepts.

Bubble sort

The **bubble sort** is one of the simplest sorting algorithms, but not one of the most efficient. It puts a list into increasing order by successively comparing adjacent elements, interchanging them if they are in the wrong order. To carry out the bubble sort, we perform the basic operation, that is, interchanging a larger element with a smaller one following it, starting at the beginning of the list, for a full pass.We iterate this procedure until the sort is complete. Pseudocode for the bubble sort is given as Procedure 6. We can imagine the elements in the list placed in a column. In the bubble sort, the smaller elements bubble to the top as they are interchanged with larger elements. The larger elements sink to the bottom. This is illustrated in Example 4.4.1.

Data: A : list of sortable items
Result: A : list of sorted items
n = lenght(A)

```
Function bubblesort():
    while not swapped do
        swapped = false
        for i = 1; 1 ≤ (n − 1); i++ do
            /* if this pair is out of order */
            if A[i-1] > A[i] then
                /* swap them and remember something changed */
                swap( A[i-1], A[i] )
                swapped = true
            end
        end
    end
```

Procedure 1: Bubble sort pseudocode.

Example 4.4.1. Use the bubble sort to put 3, 2, 4, 1, 5 into increasing order.

Solution: The steps of this algorithm are illustrated in Figure 4.5. Begin by comparing the first two elements, 3 and 2. Because $3 > 2$, interchange 3 and 2, producing the list 2, 3, 4, 1, 5. Because $3 <$

4, continue by comparing 4 and 1. Because $4 > 1$, interchange 1 and 4, producing the list 2, 3, 1, 4, 5. Because $4 < 5$, the first pass is complete. The first pass guarantees that the largest element, 5, is in the correct position.

The second pass begins by comparing 2 and 3. Because these are in the correct order, 3 and 1 are compared. Because $3 > 1$, these numbers are interchanged, producing 2, 1, 3, 4, 5. Because $3 < 4$, these numbers are in the correct order. It is not necessary to do any more comparisons for this pass because 5 is already in the correct position. The second pass guarantees that the two largest elements, 4 and 5, are in their correct positions.

The third pass begins by comparing 2 and 1. These are interchanged because $2 > 1$, producing 1, 2, 3, 4, 5. Because $2 < 3$, these two elements are in the correct order. It is not necessary to do any more comparisons for this pass because 4 and 5 are already in the correct positions. The third pass guarantees that the three largest elements, 3, 4, and 5, are in their correct positions. The fourth pass consists of one comparison, namely, the comparison of 1 and 2. Because $1 < 2$, these elements are in the correct order. This completes the bubble sort.

Bubble sort has worst-case and average complexity both (n^2), where n is the number of items being sorted. There exist many sorting algorithms, such as mergesort with substantially better worst-case or average complexity of $O(n \log n)$. Even other (n^2) sorting algorithms, such as insertion sort, tend to have better performance than bubble sort. Therefore, bubble sort is not a practical sorting algorithm when n is large.

The only significant advantage that bubble sort has over most other implementations, even quicksort, but not insertion sort, is that the ability to detect that the list is sorted efficiently is built into the algorithm. When the list is already sorted (best-case), the complexity of bubble sort is only $O(n)$. By contrast, most other algorithms, even those with better average-case complexity, perform their entire sorting process on the set and thus are more complex. However, not only does insertion sort have this mechanism too, but it also performs better on a list that is substantially sorted (having a small number of inversions).

Bubble sort should be avoided in the case of large collections. It will not be efficient in the case of a reverse-ordered collection.

Selection sort

The algorithm divides the input list into two parts: the sublist of items already sorted, which is built up from left to right at the front (left) of the list, and the sublist of items remaining to be sorted that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right, as shown in Procedure 7.

We illustrated one sorting process with Example 4.4.2.

```
Data: A : list of sortable items
Result: A : list of sorted items
n = lenght(A)

Function selectionsort():
    for j = 0; j < n − 1; j + + do
        /* find the min element in the unsorted a[j .. n-1] */
        /* assume the min is the first element */
        iMin = j
        /* test against elements after j to find the smallest */
        for i = j + 1; i < n; i + + do
            /* if this element is less, then it is the new minimum */
            if A[i] < A[iMin] then
                /* found new minimum; remember its index */
                iMin = i
            end
        end
        if iMin ≠ j then
            swap(A[j], A[iMin])
        end
    end
```

Procedure 2: Selection sort pseudocode.

Example 4.4.2. Use the selection sort to put 3, 2, 4, 1, 5 into increasing order.

Solution: We begin by assuming that the minimum is the first element, as suggested by the pseudocode above. Now we scan the rest of the list, which is a sublist of 2, 4, 1, 5 to find a minimum (if there is any number smaller than 3). In this step we find 1 and we switch it with the position of number 3. We now have a list that is divided into two sublists: one sorted [1] and one unsorted [2, 4, 3, 5]. Next, the we attempt to find the next minimum in the unsorted list, but arbitrarily, we already have the number 2 in the right place. The sublists are now: sorted [1, 2] and unsorted [4, 3, 5]. In the next step, we find a minimum of 3, so we swap 3 and 4. Because number 5 is already in the right place, this step concludes sorting the array.

Selection sort is not difficult to analyze compared to other sorting algorithms since none of the loops depend on the data in the array. Selecting the lowest element requires scanning all n elements (this takes $n-1$ comparisons) and then swapping it into the first position. Finding the next lowest element requires scanning the remaining $n-1$ elements and so on, for $(n-1)+(n-2)+...+2+1 = n(n-1)/2 \in \Theta(n^2)$ comparisons. Each of these scans requires one swap for $n-1$ elements (the final element is already in place).

Among simple average-case $\Theta(n^2)$ algorithms, selection sort almost always outperforms bubble sort. Insertion sort is very similar in that after the kth iteration, the first k elements in the array are in sorted order. Insertion sort's advantage is that it only scans as many elements as it needs in order to place the $k+1$st element, while selection sort must scan all remaining elements to find the $k+1$st element.

Simple calculation shows that insertion sort will therefore usually perform about half as many comparisons as selection sort, although it can perform just as many or far fewer depending on the order the array was in prior to sorting. It can be seen as an advantage for some real-time applications that selection sort will perform identically regardless of the order of the array, while insertion sort's running time can vary considerably. However, this is more often an advantage for insertion sort in that it runs much more efficiently if the array is already sorted or "close to sorted."

Finally, selection sort is greatly outperformed on larger arrays by $\Theta(n \log n)$ divide-and-conquer algorithms such as mergesort. However, insertion sort or selection sort are both typically faster for small arrays (i.e. fewer than 10-20 elements). A useful optimization in practice for the recursive algorithms is to switch to insertion sort or selection sort for "small enough" sublists.

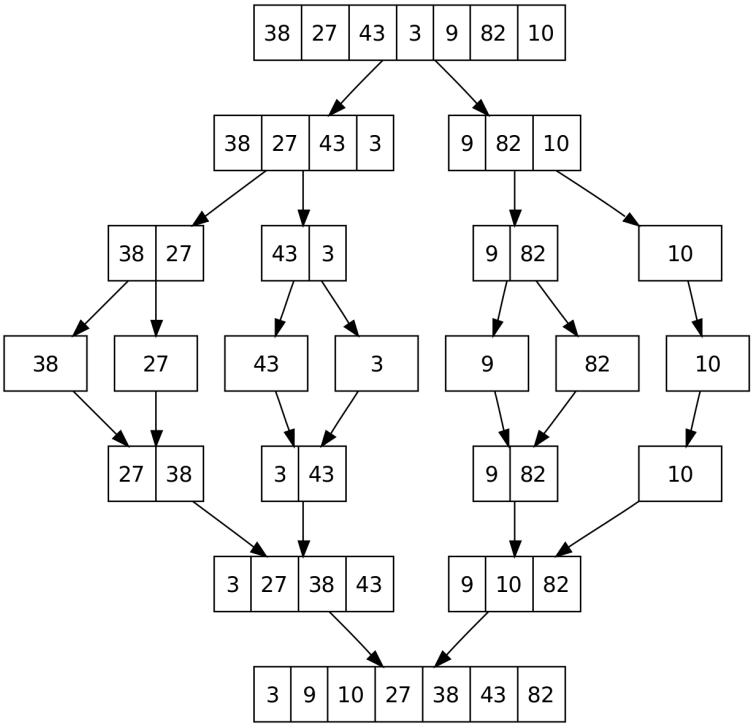


Figure 4.6: Steps of mergesort illustrated on an example.

Mergesort

Conceptually, a mergesort works as follows:

(a) Divide the unsorted list into n sublists, each containing 1 element (a list of 1 element is considered sorted).

(b) Repeatedly merge sublists to produce new sorted sublists until there is only 1 sublist remaining. This will be the sorted list.

To demonstrate how it works, we will try to sort a larger list of elements in Example 4.4.3.
Example 4.4.3. Use the mergesort to put 38, 27, 43, 3, 9, 82, 10 into increasing order.

Solution: We begin by splitting the initial array into two parts. Like the pseudocode suggests, we calculate the index of the middle element, which is our case is $mid = \frac{0+6}{2} = 3$. This means that the first part of the splitted array will be up to the element with index 3. Note that we here we will include the element on that index, even though the algorithm works just as well if we were to exclude it in the first half of the list. We therefore obtain two lists: [38, 27, 43, 3] and [9, 82, 10] (see Figure 4.6). The procedure is now called upon both lists recursively. Therefore the first is further split into lists: [38, 27] and [43, 3] and finally [38] [27] [43] [3]. The second list is split into: [9, 82] and [10] and finally into [9], [82] and [10]. At this step we are "in the bottom" of the recursive calls. As the recursion backtracks (because of the if statement, which is an important stopping condition[4]), the lists are merged with a procedure *merge* which also sorts the elements when merging them. Therefore the program start building up the final solution by merging into lists: [27, 38], [3, 43], [9, 82] and [10]. Next it merges the four lists into 2: [3,27, 38, 43] and [9, 10, 82]. Finally, the two lists are merged into the last one: [3, 9, 10, 27, 38, 43, 82].

[4]Be careful to implement the correct stopping condition in the beginning of the recursive function, otherwise your code will "loop forever". If you are not sure what is going on, try to debug your code by executing line per line and observe what is happening with the program stack.

Data: A : list of sortable items
Result: A : list of sorted items
n = lenght(A)
first = A[0]
last = A[n-1]

Function `mergesort`(*A, first, last*)**:**
  if $last - first \leq 1$ **then**
    return
  end
  mid = $\frac{first+last}{2}$
  /* first recursive call */
  $mergesort(A, first, mid)$
  /* second recursive call */
  $mergesort(A, mid, last)$
  /* merge two lists */
  $merge(A, first, mid, last)$
  return

Procedure 3: Mergesort pseudocode.

In sorting n objects, mergesort has an average and worst-case performance of $O(n \log n)$. If the running time of mergesort for a list of length n is $T(n)$, then the recurrence $T(n) = 2T(\frac{n}{2}) + n$ follows from the definition of the algorithm (apply the algorithm to two lists of half the size of the original list, and add the n steps taken to merge the resulting two lists).

In the worst case, the number of comparisons mergesort makes is equal to or slightly smaller than $(n\lceil \log_2 n \rceil - 2\lceil \log_2 n \rceil + 1)$, which is between $(n \log_2 n - n + 1)$ and $(n \log_2 n + n + O(\log_2 n))$.

For large n and a randomly ordered input list, mergesort's expected (average) number of comparisons approaches $\alpha * n$ fewer than the worst case where $\alpha = -1 + \sum_{k=0}^{\infty} \frac{1}{2^k+1} \approx 0.2645$.

In the worst case, mergesort does about 39% fewer comparisons than quicksort does in the average case. In terms of moves, mergesort's worst case complexity is $O(n \log n)$-the same complexity as quicksort's best case, and mergesort's best case takes about half as many iterations as the worst case.

Mergesort is more efficient than quicksort for some types of lists if the data to be sorted can only be efficiently accessed sequentially, and is thus popular in languages such as Lisp, where sequentially accessed data structures are very common. Unlike some (efficient) implementations of quicksort, mergesort is a stable sort.

Mergesort's most common implementation does not sort in place; therefore, the memory size of the input must be allocated for the sorted output to be stored in.

Quicksort

Quicksort (sometimes called partition-exchange sort) is an efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order. In efficient implementations it is not a stable sort, meaning that the relative order of equal sort items is not preserved. Quicksort can operate in-place on an array, requiring small additional amounts of memory to perform the sorting.

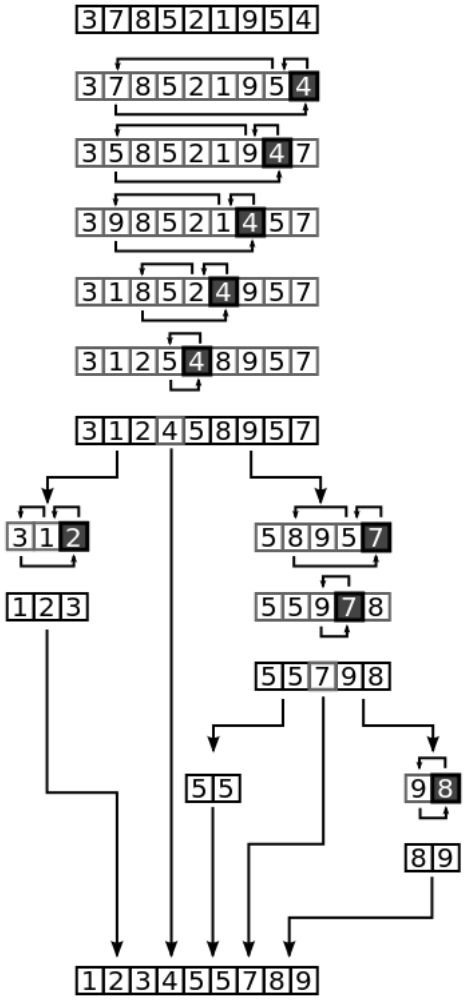
Mathematical analysis of quicksort shows that, on average, the algorithm takes $O(n \log n)$ comparisons to sort n items. In the worst case, it makes $O(n^2)$ comparisons, though this behavior is rare.

Quicksort is a divide-and-conquer algorithm. Quicksort first divides a large array into two smaller sub-arrays: the low elements and the high elements. Quicksort can then recursively sort the sub-arrays. The steps are:

(a) Pick an element, called a **pivot**, from the array.

```
Data: A : list of sortable items
Result: A : list of sorted items
low = 0
// index of the 'low' element
high = n-1
// index of the 'high' element
Function quicksort(A, low, high):
    if low < high then
        p = partition(A, low, high)
        quicksort(A, low, p − 1)
        quicksort(A, p + 1, high)
    end
    return
Function partition(A, low, high):
    pivot = A[high]
    i = low - 1
    for j = low; j ≤ (high − 1), j + + do
        if A[j] < pivot then
            i = i + 1
            swap A[i] with A[j]
        end
    end
    swap A[i+1] with A[high]
    return i + 1
```

(a) Quicksort pseudocode.



(b) Steps of quicksort illustrated on an example.

Figure 4.7: quicksort pseudocode (left) and example of sorting one array (right).

(b) Partitioning: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.

(c) Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

The base case of the recursion is arrays of size zero or one, which never need to be sorted. The pivot selection and partitioning steps can be done in several different ways; the choice of specific implementation schemes greatly affects the algorithm's performance. We demonstrate how the algorithm works with Example 4.4.4 and Procedure 4.7a.

Example 4.4.4. Use the mergesort to put 3, 7, 8, 5, 2, 1, 9, 5, 4 into increasing order.

Solution: The reader can observe the steps of the algorithm in Figure 4.7b. The shaded element is the pivot. It is sometimes chosen as the last element of the partition. However, always choosing the last element in the partition as the pivot in this way results in poor performance ($O(n^2)$) on already sorted arrays, or arrays of identical elements. Since sub-arrays of sorted/identical elements crop up a lot

towards the end of a sorting procedure on a large set, versions of the quicksort algorithm that choose the **pivot as the middle element**[5] run much more quickly than the algorithm described in this diagram on large sets of numbers. The second step of the algorithm is to reorder the elements in the list so that all the elements on the "left" of the pivot and smaller and all the elements on the "right" side of the pivot are larger. Here we can see first-hand why such a choice of pivot is not optimal. The pivot is compared with the first element in the list, number 3. Since it is smaller and on the left side, it can remain there. Next the pivot is compared to the number 7. Because $7 > 4$, the pivot has to be shifted left to provide space for the number 7. There is one element that is therefore pushed to the location where the number 7 was before the shift. That's why number 5 is now the second element of the list. Next couple of steps are similar, until the pivot is in the right place - in our case that is in the fourth element of the list. With this, the partitioning phase is complete. The last step of the algorithm is simply doing the the same thing, but recursively on the partitioned parts of the list. In our case, this means partitioning the list [3, 1, 2] by choosing another pivot, in this case 2, and scrambling the list so that all numbers on the left are lower than two and all numbers on the right are greater than two. The same thing will happen to the other part of the partitioned list, that is the sublist [5, 8, 9, 5, 7]. We are stil following the same strategy to pick a pivot, so number 7 is the "new" pivot. After the partitioning phase, this list is still not ordered, so it is split again (the second "level of depth" in the recursion). After the final switch of 8 and 9, the list is ordered.

Quicksort is a space-optimized version of the binary tree sort. Instead of inserting items sequentially into an explicit tree, quicksort organizes them concurrently into a tree that is implied by the recursive calls. The algorithms make exactly the same comparisons, but in a different order. An often desirable property of a sorting algorithm is stability - that is the order of elements that compare equal is not changed, allowing controlling order of multikey tables (e.g. directory or folder listings) in a natural way. This property is hard to maintain for in situ (or in place) quicksort (that uses only constant additional space for pointers and buffers, and $O(\log n)$ additional space for the management of explicit or implicit recursion). For variant quicksorts involving extra memory due to representations using pointers (e.g. lists or trees) or files (effectively lists), it is trivial to maintain stability. The more complex, or disk-bound, data structures tend to increase time cost, in general making increasing use of virtual memory or disk.

The most direct competitor of quicksort is heapsort. Heapsort's running time is $O(n \log n)$, but heapsort's average running time is usually considered slower than in-place quicksort. This result is debatable; some publications indicate the opposite. Introsort is a variant of quicksort that switches to heapsort when a bad case is detected to avoid quicksort's worst-case running time.

Quicksort also competes with mergesort, another $O(n \log n)$ sorting algorithm. Mergesort is a stable sort, unlike standard in-place quicksort and heapsort, and can be easily adapted to operate on linked lists and very large lists stored on slow-to-access media such as disk storage or network attached storage. Although quicksort can be implemented as a stable sort using linked lists, it will often suffer from **poor pivot choices** without random access. The main disadvantage of mergesort is that, when operating on arrays, efficient implementations require $O(n)$ auxiliary space, whereas the variant of quicksort with in-place partitioning and tail recursion uses only $O(\log n)$ space. (Note that when operating on linked lists, mergesort only requires a small, constant amount of auxiliary storage.)

For the interested reader, there is a video available on YouTube, visualizing the performance of 15 different sorting algorithms here: `https://www.youtube.com/watch?v=kPRA0W1kECg` .

[5] This is also the strategy used for automatic code assessment in the assignment.

4.5 Supervision session exercises

This Section includes all the exercises that will be focused on during this week's supervision session.

Exercise 4.1. Use the bubble sort to sort 3, 1, 5, 7, 4, showing the lists obtained at each step.

Exercise 4.2. Sort these lists using the selection sort.

a) 3, 5, 4, 1, 2

b) 5, 4, 3, 2, 1

c) 1, 2, 3, 4, 5

Exercise 4.3. To establish a big-O relationship, find witnesses C and k such that $|f(x)| \leq C|g(x)|$ whenever $x > k$. Find the least integer n such that $f(x)$ is $O(x^n)$ for each of these functions.

a) $f(x) = 2x^3 + x^2 \log x$

b) $f(x) = 3x^3 + (\log x)^4$

c) $f(x) = (x^4 + x^2 + 1)/(x^3 + 1)$

d) $f(x) = (x^4 + 5\log x)/(x^4 + 1)$

Exercise 4.4. Determine whether each of the functions $2^{(n+1)}$ and $2^{(2n)}$ is $O(2^n)$.

Exercise 4.5. Suppose that you have two different algorithms for solving a problem. To solve a problem of size n, the first algorithm uses exactly $n(\log n)$ operations and the second algorithm uses exactly $n^{(3/2)}$ operations. As n grows, which algorithm uses fewer operations?

Exercise 4.6. Give as good a big-O estimate as possible for each of these functions.

a) $(n^2 + 8)(n + 1)$

b) $(n \log n + n^2)(n^3 + 2)$

c) $(n! + 2^n)(n^3 + \log(n^2 + 1))$

Exercise 4.7. Software packages A and B have processing time exactly $TEP = 3n^{1.5}$ and $TWP = 0.03n^{1.75}$, respectively. If you are interested in faster processing of up to $n = 10^8$ data items, then which package should be chosen?[6] *Hint: try to draw* $3n^{1.5}$ *and* $0.03n^{1.75}$ *on the same coordinate system, maybe using online tools...*

Exercise 4.8. Give an analysis of the running time (Big-Oh notation) for each of the following 4 program fragments (see Figure 4.8). In addition, answer questions i) and ii). Note that the running time corresponds here to the number of times the operation $sum++$ is executed. Also, "sqrt" is the function that returns the square root of a given number.[7]

i) If it takes 10 ms to run program (b) for $n = 100$, how long will it take to run for $n = 400$?
ii) If it takes 10 ms to run program (a) for $n = 100$, how large a problem can be solved in 40 ms?

[6] Exercise source: `https://www.cs.auckland.ac.nz/courses/compsci220s1t/lectures/lecturenotes/GG-lectures/220exercises1.pdf`

[7] Exercise source: `http://www2.cs.uregina.ca/~mouhoubm/=postscript/=c3620/solmidtW06.pdf`

(a)
```
sum = 0;
for(i=0;i<sqrt(n)/2;i++)
 sum++;
for(j=0 ;j<sqrt(n)/4;j++)
 sum++;
for(k=0;k<8+j;k++)
 sum++;
```

(b)
```
sum = 0;
for(i=0;i<sqrt(n)/2;i++)
  for(j=i;j<8+i;j++)
   for(k=j;k<8+j;k++)
    sum++;
```

(c)
```
sum = 0;
for(i=1;i<2*n;i++)
 for(j=1;j<i*i;j++)
   for(k=1;k<j;k++)
     if (j % i == 1)
       sum++;
```

(d)
```
sum = 0;
for(i=1;i<2*n;i++)
 for(j=1;j<i*i;j++)
   for(k=1;k<j;k++)
     if (j % i)
       sum++;
```

Figure 4.8: Code snippets for four short programs.

Solution 4.1. At the end of the first pass: 1, 3, 5, 4, 7; at the end of the second pass: 1, 3, 4, 5, 7; at the end of the third pass: 1, 3, 4, 5, 7; at the end of the fourth pass: 1, 3, 4, 5, 7.

Solution 4.2. a) 1, 5, 4, 3, 2; 1, 2, 4, 3, 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5

b) 1, 4, 3, 2, 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5

c) 1, 2, 3, 4, 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5

Solution 4.3. a) $n = 3,\ C = 3,\ k = 1$

b) $n = 3,\ C = 4,\ k = 1$

c) $n = 1,\ C = 2,\ k = 1$

d) $n = 0,\ C = 2,\ k = 1$

Solution 4.4. $2^{(n+1)}$ is $O(2n)$; $2^{(2n)}$ is not.

Solution 4.5. The algorithm that uses $n \log n$ operations.

Solution 4.6. a) $O(n^3)$

b) $O(n^5)$

c) $O(n^3 * n!)$

Solution 4.7. In the Big-Oh sense, the package A is better. But it outperforms the package B when $T_A(n) \leq T_B(n)$, that is, when $3 * n^{1.5} \leq 0.03 * n^{1.75}$. This inequality reduces to $n^{0.25} \geq 3/0.03 (= 100)$, or $n \geq 10^8$. Thus for processing up to 10^8 data items, the package of choice is B.

Solution 4.8. a) $O(\sqrt{n})$

b) $O(\sqrt{n})$

c) $O(n^4)$

d) $O(n^5)$

i) 20 ms. ii) $n = 1600$.

Chapter 5

Proofs

> "Mathematicians are very happy to discuss things that don't exist, but engineers are very unhappy about it."
>
> – Maurice V. Wilkes, 10 May 1999

5.1 Why Proofs?

Every year students of our program wonder why they need to do proofs. Not many students like it, and that's generally common for all bachelor level engineering courses, where (obviously!) nobody signed up for the program to prove things, but rather to find solutions to practical problems. Before you continue reading this, take 4 minutes and let TedEd answer this question for you (follow link below) [1]. If you are not after 1 million dollars and are still not convinced, then let's give it another try. In 2010, K. Hemmi and C. Löfwall conducted a pilot study presented at Congress of the European Society for Research in Mathematics Education [2]. The authors asked several questions about how important it is to teach basics of proofs to under-graduate engineering students. They gathered and reported data from sending out surveys to mathematicians in Sweden. The answers they got varied, but they mention a few important reasons:

a) Proofs are the soul and the backbone of mathematics - they teach you about being *transparent* in your argumentation. No fuzzy words, speak the truth in one universal language that everyone on Earth can understand!

b) Many mathematicians consider school mathematics as teaching students to apply rules they get through examples from the teacher or a textbook, i.e., following a recipe. This manner does not lead to understanding of what mathematics is, "i.e., concepts and intuitive and logical reasoning about these concepts and their relationships". Proofs *explain* how the concepts are related to each other.

c) Proofs connect all mathematics, without proofs "everything would collapse". You can think of building complex theories as assembling smaller theories together. The smaller theories you assembly with even smaller, simpler theories. But to *verify* your complex theory you need to have a verifiable chain all the way down to your smallest theory.

d) One mathematician stated that proofs enhance students' interest towards mathematics because of many "aha" moments they experience. One reason is therefore our very conscious intention to *challenge you intellectually.*

[1] `https://ed.ted.com/lessons/scott-kennedy-how-to-prove-a-mathematical-theory`

e) One of the mathematicians talked about proof as useful in the learning of mathematical language. This refers to the function of *communication*.

f) Finally, one respondent to the survey believed that it is important to show a few beautiful proofs to the student to demonstrate the *aesthetics* of mathematics.

Of course, some of you might still say "that's nonsense, I will never use proofs again after I pass this course" and you may be right. Maybe you will never write out a proof on a piece of paper again. And, come on, who cares about this "aesthetics" of some mathematical proofs that we hear about - we are engineers, not internal decorators. But, even though furniture and their composition doesn't have to be aesthetic - some of it kind of has to be there (if not for basic living standards of the 20th century).

So maybe you will have to logically reason about your software. You will have to convince people about the correctness and speed of your code. In fact, you may even have to convince your manager that your software is indeed solving the problem in the first place. And even if you will not use mathematical induction (one technique that we look at) to prove it, you will reason logically to communicate the properties of your software. And that's why you need to understand the basics. If you want to know more about the impact of formal methods in general, I suggest to have a look at the book "Industrial-Strength Formal Methods in Practice" by Michael G. Hinchey and Jonathan Peter Bowen.

5.2 Definitions

Before we talk about proofs, we need to mention so called **rules of inference** in propositional logic. These rules of inference are among the most important ingredients in producing valid arguments. We will only touch upon the basic rules of inference in the course script. If you are interested to read more about it, have a look at the book Chapter 1.6 in [4].

The tautology $(p \wedge (p \rightarrow q)) \rightarrow q$ is the basis of the rule of inference called **modus ponens**. This tautology leads to the following *valid argument* (where, as before, the symbol $\therefore$ means "therefore"): Using this notation, the hypotheses are written in a column, followed by a horizontal bar, followed by

$$\begin{array}{ll} & p \\ & p \rightarrow q \\ \hline \therefore & \mathrm{q} \end{array}$$

a line that begins with the therefore symbol and ends with the conclusion. In particular, modus ponens tells us that if a conditional statement and the hypothesis of this conditional statement are both true, then the conclusion must also be true.

Example 5.2.1. Suppose that the conditional statement "*If it snows today, then we will go skiing*" and its hypothesis, "*It is snowing today*," are true. Then, by modus ponens, it follows that the conclusion of the conditional statement, "*We will go skiing*," is true.

There are many useful rules of inference for propositional logic. Perhaps the most widely used of these are listed in Figure 5.1. You don't have to know this table by heart, and we will not focus on rules of inference. We mention them, as they are crucial for learning about proofs.

Formally, a **theorem** is a statement that can be shown to be true. In mathematical writing, the term theorem is usually reserved for a statement that is considered at least somewhat important. Less important theorems sometimes are called **propositions**. Theorems can also be referred to as facts or results. A theorem may be the universal quantification of a conditional statement with one or more premises and a conclusion. However, it may be some other type of logical statement, as the examples later in this chapter will show. We demonstrate that a theorem is true with a **proof**. A proof is a valid argument that establishes the truth of a theorem. The statements used in a proof can include **axioms** (or **postulates**), which are statements we assume to be true (for example, $2 + 2 = 4$), the premises, if any, of the theorem, and previously proven theorems. Axioms may be stated using primitive terms that

[2]Figure taken from [4].

Rule of Inference	*Tautology*	*Name*
p $p \to q$ $\therefore q$	$(p \wedge (p \to q)) \to q$	Modus ponens
$\neg q$ $p \to q$ $\therefore \neg p$	$(\neg q \wedge (p \to q)) \to \neg p$	Modus tollens
$p \to q$ $q \to r$ $\therefore p \to r$	$((p \to q) \wedge (q \to r)) \to (p \to r)$	Hypothetical syllogism
$p \vee q$ $\neg p$ $\therefore q$	$((p \vee q) \wedge \neg p) \to q$	Disjunctive syllogism
p $\therefore p \vee q$	$p \to (p \vee q)$	Addition
$p \wedge q$ $\therefore p$	$(p \wedge q) \to p$	Simplification
p q $\therefore p \wedge q$	$((p) \wedge (q)) \to (p \wedge q)$	Conjunction
$p \vee q$ $\neg p \vee r$ $\therefore q \vee r$	$((p \vee q) \wedge (\neg p \vee r)) \to (q \vee r)$	Resolution

Figure 5.1: Most commonly used rules of inference for propositional logic.[2]

do not require definition, but all other terms used in theorems and their proofs must be defined. Rules of inference, together with definitions of terms, are used to draw conclusions from other assertions, tying together the steps of a proof. In practice, the final step of a proof is usually just the conclusion of the theorem.

5.3 How Do You Prove Things?

Let us start with a silly example[3]. Consider the following conversation between mathematicians Alpha and Beta.

Alpha: I've just discovered a new mathematical truth!

Beta: Oh really? What's that?

Alpha: For every integer x, if x is even, then x^2 is even.

Beta: Hmm...are you sure that this is true?

Alpha: Well, isn't it obvious?

Beta: No, not to me.

Alpha: OK, I'll tell you what. You give me any integer x, and I'll show you that the sentence "if x is even, then x^2 is even" is true. Challenge me.

Beta (eyes narrowing to slits): All right, how about $x = 17$.

Alpha: That's easy. 17 is not even, so the statement "if 17 is even, then 17^2 is even" is vacuously true. Give me a harder one.

Beta: OK, try $x = 62$.

Alpha: Since 62 is even, I guess I have to show you that 62^2 is even.

Beta: That's right.

Alpha (counting on her fingers furiously): According to my calculations, $62^2 = 3844$, and 3844 is clearly even...

Beta: Hold on. It's not so clear to me that 3844 is even. The definition says that 3844 is even if there exists an integer y such that $3844 = 2 * y$. If you want to go around saying that 3844 is even, you have to produce an integer y that works.

Alpha: How about $y = 1922$.

Beta: Yes, you have a point there. So you've shown that the sentence "if x is even, then x^2 is even" is true when $x = 17$ and when $x = 62$. But there are billions of integers that x could be. How do you know you can do this for every one?

Alpha: Let x be any integer.

Beta: Which integer?

Alpha: Any integer at all. It doesn't matter which one. I'm going to show you, using only the fact that x is an integer and nothing else, that if x is even then x^2 is even.

Beta: All right...go on.

Alpha: So suppose x is even.

Beta: But what if it isn't?

Alpha: If x isn't even, then the statement "if x is even, then x^2 is even" is vacuously true. The only time I have anything to worry about is when x is even.

[3]Example source: `https://math.berkeley.edu/~hutching/teach/proofs.pdf`

Beta: OK, so what do you do when x is even?

Alpha: By the definition of "even", we know that there exists at least one integer y such that $x = 2*y$.

Beta: Only one, actually.

Alpha: I think so. Anyway, let y be an integer such that $x = 2 * y$. Squaring both sides of this equation, we get $x^2 = 4 * y^2$. Now to prove that x^2 is even, I have to exhibit an integer, twice which is x^2.

Beta: Doesn't $2 * y^2$ work?

Alpha: Yes, it does. So we're done.

Beta: And since you haven't said anything about what x is, except that it's an integer, you know that this will work for any integer at all.

Alpha: Right.

Beta: OK, I understand now.

Alpha: So here's another mathematical truth. For every integer x, if x is odd, then x^2 is...

This dialogue illustrates several important points. First, a proof is an explanation which convinces other mathematicians that a statement is true. A good proof also helps them understand *why* it is true. The dialogue also illustrates several of the basic techniques for proving that statements are true. Table 5.1 summarizes how to handle proving logical statements with different operators. Don't worry if you do not know exactly what it means for now, you can revisit it after having seen a few examples. Also, note that you will very often need to express what an even and odd integer is in mathematical terms. As you noticed in the dialog above, an integer x **is even** if (and only if) there exists an integer y such that $x = 2 * y$. Similarly, an integer x **is odd** if (and only if) there exists an integer y such that $x = 2 * y + 1$.

Using Table 5.1 we will now go over one small proof. Here we will demonstrate how to prove "for every" statements and "if...then" statements, and how to use "there exists" statements.

Example 5.3.1. Write a proof that for every integer x, if x is odd, then $x + 1$ is even.

Solution: This is a "for every" statement, so the first thing we do is write:

Let x be any integer.

We have to show, using only the fact that x is an integer, that if x is odd then $x+1$ is even. So we write:

Suppose x is odd.

We must somehow use this assumption to deduce that $x + 1$ is even. Recall that the statement "x is odd" means that there exists an integer y such that $x = 2y + 1$. Also, we can give this integer y any name we like; so to avoid confusion, we are going to call it 'w'. So we write:

Let w be an integer such that x = 2w + 1.

Now we want to prove that $x + 1$ is even, i.e., that there exists an integer y such that $x + 1 = 2y$. Here's how we do it:

Adding 1 to both sides of this equation, we get $x + 1 = 2w + 2$. *Let* $y = w + 1$*; then y is an integer and* $x + 1 = 2y$*, so* $x + 1$ *is even.*

We have completed our proof, so we can write:

QED or the symbol on the right

□

which stands for something in Latin which means "that which was to be shown".

The next example will demonstrate how to prove "and" statements.

Statement	Ways to Prove it	Ways to Use it	How to Negate it
p	• Prove that p is true. • Assume p is false, and derive a contradiction.	• p is true. • If p is false, you have a contradiction.	not p
p and q	• Prove p, and then prove q.	• p is true. • q is true.	(not p) or (not q)
p or q	• Assume p is false, and deduce that q is true. • Assume q is false, and deduce that p is true. • Prove that p is true. • Prove that q is true.	• If $p \to r$ and $q \to r$, then r is true. • If p is false, then q is false. • If q is false, then p is true.	(not p) and (not q)
$p \to q$	• Assume p is true, and deduce that q is true. • Assume q is false, and deduce that p is false.	• If p is true, then q is true. • If q is false, then p is false.	p and (not q)
$p \leftrightarrow q$	• Prove $p \to q$ and then prove $q \to p$. • Prove p and q. • Prove (not p) and (not q).	• Statements p and q are interchangeable.	(p and (not q)) or ((not p) and q)
$(\exists x \in S)P(x)$	• Find an x in S for which $P(x)$ is true.	• Say "Let x be an element of S, such that $P(x)$ is true".	$(\forall x \in S)$ not $P(x)$
$(\forall x \in S)P(x)$	• Say "Let x be any element of S". Prove that $P(x)$ is true.	• If $x \in S$, then $P(x)$ is true. • If $P(x)$ is false, then $x \notin S$.	$(\exists x \in S)$ not $P(x)$

Table 5.1: What do do with logical statements when we need to prove them.

Example 5.3.2. Write a proof that for every integer x and for every integer y, if x is odd and y is odd then xy is odd. (Note that the first 'and' in this statement is not a logical 'and'; it is just there to smooth things out when we translate the symbols $(\forall x \in Z)(\forall y \in Z)$ into English.)

Solution: First, following the standard procedure for proving statements that begin with "for every", we write:

Let x and y be any integers.

We need to prove that if x is odd and y is odd then xy is odd. Following the standard procedure for proving "if...then" statements, we write:

Suppose x is odd and y is odd.

This is an "and" statement. We can use it to conclude that x is odd. We can then use the statement that x is odd to give us an integer w such that $x = 2w + 1$. In our proof, we write :

Since x is odd, choose an integer w such that $x = 2w + 1$.

We can also use our "and" statement to conclude that y is odd. We write:

Since y is odd, choose an integer v such that $y = 2v + 1$.

Now we need to show that xy is odd. We can do this as follows:

Then $xy = 4vw + 2v + 2w + 1$. Let $z = 2vw + v + w$; then $xy = 2z + 1$, so xy is odd.

□

The final example here will demonstrate how to prove "if and only if" statements. The proof of a statement of the form $p \leftrightarrow q$ usually looks like this:

$(\rightarrow)$ [proof that $p \rightarrow q$]

$(\leftarrow)$ [proof that $q \rightarrow p$]

Example 5.3.3. Write a proof that for every integer x, x is even if and only if $x + 1$ is odd.

Solution: Let x be any integer. We must show x is even if and only if $x + 1$ is odd.

$(\rightarrow)$ Suppose x is even. Choose an integer y such that $x = 2y$. Then y is also an integer such that $x + 1 = 2y + 1$, so $x + 1$ is odd.

$(\leftarrow)$ Suppose $x + 1$ is odd. Choose an integer y such that $x + 1 = 2y + 1$. Then y is also an integer such that $x = 2y$, so x is even.

□

Now we can conclude that for any integer x, the statements "*x is even*" and "*x + 1 is odd*" are **interchangeable**; this means that we can take any true statement and replace some occurrences of the phrase "**x is even**" with the phrase "**x + 1 is odd**" to get another true statement. For example, mathematicians Alpha and Beta proved in the dialogue that "For every integer x, if x is even then x^2 is even". So the following is also a true statement: For every integer x, if $x + 1$ is odd then x^2 is even.

You may have noticed that the statements above can be proved in a simpler way. In the examples above we have used unusually detailed proofs for didactic purpose. The rest of this chapter will go over 3 basic proof techniques that will help you prove statements faster.

5.4 Basic Proof Techniques

5.4.1 Direct Proof

A direct proof of a conditional statement $p \implies q$ is constructed when the first step is the assumption that p is true; subsequent steps are constructed using rules of inference, with the final step showing that

q must also be true. A direct proof shows that a conditional statement $p \implies q$ is true by showing that if p is true, then q must also be true, so that the combination p true and q false never occurs. In a direct proof, we assume that p is true and use axioms, definitions, and previously proven theorems, together with rules of inference, to show that q must also be true. You will find that direct proofs of many results are quite straightforward, with a fairly obvious sequence of steps leading from the hypothesis to the conclusion. However, direct proofs sometimes require particular insights and can be quite tricky. Let's look at one example.

Example 5.4.1. Give a direct proof that if m and n are both perfect squares, then nm is also a perfect square. (An integer a is a perfect square if there is an integer b such that $a = b^2$.)

Solution: To produce a direct proof of this theorem, we assume that the hypothesis of this conditional statement is true, namely, we assume that m and n are both perfect squares. By the definition of a perfect square, it follows that there are integers s and t such that $m = s^2$ and $n = t^2$. The goal of the proof is to show that mn must also be a perfect square when m and n are; looking ahead we see how we can show this by substituting s^2 for m and t^2 for n into mn. This tells us that $mn = s^2t^2$. Hence, $mn = s^2t^2 = (ss)(tt) = (st)(st) = (st)^2$, using commutativity and associativity of multiplication. By the definition of perfect square, it follows that mn is also a perfect square, because it is the square of st, which is an integer. We have proved that if m and n are both perfect squares, then mn is also a perfect square.

□

5.4.2 Proof By Contradiction

Direct proofs lead from the premises of a theorem to the conclusion. They begin with the premises, continue with a sequence of deductions, and end with the conclusion. Proofs of theorems that do not start with the premises and end with the conclusion, are called indirect proofs. Such are proof by contraposition (not studied here) and contradiction. Notice that in Table 5.1, we mention that one can prove a statement by assuming that it is false and deducing a contradiction. Here is how it works. Suppose that we want to prove that the statement P is true. We begin by assuming that P is false. We then try to deduce a contradiction, i.e. some statement Q which we know is false. If we succeed, then our assumption that P is false must be wrong! So P is true, and our proof is finished. Let's look at a few examples.

Example 5.4.2. Prove that if x is rational and y is irrational, then $x + y$ is irrational. More precisely we should perhaps include quantifiers and say "for all rational numbers x and all irrational numbers y, the sum $x + y$ is irrational". Recall that a real number x is **rational** if there exist integers p and q with $q \neq 0$ such that $x = p/q$. If x is not rational it is called **irrational**.

Solution: Let us assume the negation of what we are trying to prove: namely that there exist a rational number x and an irrational number y such that $x + y$ is rational. We observe that

$y = (x + y) - x$.

Now $x + y$ and x are rational by assumption, and the difference of two rational numbers is rational, since:

$$\frac{p}{q} - \frac{p^{'}}{q^{'}} = \frac{pq^{'} - qp^{'}}{qq^{'}} \tag{5.1}$$

Thus y is rational. But that contradicts our assumptions. So our assumptions cannot be right! So if x is rational and y is irrational then $x + y$ is irrational.

□

Example 5.4.3. Prove that $\sqrt{2}$ is irrational.

Solution: Suppose $\sqrt{2}$ is rational, i.e. $\sqrt{2} = a/b$ for some integers a and b with $b \neq 0$. We can assume that b is positive, since otherwise we can simply change the signs of both a and b. (Then a is positive too, although we will not need this.) Let us choose integers a and b with $\sqrt{2} = a/b$, such that

b is positive and as small as possible. (We can do this by the Well-Ordering Principle, which says that every nonempty set of positive integers has a smallest element.) Squaring both sides of the equation $\sqrt{2} = a/b$ and multiplying both sides by b^2, we obtain $a^2 = 2b^2$. Since a^2 is even, it follows that a is even. Thus $a = 2k$ for some integer k, so $a^2 = 4k^2$, and hence $b^2 = 2k^2$. Since b^2 is even, it follows that b is even. Since a and b are both even, $a/2$ and $b/2$ are integers with $b/2 > 0$, and $\sqrt{2} = (a/2)/(b/2)$, because $(a/2)/(b/2) = a/b$. But we said before that b is as small as possible, so this is a contradiction. Therefore $\sqrt{2}$ cannot be rational.

□

This particular type of proof by contradiction is known as **infinite descent**, which is used to prove various theorems in classical number theory. If there exist positive integers a and b such that $a/b = \sqrt{2}$, then the above proof shows that we can find smaller positive integers a and b with the same property, and repeating this process, we will get an infinite descending sequence of positive integers, which is impossible. Recall that in the above proof, we said We can assume that b is positive, since otherwise we can simply change the signs of both a and b. Another way to write this would be

Without loss of generality, $b > 0$.

"Without loss of generality" means that there are two or more cases (in this proof the cases when $b > 0$ and $b < 0$), but considering just one particular case is enough to prove the theorem, because the proof for the other case or cases works the same way.

Example 5.4.4. Show that at least four of any 22 days must fall on the same day of the week.

Solution: Let p be the proposition "At least four of 22 chosen days fall on the same day of the week." Suppose that $\neg p$ is true. This means that at most three of the 22 days fall on the same day of the week. Because there are seven days of the week, this implies that at most 21 days could have been chosen, as for each of the days of the week, at most three of the chosen days could fall on that day. This contradicts the premise that we have 22 days under consideration. That is, if r is the statement that 22 days are chosen, then we have shown that $\neg p \implies (r \wedge \neg r)$. Consequently, we know that p is true. We have proved that at least four of 22 chosen days fall on the same day of the week.

□

Example 5.4.5. Give a proof by contradiction of the theorem "If $3n + 2$ is odd, then n is odd."

Solution: Let p be "$3n + 2$ is odd" and q be "n is odd". To construct a proof by contradiction, assume that both p and $\neg q$ are true. That is, assume that $3n + 2$ is odd and that n is not odd. Because n is not odd, we know that it is even. Because n is even, there is an integer k such that $n = 2k$. This implies that $3n + 2 = 3(2k) + 2 = 6k + 2 = 2(3k + 1)$. Because $3n + 2$ is $2t$, where $t = 3k + 1, 3n + 2$ is even. Note that the statement "3n + 2 is even" is equivalent to the statement $\neg p$, because an integer is even if and only if it is not odd. Because both p and $\neg p$ are true, we have a contradiction. This completes the proof by contradiction, proving that if $3n + 2$ is odd, then n is odd.

□

The Halting Problem

The finite-state automata studied earlier in this course cannot be used as general models of computation. They are limited in what they can do. For example, finite-state automata are able to recognize regular sets, but are not able to recognize many easy-to-describe sets, including $\{0^n 1^n | n \not\geq 0\}$, which computers recognize using memory. We can use finite-state automata to compute relatively simple functions such as the sum of two numbers, but we cannot use them to compute functions that computers can, such as the product of two numbers. To overcome these deficiencies we can use a more powerful type of machine known as a Turing machine, after **Alan Turing**, the famous mathematician and computer scientist who invented them in the 1930s. Basically, a Turing machine consists of a control unit, which at any step is in one of finitely many different states, together with a tape divided into cells, which is infinite in both directions. Turing machines have read and write capabilities on the tape as the control unit moves back

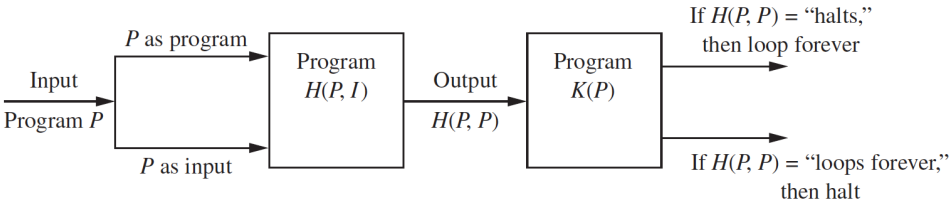


Figure 5.2: Showing that the Halting Problem is unsolvable.[4]

and forth along this tape, changing states depending on the tape symbol read. Turing machines are more powerful than finite-state machines because they include memory capabilities that finite-state machines lack. Turing machines are the most general models of computation; essentially, they can do whatever a computer can do. Note that Turing machines are much more powerful than real computers, which have finite memory capabilities.

We will now describe a proof of one of the most famous theorems in computer science. We will show that there is a problem that cannot be solved using any procedure. That is, we will show there are unsolvable problems. The problem we will study is the **halting problem**. It asks whether there is a procedure that does this: It takes as input a computer program and input to the program and determines whether the program will eventually stop when run with this input. It would be convenient to have such a procedure, if it existed. Certainly being able to test whether a program entered into an infinite loop would be helpful when writing and debugging programs. However, in 1936 Alan Turing showed that no such procedure exists.

Before we present a proof that the halting problem is unsolvable, first note that we cannot simply run a program and observe what it does to determine whether it terminates when run with the given input. If the program halts, we have our answer, but if it is still running after any fixed length of time has elapsed, we do not know whether it will never halt or we just did not wait long enough for it to terminate. We will describe Turing's proof that the halting problem is unsolvable; it is a proof by contradiction.

Assume there is a solution to the halting problem, a procedure called $H(P, I)$. The procedure $H(P, I)$ takes two inputs, one a program P and the other I, an input to the program P. $H(P, I)$ generates the string "halt" as output if H determines that P stops when given I as input. Otherwise, $H(P, I)$ generates the string "loops forever" as output. We will now derive a contradiction.

When a procedure is coded, it is expressed as a string of characters; this string can be interpreted as a sequence of bits. This means that a program itself can be used as data. Therefore a program can be thought of as input to another program, or even itself. Hence, H can take a program P as both of its inputs, which are a program and input to this program. H should be able to determine whether P will halt when it is given a copy of itself as input.

To show that no procedure H exists that solves the halting problem, we construct a simple procedure $K(P)$, which works as follows, making use of the output $H(P, P)$. If the output of $H(P, P)$ is "loops forever", which means that P loops forever when given a copy of itself as input, then $K(P)$ halts. If the output of $H(P, P)$ is "halt", which means that P halts when given a copy of itself as input, then $K(P)$ loops forever. That is, $K(P)$ does the opposite of what the output of $H(P, P)$ specifies, as shown in Figure 5.2.

Now suppose we provide K as input to K. We note that if the output of $H(K, K)$ is "loops forever", then by the definition of K we see that $K(K)$ halts. Otherwise, if the output of $H(K, K)$ is "halt", then by the definition of K we see that $K(K)$ loops forever, in violation of what H tells us. In both cases, we have a contradiction.

Thus, H cannot always give the correct answers. Consequently, there is no procedure that solves the halting problem. Have a look at this video for a more graphical explanation: `https://www.youtube.com/watch?v=macM_MtS_w4`.

[4]Figure taken from [4].

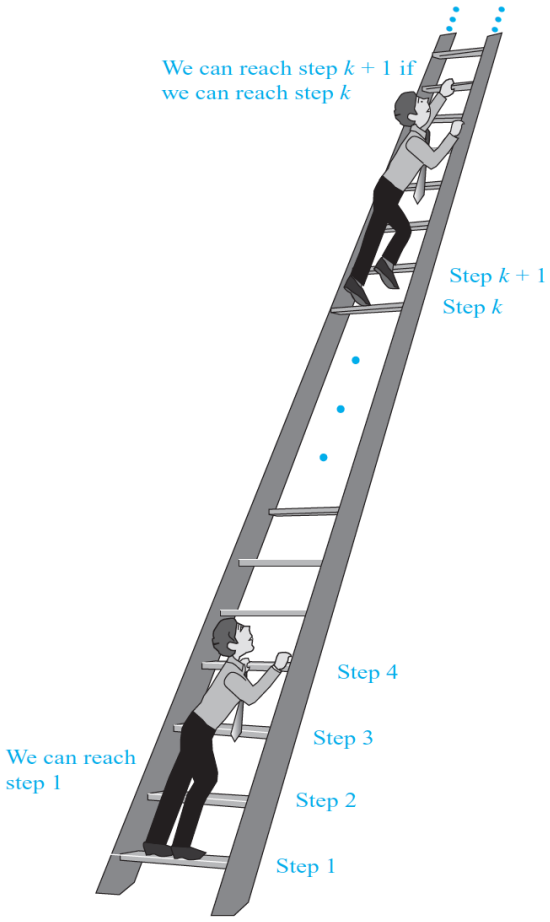


Figure 5.3: Climbing an infinite ladder.

5.4.3 Mathematical Induction

Suppose that we have an infinite ladder, as shown in Figure 5.3, and we want to know whether we can reach every step on this ladder. We know two things:

1. We can reach the first rung of the ladder.

2. If we can reach a particular rung of the ladder, then we can reach the next rung.

Can we conclude that we can reach every rung? By (1), we know that we can reach the first rung of the ladder. Moreover, because we can reach the first rung, by (2), we can also reach the second rung; it is the next rung after the first rung. Applying (2) again, because we can reach the second rung, we can also reach the third rung. Continuing in this way, we can show that we can reach the fourth rung, the fifth rung, and so on. For example, after 100 uses of (2), we know that we can reach the 101st rung. But can we conclude that we are able to reach every rung of this infinite ladder? The answer is yes, with mathematical induction.

In general, mathematical induction can be used to prove statements that assert that $P(n)$ is true for all positive integers n, where $P(n)$ is a propositional function. A proof by mathematical induction has two parts, a basis step, where we show that $P(1)$ is true, and an inductive step, where we show that for all positive integers k, if $P(k)$ is true, then $P(k + 1)$ is true.

Definition 46. *To prove that $P(n)$ is true for all positive integers n, where $P(n)$ is a propositional function, we complete two steps:*

BASIS STEP: We verify that P(1) is true.

INDUCTIVE STEP: We show that the conditional statement $P(k) \implies P(k+1)$ *is true for all positive integers* k.

To complete the inductive step of a proof using the principle of mathematical induction, we assume that $P(k)$ is true for an arbitrary positive integer k and show that under this assumption, $P(k+1)$ must also be true. The assumption that $P(k)$ is true is called the **inductive hypothesis**. Once we complete both steps in a proof by mathematical induction, we have shown that $P(n)$ is true for all positive integers, that is, we have shown that $\forall n P(n)$ is true where the quantification is over the set of positive integers. In the inductive step, we show that $\forall k(P(k) \implies P(k+1))$ is true, where again, the domain is the set of positive integers.

Expressed as a rule of inference, this proof technique can be stated as

$$(P(1) \wedge \forall k(P(k) \implies P(k+1))) \rightarrow \forall n P(n), \tag{5.2}$$

when the domain is the set of positive integers. Let's look at a few examples.

Example 5.4.6. Prove that for every positive integer n,

$$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}. \tag{5.3}$$

Solution:

BASIS STEP: When $n = 1$, we have $1 + ... + n = 1$, and $n(n+1)/2 = 1 \times 2/2 = 1$.

INDUCTIVE STEP: Suppose that for a given $n \in Z^+$,

$$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}. \qquad \text{(inductive hypothesis)} \tag{5.4}$$

Our goal is to show that

$$1 + 2 + 3 + ... + n + (n+1) = \frac{[n+1]([n+1]+1)}{2} \tag{5.5}$$

i.e.,

$$1 + 2 + 3 + ... + n + (n+1) = \frac{(n+1)(n+2)}{2} \tag{5.6}$$

Adding $(n+1)$ to both sides of the inductive hypothesis, we get

$$\begin{aligned} 1 + 2 + ... + n + (n+1) &= \frac{n(n+1)}{2} + (n+1) \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} \\ &= \frac{(n+1)(n+2)}{2} \end{aligned} \tag{5.7}$$

□

We have reached our goal.

Example 5.4.7. Using mathematical induction, prove that $1 + 3 + 5 + ... + (2n - 1) = n^2$.

Solution: Let $P(n)$ denote the proposition that the sum of the first n odd positive integers is n^2. Our conjecture is that $P(n)$ is true for all positive integers. To use mathematical induction to prove this conjecture, we must first complete the basis step; that is, we must show that $P(1)$ is true. Then we must carry out the inductive step; that is, we must show that $P(k + 1)$ is true when $P(k)$ is assumed to be true. We now attempt to complete these two steps.

BASIS STEP: $P(1)$ states that the sum of the first one odd positive integer is 12. This is true because the sum of the first odd positive integer is 1. The basis step is complete.

INDUCTIVE STEP: To complete the inductive step we must show that the proposition $P(k) \implies P(k + 1)$ is true for every positive integer k. To do this, we first assume the inductive hypothesis. The inductive hypothesis is the statement that $P(k)$ is true for an arbitrary positive integer k, that is,

$1 + 3 + 5 + ... + (2k - 1) = k^2$.

(Note that k is a positive integer.) To show that $\forall k(P(k) \implies P(k+1))$ is true, we must show that if $P(k)$ is true (the inductive hypothesis), then $P(k + 1)$ is true. Note that $P(k + 1)$ is the statement that

$1 + 3 + 5 + ... + (2k - 1) + (2k + 1) = (k + 1)^2$.

So, assuming that $P(k)$ is true, it follows that

$$1 + 3 + 5 + ... + (2k - 1) + (2k + 1) = [1 + 3 + ... + (2k - 1)] + (2k + 1) \tag{5.8}$$

$$\stackrel{\text{IH}}{=} k^2 + (2k + 1) \tag{5.9}$$

$$= k^2 + 2k + 1 \tag{5.10}$$

$$= (k + 1)^2 \tag{5.11}$$

□

This shows that $P(k + 1)$ follows from $P(k)$. Note that we used the inductive hypothesis $P(k)$ in the second equality to replace the sum of the first k odd positive integers by k^2. We have now completed both the basis step and the inductive step. That is, we have shown that $P(1)$ is true and the conditional statement $P(k) \implies P(k + 1)$ is true for all positive integers k.

Consequently, by the principle of mathematical induction we can conclude that $P(n)$ is true for all positive integers n. That is, we know that $1 + 3 + 5 + ... + (2n - 1) = n^2$ for all positive integers n.

Example 5.4.8. Using mathematical induction, to show that $1 + 2 + 2^2 + ... + 2^n = 2^{n+1} - 1$ for all non-negative integers n..

Solution: Let $P(n)$ be the proposition that $1 + 2 + 2^2 + ... + 2^n = 2^{n+1} - 1$ for the integer n.

BASIS STEP: $P(0)$ is true because 20=1=21-1. This completes the basis step.

INDUCTIVE STEP: For the inductive hypothesis, we assume that $P(k)$ is true for an arbitrary non-negative integer k. That is, we assume that

$1 + 2 + 2^2 + ... + 2^k = 2^{k+1} - 1$.

To carry out the inductive step using this assumption, we must show that when we assume that $P(k)$ is true, then $P(k + 1)$ is also true. That is, we must show that

$1 + 2 + 2^2 + ... + 2^k + 2^{k+1} = 2^{k+1+1} - 1 = 2^{k+2} - 1$

assuming the inductive hypothesis $P(k)$. Under the assumption of $P(k)$, we see that

$$1 + 2 + 2^2 + ... + 2^k + 2^{k+1} = (1 + 2 + 2^2 + ... + 2^k) + 2^{k+1} \tag{5.12}$$

$$\stackrel{\text{IH}}{=} (2^{k+1} - 1) + 2^{k+1} \tag{5.13}$$

$$= 2 * 2^{k+1} - 1 \tag{5.14}$$

$$= 2^{k+2} - 1 \tag{5.15}$$

□

Note that we used the inductive hypothesis in the second equation in this string of equalities to replace $1 + 2 + 2^2 + ... + 2^k$ by $2^{k+1} - 1$. We have completed the inductive step.

Because we have completed the basis step and the inductive step, by mathematical induction we know that $P(n)$ is true for all non-negative integers n. That is, $1 + 2 + 2^2 + ... + 2^n = 2^{n+1} - 1$ for all non-negative integers n.

Example 5.4.9. Using mathematical induction, prove the inequality: $n < 2^n$, for all positive integers n.

Solution: Let $P(n)$ be the proposition that $n < 2n$.

BASIS STEP: $P(1)$ is true, because $1 < 21 = 2$. This completes the basis step.

INDUCTIVE STEP: We first assume the inductive hypothesis that $P(k)$ is true for an arbitrary positive integer k. That is, the inductive hypothesis $P(k)$ is the statement that $k < 2^k$. To complete the inductive step, we need to show that if $P(k)$ is true, then $P(k + 1)$, which is the statement that $k + 1 < 2^{k+1}$, is true. That is, we need to show that if $k < 2^k$, then $k + 1 < 2^{k+1}$. To show that this conditional statement is true for the positive integer k, we first add 1 to both sides of $k < 2^k$, and then note that $1 \leq 2^k$. This tells us that

$$k + 1 \stackrel{\text{IH}}{<} 2^k + 1 \leq 2^k + 2^k = 2 * 2^k = 2^{k+1} \tag{5.16}$$

□

This shows that $P(k + 1)$ is true, namely, that $k + 1 < 2^{k+1}$, based on the assumption that $P(k)$ is true. The induction step is complete. Therefore, because we have completed both the basis step and the inductive step, by the principle of mathematical induction we have shown that $n < 2^n$ is true for all positive integers n.

5.5 Supervision session exercises

This Section includes all the exercises that will be focused on during this week's supervision session.

Exercise 5.1. Prove that if $m + n$ and $n + p$ are even integers, where m, n, and p are integers, then $m + p$ is even. What kind of proof did you use?

Exercise 5.2. Show that if n is an integer and $n^3 + 5$ is odd, then n is even using proof by contradiction.

Exercise 5.3. Let $P(n)$ be the proposition *If a and b are positive real numbers, then* $(a+b)^n \geq a^n + b^n$. Prove that $P(1)$ is true. What kind of proof did you use?

Exercise 5.4. Prove that at least one of the real numbers $a_1, a_2, ..., a_n$ is greater than or equal to the average of these numbers. What kind of proof did you use?

Exercise 5.5. Prove that 2 divides $n^2 + n$ whenever n is a positive integer.

Exercise 5.6. Prove that if n is a positive integer, then 133 divides $11^{(n+1)} + 12^{(2*n-1)}$.

Solution 5.1. Direct proof: Suppose that $m+n$ and $n+p$ are even. Then $m+n=2s$ for some integer s and $n+p=2t$ for some integer t. If we add these, we get $m+p+2n=2s+2t$. Subtracting $2n$ from both sides and factoring, we have $m+p=2s+2t-2n=2(s+t-n)$. Because we have written $m+p$ as 2 times an integer, we conclude that $m+p$ is even.

□

Solution 5.2. Suppose that n^3+5 is odd and n is odd. Because n is odd and the product of two odd numbers is odd, it follows that n^2 is odd and then that n^3 is odd. But then $5=(n^3+5)-n^3$ would have to be even because it is the difference of two odd numbers. Therefore, the supposition that n^3+5 and n were both odd is wrong.

□

Solution 5.3. $P(1)$ is true because $(a+b)^1=a+b\geq a^1+b^1=a+b$. Direct proof.

□

Solution 5.4. We will give a proof by contradiction. Suppose that $a_1, a_2, ..., a_n$ are all less than A, where A is the average of these numbers. Then $a_1+a_2++a_n<n*A$. Dividing both sides by n shows that $A=(a_1+a_2++a_n)/n<A$, which is a contradiction.

□

Solution 5.5. Basis step: $1+1=2$ is divisible by 2.

Inductive step: Assume the inductive hypothesis, that k^2+k is divisible by 2. Then

$$\begin{aligned}(k+1)^2+(k+1)=\\ k^2+2k+1+k+1=\\ (k^2+k)+2(k+1)\end{aligned} \tag{5.17}$$

The sum of a multiple of 2 (by the inductive hypothesis) and a multiple of 2 (by definition) is divisible by 2.

□

Solution 5.6. Basis step: $11^{(1+1)}+12^{(2*1-1)}=121+12=133$

Inductive step: Assume the inductive hypothesis, that $11^{(n+1)}+12^{(2*n-1)}$ is divisible by 133. Then

$$\begin{aligned}11^{((n+1)+1)}+12^{(2(n+1)-1)}=\\ 11*11^{(n+1)}+144*12^{(2n-1)}=\\ 11*11^{(n+1)}+(11+133)*12^{(2n-1)}=\\ 11(11^{(n+1)}+12^{(2n-1)})+133*12^{(2n-1)}\end{aligned} \tag{5.18}$$

The expression in parentheses (first term of the last line in equation above) is divisible by 133 by the inductive hypothesis, and obviously the second term is divisible by 133, so the entire quantity is divisible by 133.

□

Chapter 6

Introduction to Statistics

See chapters 1-6 and 8-12 from the course literature [5].

Statistics is a branch of mathematics dealing with the collection, organization, analysis, interpretation and presentation of data. Thus, data collection is required to study or learn about something.

6.1 Basics of statistics

In cases in which a study does not provide a set of data, it has to be generated conducting experiments. For example, if a company wants to determine which is the best documentation tool for their interests and they have narrowed their scope to two options. To extract data, the representatives of the company could divide some volunteers into two groups and give a different tool to each group. The extracted data can be analyzed and interpreted afterwards.

It is important to note, however, that in order to be able to draw a valid conclusion from the data, it is essential that the volunteers were divided randomly. If other policies are applied (e.g. the groups are split in terms of age, background or gender) the generated data can be biased.

In the previous example, the representatives of the company strive to obtain information from a total collection of elements—the workers of their company—. which in statistics are refer as *population*. However, the population is often too large for us to examine each of its members and therefore we try to learn about this population by choosing and then examining a subgroup of its elements. This subgroup of a population is called a *sample*. A sample must be representative of a population and, as explained before, the best way of accomplishing that is by choosing its members in a totally random fashion. Following the previous example, while the whole population of the experiment are the total amount of workers within the company, the sample is the group of volunteers that performed the tasks for the experiment. The following example tries to illustrate the term of sample:

Example 6.1.1. To determine the proportion of people in your town who are smokers, it has been decided to poll people at one of the following local spots:

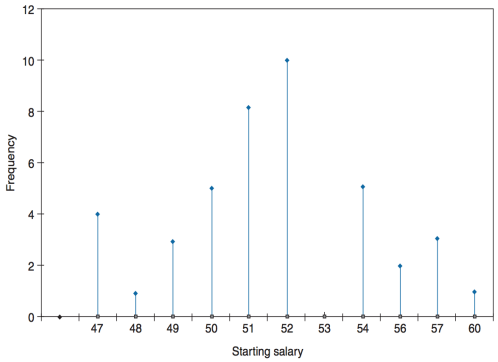
- the pool hall;
- the bowling alley;
- the shopping mall;
- the library.

Which of these potential polling places would most likely result in a reasonable approximation to the desired proportion? Why?

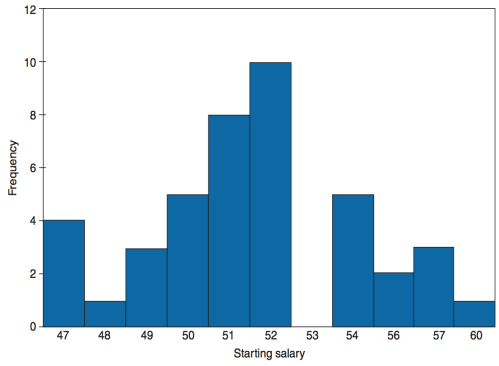
In order to better analyze and interpret the extracted data from our experiment, is important to present it in a proper way. For this reason different types of tables and graphs are used to represent a variety of features from our data. A data set having a relatively small number of distinct values can be conveniently presented in a *frequency table*, as seen in Table 6.1a. Then, data from those tables can be represented in a graphical fashion using different types of graphs. Some examples are presented in Figure 6.1. The term *relative frequency* refers to the ratio f/n, where f stands for te frequency of a particular value and n the number of values of a given data set. That is, the proportion of the data that have that value. A way of representing such values is depicted in Figure 6.2. For the interested reader, we refer to Section 2.2 of [5], where further information can be found with respect to ways of describing data sets.

Starting Salary	Frequency
47	4
48	1
49	3
50	5
51	8
52	10
53	0
54	5
56	2
57	3
60	1

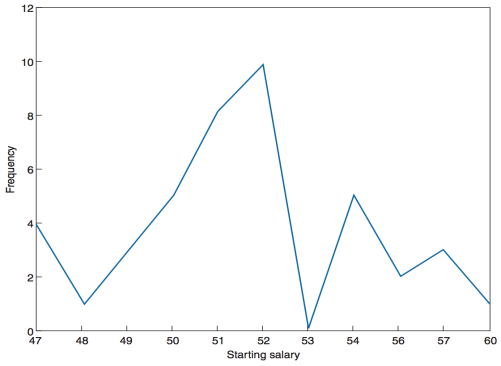
(a) Frequency table



(b) Line graph



(c) Bar graph



(d) Polygon graph

Figure 6.1: Frequency tables and graphs

There is a certain set of measures that are normally used in statistics when dealing with big sets of data. In the following we present some summarizing statistics, where a statistic is a numerical quantity whose value is determined by the data. The first two definitions correspond with statistics that describe the central tendencies of a data set.

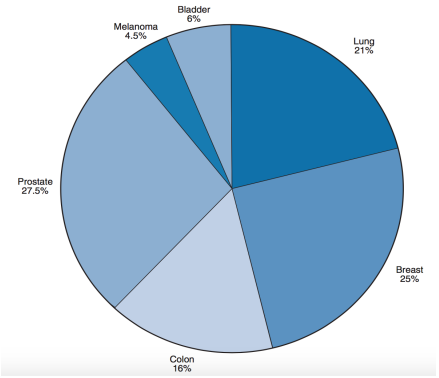
Definition 47. Sample mean. *Consider a data set consisting of the n numerical values $x_1, x_2, ..., x_n$. The sample mean is the arithmetic average of these values.*

$$\bar{x} = \sum_{i=1}^{n} x_i/n$$

Definition 48. Sample median. *Order the values of a data set of size n from smallest to largest. If n is odd, the sample median is the value in position $(n+1)/2$; if n is even, it is the average of the values in positions $n/2$ and $n/2+1$.*

Type of Cancer	Number of New Cases	Relative Frequency
Lung	42	.21
Breast	50	.25
Colon	32	.16
Prostate	55	.275
Melanoma	9	.045
Bladder	12	.06

(a) Relative frequency table



(b) Pie graph

Figure 6.2: Relative frequency tables and graphs

The following definitions describe the spread or variability of the data values.

Definition 49. Sample variance. *Consider a data set consisting of the n numerical values $x_1, x_2, ..., x_n$. The sample variance measures the average value of the squares of the distances between the data values and the sample mean.*

$$s^2 = \sum_{i=1}^{n} (x_i - \bar{x})^2/(n-1)$$

Definition 50. Sample standard deviation. *Consider a data set consisting of the n numerical values $x_1, x_2, ..., x_n$. The sample standard deviation is a measure of the spread of scores within a set of data.*

$$s = \sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2/(n-1)}$$

Example 6.1.2. Consider the set of values $A = 3, 4, 6, 7, 10$ (which means $n = 5$). Use the previously defined definitions to calculate the metrics of the statistics:

1. Sample mean;
2. Sample median;
3. Sample variance;
4. Sample standard deviation.

Solution:

1. Sample mean. Given the data set and the following formula:

$$\bar{x} = \sum_{i=1}^{n} x_i/n$$

$$\bar{x} = \frac{3+4+6+7+10}{5} = 6$$

2. Sample median. We have an odd number of values, so the median is the value in position $(n+1)/2$, so:

$$M = 6$$

3. Sample variance. We already have the value of the Sample mean of our data set, so:

$$s^2 = \sum_{i=1}^{n}(x_i - \bar{x})^2/(n-1)$$

$$\bar{x} = 6$$

$$s^2 = \frac{[(3-6)^2 + (4-6)^2 + (6-6)^2 + (7-6)^2 + (10-6)^2]}{(5-1)} = 7.5$$

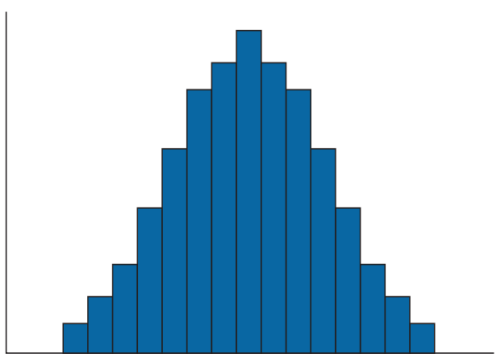
4. Sample standard deviation. Since the Sample standard deviation is the positive squared root of the Sample variance:

$$s = \sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2/(n-1)}$$

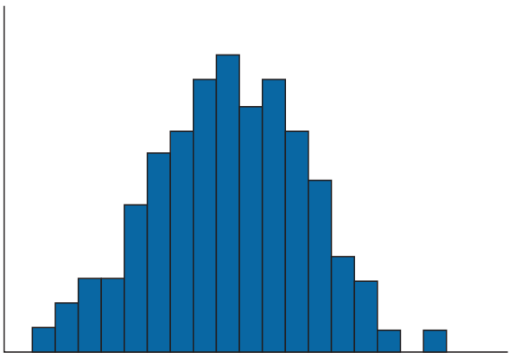
$$s = \sqrt{s^2}$$

$$= \sqrt{7.5}$$

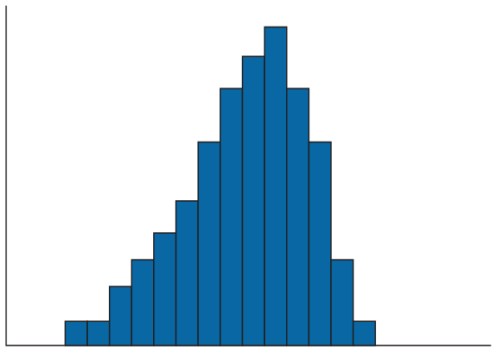
Many of the large data sets observed in practice have histograms that are similar in shape. These histograms often reach their peaks at the sample median and then decrease on both sides of this point in a bell-shaped symmetric fashion. Such data sets are said to be normal and their histograms are called *normal histograms*. An example of such histogram is presented in Figure 6.3a



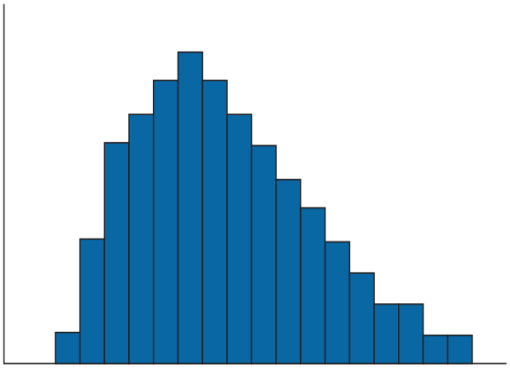
(a) Histogram of a normal data set



(b) Histogram of a approximately normal data set



(c) Left skewed histogram of a data set



(d) Right skewed histogram of a data set

Figure 6.3: Normal data sets

If the histogram of a data set is close to being a normal histogram, then we say that the data set is *approximately normal*, see Figure 6.3b. It follows from the symmetry of the normal histogram that a data set that is approximately normal will have its sample mean and sample median approximately equal. On the other hand, any data set that is not approximately symmetric about its sample median is

said to be skewed. It is "skewed to the right" if it has a long tail to the right (Figure 6.3d) and "skewed to the left" if it has a long tail to the left (Figure 6.3c).

THE EMPIRICAL RULE Considering $\bar{x}$ the sample mean and s the standard deviation of a given approximately normal data set, the *empirical rule* specifies the approximate proportions of the data observations that are within s, 2s, and 3s of the sample mean $\bar{x}$. For such a data set, the following statements are true:

1. Approximately 68 percent of the observations lie within
$$\bar{x} \pm s$$
2. Approximately 95 percent of the observations lie within
$$\bar{x} \pm 2s$$
3. Approximately 99.7 percent of the observations lie within
$$\bar{x} \pm 3s$$

Example 6.1.3. Let's assume a population of animals in a zoo is known to be normally distributed. Each animal lives to be 13.1 years old on average and the standard deviation of lifespan is 1.5 years. If someone wants to know the probability that an animal will live longer than 14.6 years, they could use the empirical rule. Knowing $\bar{x} = 13.1$ and $s = 1.5$, the following age ranges occur for each standard deviation:

1. s: $(13.1 - 1.5)$ to $(13.1 + 1.5)$, or 11.6 to 14.6
2. $2s$: $13.1 - (2 \times 1.5)$ to $13.1 + (2 \times 1.5)$, or 10.1 to 16.1
3. $3s$: $13.1 - (3 \times 1.5)$ to $13.1 + (3 \times 1.5)$, or, 8.6 to 17.6

The person solving this problem needs to calculate the total probability of the animal living 14.6 years or long. The empirical rule shows that 68% of the distribution lies within one standard deviation, in this case, from 11.6 to 14.6 years. Thus, the remaining 32% of the distribution lies outside this range. Half lies above 14.6 and half lies below 11.6. So the probability of the animal living more than 14.6 is 16% (calculated as 32% divided by two).

Another type of data sets that consist of pairs of values that have a relationship are called paired data sets. Such a data set is represented in Table 6.4a. Considering the elements representing the temperature the x values and the elements representing the number of elements as the y values, we can represent the *ith* data point by the pair (xi, yi). A common way of representing paired data sets is by using *scatter diagrams*, as the one in Figure 6.4b

Definition 51. Sample correlation coefficient. *Consider the data pairs $(xi, yi), i = 1, ..., n$ and let s_x and s_y denote, respectively, the sample standard deviations of the x values and the y values. The sample correlation coefficient, call it r, of the data pairs $(xi, yi), i = 1, ..., n$ is defined by:*

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y}$$

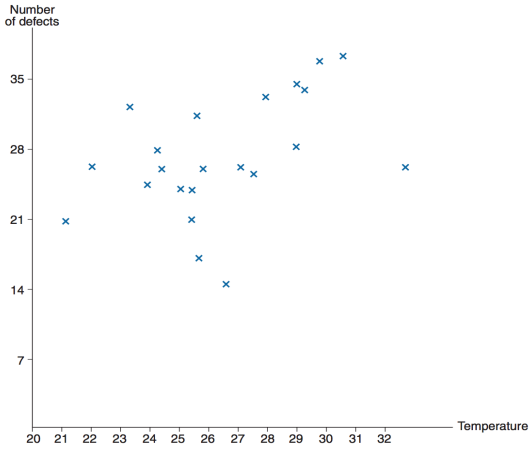
$$= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Properties of r

1. the sample correlation coefficient r is always between 1 and +1
$$-1 \leq r \leq +1$$

Day	Temperature	Number of Defects
1	24.2	25
2	22.7	31
3	30.5	36
4	28.6	33
5	25.5	19
6	32.0	24
7	28.6	27
8	26.5	25
9	25.3	16
10	26.0	14
11	24.4	22
12	24.8	23
13	20.6	20
14	25.1	25
15	21.4	25
16	23.7	23
17	23.9	27
18	25.2	30
19	27.4	33
20	28.3	32
21	28.8	35
22	26.6	24

(a) Paired data set table



(b) A scatter diagram

Figure 6.4: Paired data set and representation

2. r will equal $+1$ when there is a straight line (also called a linear) relation between the paired data such that large y values are attached to large x values. If for constants a and b,with $b > 0$:

$$y_i = a + bx_i, i = 1, ..., n$$

then $r = 1$.

3. r will equal -1 when the relation is linear and large y values are attached to small x values If for constants a and b,with $b < 0$:

$$y_i = a + bx_i, i = 1, ..., n$$

then $r = -1$.

4. the value of r is unchanged when a constant is added to each of the x variables (or to each of the y variables) or when each x variable (or each y variable) is multiplied by a positive constant. This property implies that r does not depend on the dimensions chosen to measure the data. If r is the sample correlation coefficient for the data pairs $x_i, y_i, i = 1, ..., n$ then it is also the sample correlation coefficient for the data pairs

$$a + bx_i, c + dy_i, i = 1, ..., n$$

provided that b and d are both positive or both negative.

The sample correlation coefficient expresses both the *strength* of the linear relationship between the x and the y values of a data pair and its *direction*. A value of $|r| = 1$ (its absolute value, i.e. value without regard to its sign) means that there is a perfect linear relation—that is, a straight line can pass through all the data points $(x_i, y_i), i = 1, ..., n$. A value of $|r| = 0.8$ means that the linear relation is relatively strong; although there is no straight line that passes through all of the data points, there is one that is "close" to them all. A value for $|r| = 0.3$ means that the linear relation is relatively weak. The sign of r is positive when the linear relation is such that smaller y values tend to go with smaller x values and larger y values with larger x values (and so a straight line approximation points upward), and it is negative when larger y values tend to go with smaller x values and smaller y values with larger x values (and so a straight line approximation points downward).

CORRELATION DOES NOT IMPLY CAUSATION Before finishing this section we remark the next statement: *Correlation does not imply causation.* We cannot assume that one causes the other. We might have a case of causation, but we cannot really tell by the statistics. Thus, correlation simply measures association.

6.2 Introduction to probability

The measure of the likelihood that an event will occur is called probability. If an event is unlikely to occur, we say that the probability of this event to happen is low. Vice versa, If an event is likely to occur, we say that the probability of this event to happen is high. The probability of a given event (A) to occur is denoted as $p(A)$. Then, if $p(A) = 0$ it is impossible that event A will occur and if $p(A) = 1$ it is certain that this event will occur. Probability theory is the logic of science. In particular, is a critical component of inductive logic.

Whenever human beings are dealing with probabilistic reasoning, we have a tendency to fall into traps. We call it *probability blindness*. It is the human being's observed inability to make intuitive estimates of the probability of an event, even when presented with all the relevant evidence. As an example, consider a certain invisible medical condition afflicts one person in a thousand. It can be detected before onset by a test that yields 5% false positives. If a randomly selected person tests positive, what is the probability she has the condition? The answer is: just about exactly 2%. Let's consider 100,000 trials. Since the condition afflicts one person in a thousand, you'd expect 100 to test positive on account of having the condition. Of the remaining 99,900, five per cent, or 4995, would test positive. So the ratio of afflicted to all positive results is given by 100/5095, or 0.0196. When trying to solve this problem, people tend to discount the low incidence of the condition in favor of elevating the significance of a quite unreliable test.

The set of all possible outcomes (i.e. one of the individual results that can occur in an experiment, i.e. something that is done that produces measurable results) of an experiment is known as the *sample space* of the experiment and is denoted by S. It can be defined as in the following examples:

1. If the outcome of an experiment consists in the determination of the sex of a newborn child, then

$$S = \{g, b\}$$

   where the outcome g means that the child is a girl and b that it is a boy.

2. If the experiment consists of the running of a race among the seven horses having post positions 1, 2, 3, 4, 5, 6, 7, then

$$S = \{\text{all orderings of } (1, 2, 3, 4, 5, 6, 7)\}$$

   The outcome (2, 3, 1, 6, 5, 4, 7) means, for instance, that the number 2 horse is first, then the number 3 horse, then the number 1 horse, and so on.

Any subset E of the sample space is known as an *event*. That is, an event is a set consisting of possible outcomes of the experiment. If the outcome of the experiment is contained in E, then we say that E has occurred. For instance, in the previously considered example, if $E = \{g\}$, then E is the event that the child is a girl.

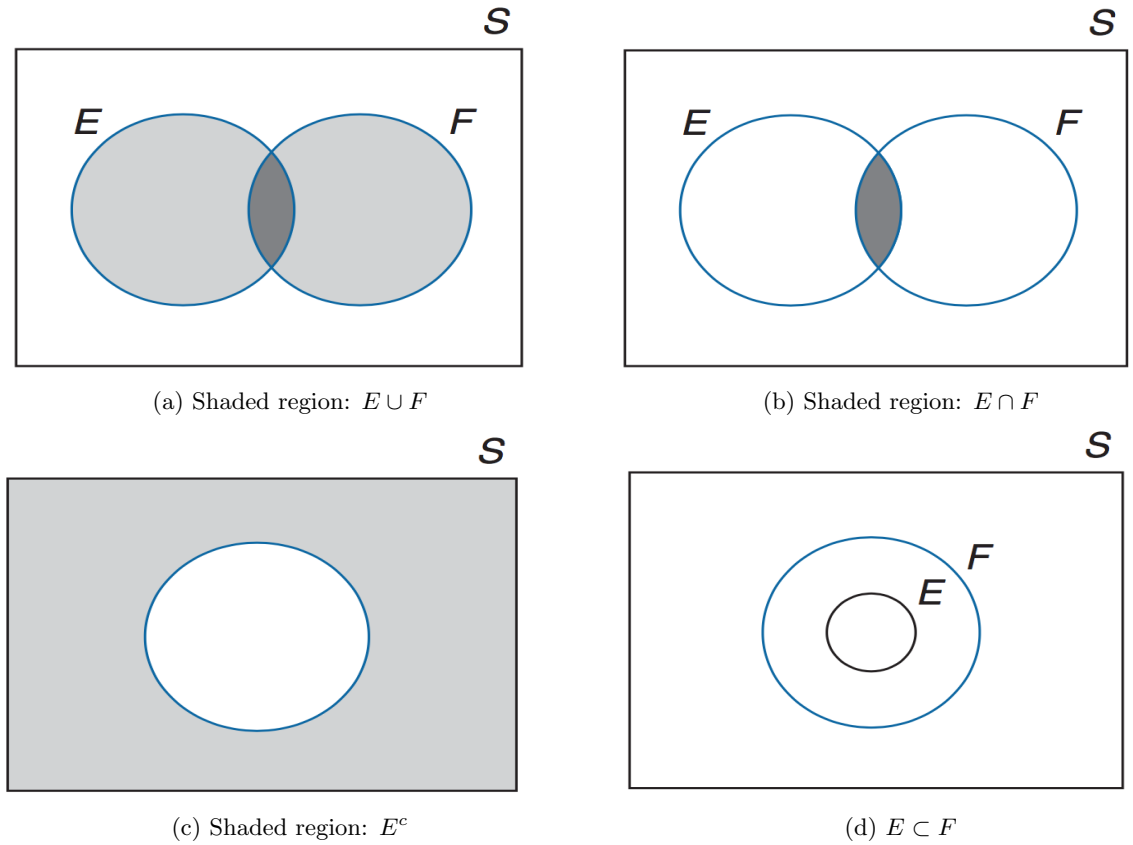
For any two events E and F of a sample space S, we define the new event $E \cup F$, called the *union* of the events E and F. it consist of all outcomes that are either in E or in F or in both E and F. That is, the event $E \cup F$ will occur if either E or F occurs. A graphical representation of such event is represented in Figure 6.5a. For instance, following the same example, if $E = \{g\}$ and $F = \{b\}$, then $E \cup F = \{g, b\}$.

We can also define the new event EF, called the *intersection* of E and F (sometimes expressed as $E \cap F$), to consist of all outcomes that are in both E and F. A graphical representation of such event is represented in Figure 6.5b. In our example, the event EF does not contain any outcome since it is not possible for a baby to be born both a girl and a boy. We refer to such an event as *null event* and denote

it by Ø. If $EF = Ø$, implying that E and F cannot both occur, then E and F are said to be *mutually exclusive*.

For any event E, we define the event E^c, referred to as the *complement* of E, to consist of all outcomes in the sample space S that are not in E. That is, E^c will occur if and only if E does not occur. A graphical representation of such event is represented in Figure 6.5c. In our example, if $E = b$ is the event that the child is a boy, then $E^c = g$ is the event that it is a girl. Also note that since the experiment must result in some outcome, it follows that $S^c = Ø$.

For any two events E and F, if all of the outcomes in E are also in F, then we say that E is contained in F and write $E \subset F$ (or equivalently, $F \supset E$). A graphical representation of such event is represented in Figure 6.5d. Thus if $E \subset F$, then the occurrence of E necessarily implies the occurrence of F. If $E \subset F$ and $F \supset E$, then we say that E and F are equal (or identical) and we write $E = F$.



(a) Shaded region: $E \cup F$

(b) Shaded region: $E \cap F$

(c) Shaded region: E^c

(d) $E \subset F$

Figure 6.5: Graphical representation of events

AXIOMS OF PROBABILITY From a purely mathematical viewpoint, we shall suppose that for each event E of an experiment having a sample space S there is a number, denoted by P(E), that is in accord with the following three axioms.

1. Axiom 1: the probability that the outcome of the experiment is contained in E is some number between 0 and 1.
$$0 \leq P(E) \leq 1$$
2. Axiom 2: with probability 1, the outcome will be a member of the sample space S.
$$P(S) = 1$$
3. Axiom 3: For any sequence of mutually exclusive events E1, E2, ..., ∞ (that is, events for

which $E_i, E_j = \emptyset$ when $i \neq j$),

$$P(\bigcup_{i=1}^{n} E_i) = \sum_{i=1}^{n} P(E_i), n = 1, 2, ..., \infty$$

We call P(E) the probability of the event E. Meaning that for any set of mutually exclusive events the probability that at least one of these events occurs is equal to the sum of their respective probabilities.

For the next proofs we will now consider an event E and its complement E^c. They are mutually exclusive, so $E \cup E^c = S$. By axioms 2 and 3:

$$1 = P(S) = E \cup E^c = P(E) + P(E^c)$$

Then:

$$P(E^c) = 1 - P(E)$$

Thus, if the probability of a baby to be born a girl is 2/5, the probability of being born a boy is 3/5. The next proposition gives the relationship between the probability of the union of two events in terms of the individual probabilities and the probability of the intersection:

$$P(E \cup F) = P(E) + P(F) - P(EF)$$

The *odds* of an event A tells how much more likely it is that A occurs than that it does not occur. It is defined by:

$$\frac{P(A)}{P(A^c)} = \frac{P(A)}{1 - P(A)}$$

As in the previous example, if the probability of a baby to be born a girl (P(A)) is 2/5, then $\frac{2/5}{1 - 2/5} = 2/3$. Consequently, it is 2/3 times as likely that A occurs as it is that it does not.

Example 6.2.1. A total of 28 percent of American males smoke cigarettes, 7 percent smoke cigars, and 5 percent smoke both cigars and cigarettes. What percentage of males smoke neither cigars nor cigarettes?

Solution Let E be the event that a randomly chosen male is a cigarette smoker and let B be the event that he is a cigar smoker. Then, the probability this person is either a cigarette or a cigar smoker is:

$$P(E \cup F) = P(E) + P(F) - P(EF) = .07 + .28 - .05 = .3$$

Thus the probability that the person is not a smoker is .7, implying that 70 percent of American males smoke neither cigarettes nor cigars.

For a large number of experiments whose sample space s is $s = \{1, 2, ..., N\}$, it is natural to assume that each point in the sample space is equally likely to occur. Then, the probability of any event E equals the proportion of points in the sample space that are contained in E:

$$P(E) = \frac{\text{Number of points in E}}{N}$$

Thus, to compute probabilities it is often necessary to be able to effectively count the number of different ways that a given event can occur. Suppose that two experiments are to be performed. Then if experiment 1 can result in any one of m possible outcomes and if, for each outcome of experiment 1, there are n possible outcomes of experiment 2, then together there are mn possible outcomes of the two experiments.

We will now introduce the concept of *conditional probability*. It is useful when we are interested in calculating probabilities when some partial information concerning the result of the experiment is

available, or in recalculating them in light of additional information. In such situations, the desired probabilities are conditional ones. Conditional probability is also useful when it turns out that the easiest way to compute the probability of an event is to first "condition" on the occurrence or nonoccurrence of a secondary event. The denotation of conditional probability pf E given that F has occurred is $P(E|F)$. A general formula for $P(E|F)$ that is valid for all events E and F is the following:

$$P(E|F) = \frac{P(EF)}{P(F)}$$

Namely, if the event F occurs, then in order for E to occur it is necessary that the actual occurrence be a point in both E and F; that is, it must be in EF. Now, since we know that F has occurred, it follows that F becomes our new (reduced) sample space and hence the probability that the event EF occurs will equal the probability of EF relative to the probability of F.

Example 6.2.2. Two balls are *randomly drawn* from a bowl containing 6 white and 5 black balls. What is the probability that one of the drawn balls is white and the other one black?

Solution

If we regard the order in which the balls are selected as being significant, then as the first drawn ball may be any of the 11 and the second any of the remaining 10, it follows that the sample space consists of $11{\cdot}10 = 110$ points. Furthermore, there are $6{\cdot}5 = 30$ ways in which the first ball selected is white and the second black, and similarly there are $5{\cdot}6 = 30$ ways in which the first ball is black and the second white. Hence, assuming that randomly drawn means that each of the 110 points in the sample space is equally likely to occur. Then we see that the desired probability of the event E ("draw a white ball and a black ball") is:

$$P(E) = \frac{\text{Number of points in E}}{N} = \frac{30+30}{121}$$

The problem can be redefined so the event E is "draw first a white ball and then a black ball replacing the first ball". The event $A = \{A_1, A_2, ..., A_i, i = 1, 2, ..., 6\}$ is defined as"draw any white ball" $B = \{B_1, B_2, ..., B_j, j = 1, 2, ..., 5\}$ as "draw any black marble". Note that A is the complement of B. Then, the sample space (draw any marble) is defined as $S = \{A_1, A_2, ..., A_i, B_1, B_2, ..., B_j, i = 1, 2, ..., 6, j = 1, 2, ..., 5\}$. One picked, the first ball will be reintroduced in the bowl, so the outcome of the first event does not affect the second one. The probabilities of both events are then:

$$p(A) = \frac{n(A)}{n(S)} = \frac{6}{11} \text{ and } p(B) = \frac{n(B)}{n(S)} = \frac{5}{11}$$

Let's redefine the problem again. Now, the first ball will not be reintroduced in the bowl once picked, so the first event will affect the second one. The event E is now "draw first a white ball and then a black ball without replacing the first ball". The probability of such event to occur may be defined as:

$$\begin{aligned} p(E) &= p((2^{nd} \text{ is black } |A) \cap A) \\ &= p(2^{nd} \text{ is black } |A) \cdot (A) \\ &= \frac{4}{10} \cdot \frac{6}{11} \end{aligned}$$

We refer to the videos that we made available on the platform (canvas) to check another example of the principle of counting.

GENERALIZED BASIC PRINCIPLE OF COUNTING If r experiments that are to be performed are such that the first one may result in any of n_1 possible outcomes, and if for each of these n_1 possible outcomes there are n_2 possible outcomes of the second experiment, and if for each of the possible outcomes of the first two experiments there are n_3 possible outcomes of the third experiment, and if, ... , then there are a total of $n1 \cdot n2 \cdots n_r$ possible outcomes of the r experiments.

Suppose now that we have n objects. The reasoning shows that there are

$$n(n-1)(n-2)\cdots 3\cdot 2\cdot 1$$

different permutations of the n objects. It is convenient to introduce the notation n!, which is read n factorial, for the foregoing expression. That is,

$$n! = n(n-1)(n-2)\cdots 3\cdot 2\cdot 1$$

It can be easily explained with the following example.

Example 6.2.3. Mr. Jones has 10 books that he is going to put on his bookshelf. Of these, 4 are mathematics books, 3 are chemistry books, 2 are history books, and 1 is a language book. Jones wants to arrange his books so that all the books dealing with the same subject are together on the shelf. How many different arrangements are possible?

Solution There are 4! 3! 2! 1! arrangements such that the mathematics books are first in line, then the chemistry books, then the history books, and then the language book. Similarly, for each possible ordering of the subjects, there are 4! 3! 2! 1! possible arrangements. Hence, as there are 4! possible orderings of the subjects, the desired answer is 4! 4! 3! 2! 1!=6,912.

Suppose now that we are interested in determining the number of different groups of r objects that could be formed from a total of n objects. In general, as $n(n1)\cdots(nr+1)$ represents the number of different ways that a group of r items could be selected from n items when the order of selection is considered relevant; and since each group of r items will be counted r! times in this count, it follows that the number of different groups of r items that could be formed from a set of n items is:

$$\frac{n(n1)\cdots(nr+1)}{r!} = \frac{n!}{(n-r)!r!}$$

The number of *combinations* of n objects taken r at a time are denoted as $\binom{n}{r}$, for $r \geq n$. It is defined by:

$$\binom{n}{r} = \frac{r!}{(n-r)!r!}$$

and represents the number of different groups of size r that can be selected from a set of size n when the order of selection is not considered relevant.

Example 6.2.4. A committee of size 5 is to be selected from a group of 6 men and 9 women. I f the selection is made randomly, what is the probability that the committee consists of 3 men and 2 women?
Solution Let us assume that "randomly selected" means that each of the $\binom{1}{5}5)$ possible combinations is equally likely to be selected. Hence, since there are $\binom{6}{3}$ possible choices of 3 men and $\binom{9}{2}$ possible choices of 2 women, it follows that the desired probability is given by:

$$\frac{\binom{6}{3}\binom{9}{2}}{\binom{15}{5}} = \frac{240}{1001}$$

BAYES FORMULA Let E and F be events. We may express E as

$$E = EF \cup EF^c$$

for, in order for a point to be in E, it must either be in both E and F or be in E but not in F (see Figure 6.6) As EF and EF c are clearly mutually exclusive, we have by Axiom 3 that

$$P(E) = P(EF) + P(EF^c)P(E) = P(E|F)P(F) + P(E|F^c)P(F^c)$$

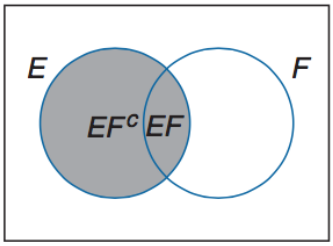


Figure 6.6: $E = EF \cup EF^c$

$$= P(E|F)P(F) + P(E|F^c)[1 - P(F)]$$

Example 6.2.5. A laboratory blood test is 99 percent effective in detecting a certain disease when it is, in fact, present. However, the test also yields a "false positive" result for 1 percent of the healthy persons tested. (That is, if a healthy person is tested, then, with probability .01, the test result will imply he or she has the disease.) If .5 percent of the population actually has the disease, what is the probability a person has the disease given that his test result is positive?

Solution Let E be the event that the tested person has the disease and F the event that his test result is positive. The desired probability $P(E|F)$ is obtained by

$$P(E|F) = \frac{P(EF)}{P(F)}$$

and

$$P(E) = P(E|F)P(F) + P(E|F^c)[1 - P(F)]$$

then

$$P(E|F) = \frac{P(F|E)P(E)}{P(F|E)P(E) + P(F|E^c)P(E^c)}$$

$$= \frac{(.99)(.005)}{(.99)(.005) + (.01)(.995)} = .3322$$

Thus, only 33 percent of those persons whose test results are positive actually have the disease.

6.3 Random variables and expectation

Very often, when we conduct experiments in Software Engineering, we have non deterministic elements as part of our systems. Non determinism is basically the given an input x to our system, the system will give different outputs $y_1, ..., y_n$. In other words, a non deterministic system behaves in a random way. Or to put it differently, if we know exactly how something or someone always will behave there is reason to validate it or run an experiment. *Random variables* are denoted with upper case letter (X, Y, Z), while in algebra we denote ordinary variables with lower case letters (x, y, z). In algebra, the ordinary variables stand for specific values, or a function of an expression. Let's consider the following example:

$$x + 4 = 5$$

$$y = x + 4$$

In the first line, we can calculate the specific value of x. In the second one, if we provide x with value y will change. That is, y will change as a function of x. This is not how random variables work. As an example, flipping a fair coin gives as a heads or a tails. We can denote the random variable X as 1 if the result is heads and 0 if it is tails. We have taken a random process and mapped its outcome by quantifying it, i.e. providing numbers to it. The good thing with doing this is that we now have a

rough model of the random process, which we later might use when calculating probabilities. Another good thing is that we now have a nifty way of expressing probabilities in short hand. Let's consider the following example to illustrate this. Letting X denote the random variable that is defined as the sum of two fair dice, then:

$$
\begin{aligned}
P\{X=2\} &= P\{(1,1)\} = \frac{1}{6\cdot 6} = \frac{1}{36} \\
P\{X=3\} &= P\{(1,2),(2,1)\} = \frac{2}{36} \\
P\{X=4\} &= P\{(1,3),(2,2),(3,1)\} = \frac{3}{36} \\
P\{X=5\} &= P\{(1,4),(2,3),(3,2),(4,1)\} = \frac{4}{36} \\
P\{X=6\} &= P\{(1,5),(2,4),(3,3),(4,2),(5,1)\} = \frac{5}{36} \\
P\{X=7\} &= P\{(1,6),(2,5),(3,4),(4,3),(5,2),(6,1)\} = \frac{6}{36} \\
P\{X=8\} &= P\{(2,6),(3,5),(4,4),(5,3),(6,2)\} = \frac{5}{36} \\
P\{X=9\} &= P\{(3,6),(4,5),(5,4),(6,3)\} = \frac{4}{36} \\
P\{X=10\} &= P\{(4,6),(5,5),(6,4)\} = \frac{3}{36} \\
P\{X=11\} &= P\{(5,6),(6,5)\} = \frac{2}{36} \\
P\{X=12\} &= P\{(6,6)\} = \frac{1}{36}
\end{aligned}
\tag{6.1}
$$

In this case, the random variables of interest could only take on a finite number of possible values.

Random variables whose set of possible values can be written either as a finite sequence x1, x2, xn or as an infinite sequence x1, x2, ... The previous example of the dice the random variable is said to be discrete, or, in other words, distinct values. The variable can assume values from 2 to 12, nothing else. Discrete variables consist of sequences. There are also random variables that can take on a continuum of possible variables. These are known as *continuous random variables*. One example is a car lifetime, which is assumed to take on any value in some interval (a, b). Hence, discrete variables consist on sequences while continuous variables consist of intervals. While the probability function of discrete variables is called the *probability mass function*, for continuous variables is called the *probability density function*. For a discrete random variable X, we define the probability mass function $p(a)$ of X by

$$p(a) = P\{X = a\}$$

Since X can take on any of the distinct values x_i, the sum of the probabilities of X assuming variables x_i is, of course, 1:

$$\sum_{i=1}^{\infty} p(x_i) = 1$$

For example, consider a random variable X that is equal to 1, 2, or 3. If we know that:

$$p(1) = \frac{1}{2} \text{ and } p(2) = \frac{1}{3}$$

then it follows (since $p(1) + p(2) + p(3) = 1$) that:

$$p(3) = \frac{1}{6}$$

The probability density function of the random continuous variable X is defined by:

$$P\{X \in B\} = \int_B f(x)dx$$

The equation states that there exists an non negative function $f(x)$ with the following property, that for any set of B of real numbers. The $f(x)$, in this case is called the probability density function of the random variable X. Since X must assume some value, $f(x)$ must satisfy:

$$1 = P\{X \in (-\infty, \infty)\} = \int_{-\infty}^{\infty} f(x)dx$$

Any probability statement about x can be answered in terms of $f(x)$. If we are dealing with continuous variables—i.e. variables in an interval—then if we let $B = [a, b]$, we get the probability of a certain value in the range a to b:

$$P\{a \leq X \leq b\} = \int_a^b f(x)dx$$

Example 6.3.1. Suppose that X is a continuous random variable whose probability density function is given by

$$f(x) = \begin{cases} C(4x - 2x^2) & 0 < x < 2 \\ 0 & \text{otherwise} \end{cases}$$

a) What is the value of C?

b) Find $P\{X > 1\}$.

Solution

a) Since f is a probability density function, we must have that $\int_{-\infty}^{\infty} f(x)dx = 1$, implying that

$$C \int_0^2 (4x - 2x^2)dx = 1$$

or

$$C\left[2x^2 - \frac{2x^3}{3}\right]\Big|_{x=2}^{x=0} = 1$$

or

$$C = \frac{3}{8}$$

b) Hence

$$P\{X > 1\} \int_1^{\infty} f(x)dx = \frac{3}{8} \int_1^2 (4x - 2x^2)dx = \frac{1}{2}$$

For a given experiment, we are often interested not only in probability distribution functions of individual random variables but also in the relationships between two or more random variables. This is called *jointly distributed random variables*, or *joint distribution*. Roughly speaking, joint distribution come in three flavors:

1. Joint distributions having two or more discrete variables
2. Joint distributions that have two or more continuous variables
3. Joint distributions that are a mix of two or more discrete and continuous variables.

i \ j	0	1	2	3	Row Sum $= P\{B = i\}$
0	.15	.10	.0875	.0375	.3750
1	.10	.175	.1125	0	.3875
2	.0875	.1125	0	0	.2000
3	.0375	0	0	0	.0375
Column Sum $= P\{G = j\}$	.3750	.3875	.2000	.0375	

Figure 6.7: $P\{B = i, G = j\}$

In the following example we illustrate the first case. Suppose that 15 percent of the families in a certain community have no children, 20 percent have 1, 35 percent have 2, and 30 percent have 3 children; suppose further that each child is equally likely (and independently) to be a boy or a girl. If a family is chosen at random from this community, then B, the number of boys, and G, the number of girls, in this family will have the joint probability mass function shown in Table 6.7.

These probabilities are obtained as follows:

$$\begin{aligned}
P\{B = 0, G = 0\} &= P\{\text{no children}\} = .15 \\
P\{B = 0, G = 1\} &= P\{\text{1 girl and total of 1 child}\} \\
&= P\{\text{1 child}\}P\{\text{1 girl—1 child}\} \\
&= (.20)(\frac{1}{2}) = .1 \\
P\{B = 0, G = 2\} &= P\{\text{2 girls and total of 2 children}\} \\
&= P\{\text{2 children}\}P\{\text{2 girls|2 children}\} \\
&= (.35)(\frac{1}{2})^2 = .0875 \\
P\{B = 0, G = 3\} &= P\{\text{3 girls and total of 3 children}\} \\
&= P\{\text{3 children}\}P\{\text{3 girls|3 children}\} \\
&= (.30)(\frac{1}{2})^3 = .0375
\end{aligned}$$

One of the most important concepts in probability theory is that of the expectation of a random variable. If X is a discrete random variable taking on the possible values $x_1, x_2, ...$, then the *expectation* or *expected value* of X, denoted by $E[X]$, is defined by:

$$E[X] = \sum_i x_i P\{X = x_i\}$$

In words, the expected value of X is a weighted average of the possible values that X can take on, each value being weighted by the probability that X assumes it. Let's illustrate this concept with an example.

Example 6.3.2. Find $E[X]$ where X is the outcome when we roll a fair die.
Solution Since $p(1) = p(2) = p(3) = p(4) = p(5) = p(6) = 1$, we obtain that

$$E[X] = 1(\frac{1}{6}) + 2(\frac{1}{6}) + 3(\frac{1}{6}) + 4(\frac{1}{6}) + 5(\frac{1}{6}) + 6(\frac{1}{6}) = \frac{7}{2}$$

For this example, the expected value of X is not a value that X could possibly assume. (That is, rolling a die cannot possibly lead to an outcome of 7/2). Thus, even though we call $E[X]$ the *expectation* of X,

it should not be interpreted as the value that we *expect* X to have but rather as the average value of X in a large number of repetitions of the experiment. That is, if we continually roll a fair die, then after a large number of rolls the average of all the outcomes will be approximately 7/2.

We can also define the expectation of a continuous random variable. Suppose that X is a continuous random variable with probability density function f. Hence, it is natural to define the expected value of X by:

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx$$

Again, we will illustrate this concept with an example.

Example 6.3.3. Suppose that you are expecting a message at some time past 5 P.M. From experience you know that X , the number of hours after 5 P.M. until the message arrives, is a random variable with the following probability density function:

$$f(x) = \begin{cases} \dfrac{1}{1.5} & 0 < x < 1.5 \\ 0 & \text{otherwise} \end{cases}$$

The expected amount of time past 5 P.M. until the message arrives is given by:

$$E[X] = \int_0^{1.5} \frac{x}{1.5}dx = .75$$

Hence, on average, you would have to wait three-fourths of an hour.

We will discuss now about the properties of the expected value. Suppose now that we are given a random variable X and its probability distribution (that is, its probability mass function in the discrete case or its probability density function in the continuous case). Suppose also that we are interested in calculating, not the expected value of X , but the expected value of some function of X , say $g(X)$. How do we do that? Since $g(X)$ is itself a random variable, it must have a probability distribution, which should be computable from a knowledge of the distribution of X. Once we have obtained the distribution of $g(X)$, we can then compute $E[g(X)]$ by the definition of the expectation. We will solve two examples of the properties of discrete and continuous expected values, respectively.

Example 6.3.4. Suppose X has the following probability mass function:

$$p(0) = .2, p(1) = .5, p(2) = .3$$

Calculate $E[X^2]$. **Solution** Letting $Y = X^2$, we have that Y is a random variable that can take on one of the values $02, 1^2, 2^2$ with respective probabilities

$$p_Y(0) = P\{Y = 0^2\} = .2$$
$$p_Y(1) = P\{Y = 1^2\} = .5$$
$$p_Y(4) = P\{Y = 2^2\} = .3$$

Hence,

$$E[X^2] = E[Y] = 0(.2) + 1(.5) + 4(.3) = 1.7$$

Example 6.3.5. The time, in hours, it takes to locate and repair an electrical breakdown in a certain factory is a random variable—call it X—whose density function is given by:

$$f(x) = \begin{cases} 1 & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}$$

If the cost involved in a breakdown of duration x is x^3, what is the expected cost of such a breakdown?

Solution Letting $Y = X^3$ denote the cost, we first calculate its distribution function as follows. For $0 < a < 1$,

$$\begin{aligned} F_Y(a) &= P\{Y \leq a\} = P\{X^3 \leq a\} \\ &= P\{X \leq a^{1/3}\} \\ &= \int_0^{1/3} dx = a^{1/3} \end{aligned}$$

By differentiating $F_Y(a)$, we obtain the density of Y,

$$f_Y(a) = \frac{1}{3}a^{2/3}, 0 < a < 13$$

Hence,

$$\begin{aligned} E[X^3] = E[Y] &= \int_{-\infty}^{\infty} a f_Y(a) da \\ &= \int_0^1 a\frac{1}{3}a^{-2/3}da = \frac{1}{3}\int_0^1 a^{1/3} \\ &= \frac{1}{3}\frac{3}{4}a4/3|_0^1 = 1/4 \end{aligned}$$

Since g(X) takes on the value g(x) when X = x, it seems intuitive that E[g(X)] should be a weighted average of the possible values g(X) with, for a given x, the weight given to g(x) being equal to the probability (or probability density in the continuous case) that X will equal x

Definition 52. *a) If X is a discrete random variable with probability mass function p(x), then for any real-valued function g,*

$$E[g(X)] = \sum_x g(x)p(x)$$

b) If X is a continuous random variable with probability density function f(x), then for any real-valued function g,

$$E[g(X)] = \int_{-\infty}^{\infty} g(x)f(x)dx$$

Given these two new formulas, we can solve again the last two examples (Example 6.3.4 and Example 6.3.5) in a simpler way:

a)

$$\begin{aligned} E[g(X)] &= \sum_x g(x)p(x) \\ E[X^2] &= 0^2(.2) + 1^2(.5) + 2^2(.3) = 1.7 \end{aligned}$$

b)

$$\begin{aligned} E[g(X)] &= \int_{-\infty}^{\infty} g(x)f(x)dx \\ E[x^3] &= \int_0^1 x^3 dx \text{ (since } f(x) = 1, 0 < x < 1 = \frac{1}{4}) \end{aligned}$$

Another property of random variables is introduced now. Suppose a and b, which are consonants. Then:

$$E[aX + b] = aE[X] + b$$

This is applicable for both discrete and the continuous cases.

The two-dimensional version of the proposition introduced in Definition 52 states that if X and Y are random variables and g is a function of two variables, then

$$\begin{aligned} E[g(X,Y)] &= \sum_y \sum_x g(x,y)p(x,y) \text{ in the discrete case} \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x,y)f(x,y)dxdy \text{ in the continuous case} \end{aligned}$$

For both the continuous and the discrete cases, for any random variables X and Y can be shown that:

$$E[X + Y] = E[X] + E[Y]$$

Expressed in a general fashion, for any n:

$$E[X_1 + X_2 \cdots + X_n] = E[X_1] + E[X_2] + \cdots + E[X_n]$$

For the interested reader, the demonstration of the previous formula can be checked in Section 4.5.1 in[5].

While E[X] yields the weighted average of the possible values of X , it does not tell us anything about the variation, or spread, of these values. How do we calculate the *variance* of X? We ca expect that X takes on values around its mean, which is E[X]. So, we might want to know how far apart X would be from its mean on average. One common way is to consider the expectation of the square difference between X and its mean. If X is a random variable with mean μ, then the variance of X denoted $Var(X)$ is defined by the following formula:

$$Var(X) = E[(X - \mu)^2]$$

However, in practice, we usually use the following formula since is the easiest way of calculating the variance of a random variable.

$$Var(X) = E[X^2] - (E[X])^2$$

The following example is used to illustrate this new concept.

Example 6.3.6. Compute Var(X) when X represents the outcome when we roll a fair die.

Solution Since $PX = i = 1, i = 1, 2, 3, 4, 5, 6$, we obtain

$$\begin{aligned} E[X^2] &= \sum_{i-1}^{6} i^2 P\{X = i\} \\ &= 1^2(\frac{1}{6}) + 2^2(\frac{1}{6}) + 3^2(\frac{1}{6}) + 4^2(\frac{1}{6}) + 5^2(\frac{1}{6}) + 6^2(\frac{1}{6}) = \frac{91}{6} \end{aligned}$$

6.4 Special random variables and sampling

Certain types of random variables occur over and over again in applications. Here we will cover the most common ones and when to use them.

6.4.1 The Bernoulli distribution

Imagine an experiment where the outcome is either success or failure, indicated by the values 1 or 0. Think of it as a coin toss, possibly with an unfair coin with different probabilities of showing heads and tails (as oppose to fair coins). If we let X = 1 when the outcome is a success and X = 0 when it is a failure, then the probability mass function of X is given by

$$P\{X = 0\} = 1 - pP\{X = 1\} = p$$

where p, $0 \leq p \geq 1$, is the probability that the trial is a "success". The expectation is:

$$E[X] = 1 \cdot PX = 1 + 0 \cdot PX = 0 = p$$

6.4.2 The Binomial distribution

The *binomial distribution* is a Bernoulli distribution which we run over and over again. It is supposed to symbolize an experiment in which we will run repeated trials that are independent of each other. If X represents the number of successes that occur in the n trials, then X is said to be a binomial random variable with parameters (n, p). The probability mass function of a binomial random variable with parameters n and p is given by:

$$PX = i = \binom{n}{i} p^i (1-p)^{n-i}, i = 0, 1, ..., n$$

where $\binom{i}{n}$ is the number of different groups of i objects that can be chosen from a set of n objects—as seen in Section 6.2.

Since a binomial random variable X, with parameters n and p, represents the number of successes in n independent trials, each having success probability p, we can represent X as follows:

$$X = \sum_{i=1}^{n} X_i$$

where

$$X_i = \begin{cases} 1 & \text{if the ith trial is a success} \\ 0 & \text{otherwise} \end{cases}$$

Because the $X_i, i = 1, ..., n$ are independent Bernoulli random variables, we have that

$$\begin{aligned} E[X_i] &= PX_i = 1 = p \\ Var(X_i) &= E[X_i^2] - p^2 \\ &= p(1-p) \end{aligned}$$

Thus

$$\begin{aligned} E[X] &= \sum_{i=1}^{\infty} E[Xi] = np \\ Var(X) &= \sum_{i=1}^{\infty} Var(X_i) \text{since the } X_i \text{ are independent} \\ &= np(1-p) \end{aligned}$$

Example 6.4.1. The color of ones eyes is determined by a single pair of genes, with the gene for brown eyes being dominant over the one for blue eyes. This means that an individual having two blue-eyed genes will have blue eyes, while one having either two brown-eyed genes or one brown-eyed and one blue-eyed gene will have brown eyes. When two people mate, the resulting offspring receives one randomly

chosen gene from each of its parents gene pair. If the eldest child of a pair of brown-eyed parents has blue eyes, what is the probability that exactly two of the four other children (none of whom is a twin) of this couple also have blue eyes?

Solution To begin, note that since the eldest child has blue eyes, it follows that both parents must have one blue-eyed and one brown-eyed gene. (For if either had two brown-eyed genes, then each child would receive at least one brown-eyed gene and would thus have brown eyes.) The probability that an offspring of this couple will have blue eyes is equal to the probability that it receives the blue-eyed gene from both parents, which is $(1/2)(1/2) = 1/4$. Hence, because each of the other four children will have blue eyes with probability $1/4$, it follows that the probability that exactly two of them have this eye color is

$$\binom{4}{2}(1/4)^2(3/4)^2 = 27/128$$

6.4.3 The Poisson distribution

Some examples of random variables that usually obey the Poisson probability law are the number of people in a community living 100 years of age or more or the number of transistors that fail on their first day of use. In the examples, we talk of large amounts of data points where there is a very small probability that it has happened. A random variable X, taking on one of the values 0, 1, 2,..., is said to be a Poisson random variable with parameter λ, $\lambda > 0$, if its probability mass function is given by

$$P\{X = i\} = e^{\lambda}\frac{\lambda^i}{i!}, i = 0, 1, ...$$

Also, both the mean and the variance of a Poisson random variable are equal to the parameter λ.

$$E[X] = \lambda$$
$$Var(X) = \lambda$$

For the interested reader, the demonstration of the previous equations can be checked in Chapter 5.2 in [5]

Example 6.4.2. Consider an experiment that consists of counting the number of particles given off in a one-second interval by one gram of radioactive material. If we know from past experience that, on the average, 3.2 such $\alpha-$particles are given off, what is a good approximation to the probability that no more than 2 $\alpha-$particles will appear?

Solution If we think of the gram of radioactive material as consisting of a large number n of atoms each of which has probability 3.2/n (then $p = 3.2/n$) of disintegrating and sending off an $\alpha-$particle during the second considered, then we see that, to a very close approximation, the number of $\alpha-$particles given off will be a Poisson random variable with parameter $\lambda = 3.2$ (because $\lambda = np$ and $p = 3.2/n$). Hence the desired probability is:

$$P\{X \leq 2\} = P\{X = 0\} + P\{X = 1\} + P\{X = 2\}$$ since we have to calculate the probability of 2 or less particles

$$P\{X \leq 2\} = e^{-3.2} + 3.2e^{-3.2} + \frac{(3.2)^2}{2}e^{-3.2}$$
$$= 0.382$$

6.4.4 The Normal distribution

A random variable is said to be normally distributed with parameters μ and σ^2, and we write $X \sim \mathcal{N}(\mu, \sigma^2)$, if its density is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, -\infty < x < \infty$$

The normal density f(x) is a bell-shaped curve that is symmetric about μ and that attains its maximum value of $\frac{1}{\sqrt{2\pi}\sigma} \approx 0.399/\sigma$ at $x = \mu$. Its expectation and variance are define by:

$$E[X] = \mu$$
$$Var(X) = \sigma^2$$

For the interested reader, the demonstration of the previous equations can be checked in Chapter 5.5 in [5].

The normal distribution was introduced by the French mathematician Abraham de Moivre in 1733 and was used by him to approximate probabilities associated with binomial random variables when the binomial parameter n is large. This result was later extended by Laplace and others and is now encompassed in a probability theorem known as the central limit theorem, which gives a theoretical base to the often noted empirical observation that, in practice, many random phenomena obey, at least approximately, a normal probability distribution. Some examples of this behavior are the height of a person, the velocity in any direction of a molecule in gas, and the error made in measuring a physical quantity.

It follows from the foregoing that if $X \sim \mathcal{N}(\mu, \sigma^2)$, then $Z = X - \mu$

$$Z = \frac{X - \mu}{\sigma}$$

is a normal random variable with mean 0 and variance 1. Such a random variable Z is said to have a standard, or unit, normal distribution. Let $\Phi(\cdot)$ denote its distribution function. That is,

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-y^2/2} dy, -\infty < x < \infty$$

This result that $Z = (X - \mu)/\sigma$ has a standard normal distribution when X is normal with parameters μ and σ^2 is quite important, for it enables us to write all probability statements about X in terms of probabilities for Z. For instance, to obtain $P\{X < b\}$, we note that X will be less than b if and only if $(X - \mu)/\sigma$ is less than $(b - \mu)/\sigma$, and so

$$\begin{aligned} P\{X < b\} &= P\left\{ \frac{X - \mu}{\sigma} < \frac{b - \mu}{\sigma} \right\} \\ &= \Phi\left(\frac{b - \mu}{\sigma} \right) \end{aligned}$$

Similarly, for any $a < b$,

$$P\{a < X < b\} = \Phi\left(\frac{b - \mu}{\sigma} \right) - \Phi\left(\frac{b - \mu}{\sigma} \right)$$

In order to compute $\Phi(x)$ we refer to the tables present either in `https://statistics.laerd.com/statistical-guides/img/normal-table-large.png` or in Table A1 of the appendix in [5].

Example 6.4.3. Suppose that a binary message—either "0" or "1"—must be transmitted by wire from location A to location B. However, the data sent over the wire are subject to a channel noise disturbance and so to reduce the possibility of error, the value 2 is sent over the wire when the message is "1" and the value -2 is sent when the message is "0". If $x, x = \pm 2$, is the value sent at location A then R, the

value received at location B, is given by R = x + N , where N is the channel noise disturbance. When the message is received at location B, the receiver decodes it according to the following rule:

if $R \geq .5$, then "1" is concluded

if $R < .5$, then "0" is concluded

Because the channel noise is often normally distributed, we will determine the error probabilities when N is a standard normal random variable. There are two types of errors that can occur: One is that the message "1" can be incorrectly concluded to be "0" and the other that "0" is incorrectly concluded to be "1". The first type of error will occur if the message is "1" and $2 + N < .5$, whereas the second will occur if the message is "0" and $-2 + N \geq .5$. Hence,

$$\begin{aligned} P\{\text{error—message is "1"}\} &= P\{N < -1.5\} \\ &= 1 - \Phi(1.5) = .0668 \end{aligned}$$

and

$$\begin{aligned} P\{\text{error—message is "0"}\} &= P\{N > 2.5\} \\ &= 1 - \Phi(2.5) = .0062 \end{aligned}$$

6.4.5 The Hypergeometric distribution

A bin contains N + M batteries, of which N are of acceptable quality and the other M are defective. A sample of size n is to be randomly chosen (without replacements) in the sense that the set of sampled batteries is equally likely to be any of the $\binom{N+M}{m}$ subsets of size n. If we let X denote the number of acceptable batteries in the sample, then N M :

$$P\{X = i\} = \frac{\binom{N}{i}\binom{M}{n-i}}{\binom{N+M}{n}}, i = 0, 1, ..., min(N, n)$$

Any random variable X whose probability mass function is given by the previous equation is said to be a *hypergeometric* random variable with parameters N , M , n.

$$\begin{aligned} E[X] &= np \\ Var(X) &= np(1-p)\left[1 - \frac{n-1}{N+M-1}\right] \end{aligned}$$

For the interested reader, the demonstration of the previous equations can be checked in Chapter 5.3 in [5].

Example 6.4.4. The components of a 6-component system are to be randomly chosen from a bin of 20 used components. The resulting system will be functional if at least 4 of its 6 components are in working condition. If 15 of the 20 components in the bin are in working condition, what is the probability that the resulting system will be functional?

$$\begin{aligned} P\{Z \geq 4\} &= \sum_{i=4}^{6} P\{X = i\} \\ &= \frac{\binom{15}{4}\binom{5}{2} + \binom{15}{5}\binom{5}{1} + \binom{15}{6}\binom{5}{0}}{\binom{20}{6}} \\ &\approx 0.8687 \end{aligned}$$

Solution If X is the number of working components chosen, then X is hypergeometric with parameters 15, 5, 6. The probability that the system will be functional is

6.4.6 The Uniform distribution

A random variable X is said to be uniformly distributed over the interval $[\alpha, \beta]$ if its probability density function is given by

$$f(x) = \begin{cases} \dfrac{1}{\beta - \alpha} & \text{if } \alpha \leq x \geq \beta \\ 0 & \text{otherwise} \end{cases}$$

Note that the foregoing meets the requirements of being a probability density function since

$$\frac{1}{\beta - \alpha} \int_{\alpha}^{\beta} dx = 1$$

The uniform distribution arises in practice when we suppose a certain random variable is equally likely to be near any value in the interval $[\alpha, \beta]$. The probability that X lies in any subinterval of $[\alpha, \beta]$ is equal to the length of that subinterval divided by the length of the interval $[\alpha, \beta]$. This follows since when [a, b] is a subinterval of $[\alpha, \beta]$

$$\begin{aligned} P\{a < X < b\} &= \frac{1}{\beta - \alpha} \int_{a}^{b} dx \\ &= \frac{b - a}{\beta - \alpha} \end{aligned}$$

The mean of a uniform $[\alpha, \beta]$ random variable and the variance are as follows:

$$E[X] = \frac{\alpha + \beta}{2}$$
$$Var(X) = \frac{(\beta - \alpha)^2}{12}$$

For the interested reader, the demonstration of the previous equations can be checked in Chapter 5.4 in [5].

Example 6.4.5. Buses arrive at a specified stop at 15-minute intervals starting at 7 A.M. That is, they arrive at 7, 7:15, 7:30, 7:45, and so on. If a passenger arrives at the stop at a time that is uniformly distributed between 7 and 7:30, find the probability that he waits:

a) less than 5 minutes for a bus;

b) at least 12 minutes for a bus.

Solution Let X denote the time in minutes past 7 A.M. that the passenger arrives at the stop. Since X is a uniform random variable over the interval (0, 30), it follows that the passenger will have to wait less than 5 minutes if he arrives between 7:10 and 7:15 or between 7:25 and 7:30. Hence, the desired probability for (a) is

$$P\{10 < X < 15\} + P\{25 < X < 30\} = \frac{5}{30} + \frac{5}{30} = \frac{1}{3}$$

Similarly, he would have to wait at least 12 minutes if he arrives between 7 and 7:03 or between 7:15 and 7:18, and so the probability for (b) is

$$P\{0 < X < 3\} + P\{15 < X < 18\} = \frac{3}{30} + \frac{3}{30} = \frac{1}{5}$$

6.4.7 The Exponential distribution

A continuous random variable whose probability density function is given, for some $\lambda > 0$, by

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

is said to be an *exponential* random variable (or, more simply, is said to be exponentially distributed) with parameter λ. The cumulative distribution function F(x) of an exponential random variable is given by

$$\begin{aligned} F(x) &= P\{X \leq x\} \\ &= \int_0^x \lambda e^{-\lambda y} dy \\ &= 1 - e^{-\lambda x}, x \geq 0 \end{aligned}$$

The exponential distribution often arises, in practice, as being the distribution of the amount of time until some specific event occurs. For instance, the amount of time (starting from now) until an earthquake occurs, or until a new war breaks out. We show that λ is the reciprocal of the mean, and the variance is equal to the square of the mean.

$$\begin{aligned} E[X] &= \frac{1}{\lambda} \\ Var(X) &= \frac{1}{\lambda^2} \end{aligned}$$

For the interested reader, the demonstration of the previous equations can be checked in Chapter 5.6 in [5].

6.4.8 Introduction to distributions of sampling statistics

In Section 6.1 we discussed the differences between population and sample. By randomly sampling from the population we believe we can draw conclusions about the population as a whole. One basic assumption needs to be fulfilled in order for this to reasonably work. There is an underlying (population) probability distribution such that the measurable values of the items in the population can be thought of as being independent random variables having this distribution. In other words, if $X_1, ..., X_n$ are independent random variables having a common distribution F, then we say that they constitute a *sample* (sometimes called a *random sample*) from the distribution F.

In same cases we can immediately see which distribution the data follows. In pother cases we need to conduct statistical tests to see if the data follows as a distribution. This is called *parametric* inference or *parametric* statistics. The opposite is *nonparametric* statistics, which really does not care about which distribution the data follows.

We often suppose that the value associated with any member of the population can be regarded as being the value of a random variable having expectation μ and variance σ^2. The quantities μ and σ^2 are called the *population mean* and the *population variance*, respectively. Let $X_1, X_2, ..., X_n$ be a sample of values from this population. The sample mean is defined by

$$\bar{X} = \frac{X_1, X_2, ..., X_n}{n}$$

Its expected value and variance are obtained as follows:

$$\begin{aligned} E[X] &= \lambda \\ Var(X) &= \frac{\sigma^2}{n} \end{aligned}$$

Hence, the expected value of the sample mean is the population mean μ whereas its variance is $1/n$ times the population variance. As a result, we can conclude that X is also centered about the population mean μ, but its spread becomes more and more reduced as the sample size increases. Again, the larger the sample size, the more we reduce the spread of $\bar{X}$, as seen in Figure 6.8.

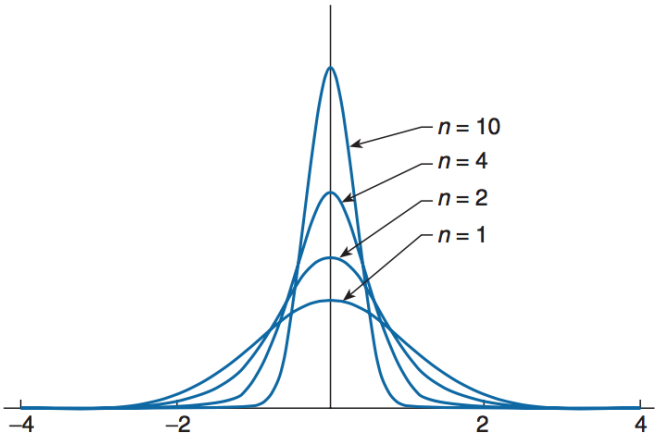


Figure 6.8: Densities of sample means from a standard normal population

6.4.9 The central limit theorem

The *central limit theorem* provides an explanation why so many things in nature, when plotted have the familiar bell-shaped curve that we discuss in previous sections. In addition, the theorem states that very often when we sample random variables from a distribution their sum or mean will move towards a normal distribution, even if the original variables themselves are not normally distributed. This is quite a counterintuitive fact, but a fact it is nevertheless.

Definition 53. *Let $X_1, X_2, ..., X_n$ be a sequence of independent and identically distributed random variables each having mean μ and variance σ^2. Then for n large, the distribution of*

$$X_1 + \cdots + X_n$$

is approximately normal with mean $n\mu$ and variance $n\sigma^2$.

It follows from the central limit theorem that

$$\frac{X_1 + \cdots + X_n - n\mu}{\sigma\sqrt{n}}$$

is approximately a standard normal random variable; thus, for n large,

$$P\left\{\frac{X_1 + \cdots + X_n - n\mu}{\sigma\sqrt{n}} < x\right\} \approx P\{Z < x\}$$

where Z is a standard normal random variable.

Example 6.4.6. An insurance company has 25,000 automobile policy holders. If the yearly claim of a policy holder is a random variable with mean 320 and standard deviation 540, approximate the probability that the total yearly claim exceeds 8.3 million.

Solution Let X denote the total yearly claim. Number the policy holders, and let X_i denote the yearly claim of policy holder i. With n = 25,000, we have from the central limit theorem that $X = \sum_{i=1}^{n} X_i$ will have approximately a normal distribution with mean $320 \times 25000 = 8 \times 10^6$ and standard deviation

$540\sqrt{25000} = 8.5381 \times 10^4$. Therefore,

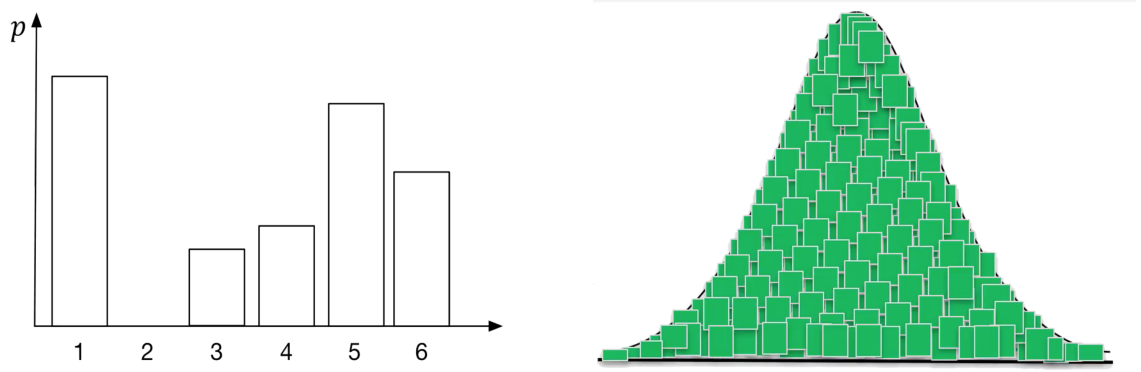
$$
\begin{aligned}
P\{X > 8.3 \times 10^6\} &= P\left\{\frac{X - 8 \times 10^6}{8.5381 \times 10^4} > \frac{8.3 \times 10^6 - 8 \times 10^6}{8.5381 \times 10^4}\right\} \\
&= P\left\{\frac{X - 8 \times 10^6}{8.5381 \times 10^4} > \frac{0.3 \times 10^6}{8.5381 \times 10^4}\right\} \\
&\approx P\{Z > 3.51\} && \text{where Z is a random variable,} \\
& && \text{as seen in Section 6.4.4} \\
&= 1 - \Phi(3.51) \approx 0.00023
\end{aligned}
$$

Thus, there are only 2.3 chances out of 10,000 that the total yearly claim will exceed 8.3 million.

Let's consider the discrete probability distribution depicted in Figure 6.9a It is obvious that it does not look like a bell-shaped curve, as opposed to a normal distribution. However, let's try an experiment drawing samples from our probability distribution. In the first sample, we choose five data points (random numbers from our probability distribution) and calculate their mean. Then we collect samples until we reach an amount of 10000 following the same procedure.

$$
\begin{aligned}
s_1 &= 1, 1, 3, 6, 6 \rightarrow \bar{x} = 3.4 \\
s_2 &= 1, 3, 4, 6, 6 \rightarrow \bar{x} = 4 \\
s_3 &= 1, 1, 1, 6, 6 \rightarrow \bar{x} = 1.8 \\
&\vdots \\
s_{10000} &= 1, 3, 4, 5, 6 \rightarrow \bar{x} = 3.8
\end{aligned}
$$

If we plot the frequency, plotting the sample mean of each sample we will have something like Figure 6.9b. The distribution of the sample statistics will go towards a normal distribution.



(a) Discrete probability distribution

(b) Bell-shaped frequency

Figure 6.9: Plotting frequency experiment

There are a few more properties that we will present now concerning the central limit theorem. First, the *kurtosis* gets lower when the sample size increases. is a measure of the "tailedness" of the probability distribution of a real-valued random variable The kurtosis of any univariate normal distribution is 3. It is common to compare the kurtosis of a distribution to this value. Distributions with kurtosis less than 3 are said to be *platykurtic*. It means the distribution produces fewer and less extreme outliers than does the normal distribution. An example of a platykurtic distribution is the uniform distribution, which does not produce outliers. Distributions with kurtosis greater than 3 are said to be *leptokurtic*. An example of a leptokurtic distribution is the Laplace distribution, which has tails that asymptotically approach zero more slowly than a Gaussian, and therefore produces more outliers than the normal distribution.

Also, the *skewness* decreases when the sample size increases. Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. The skewness value can be positive or negative, or undefined. For a unimodal distribution, negative skew indicates that the tail on the left side of the probability density function is longer or fatter than the right side—it does not distinguish these two kinds of shape. Conversely, positive skew indicates that the tail on the right side is longer or fatter than the left side. In cases where one tail is long but the other tail is fat, skewness does not obey a simple rule. For example, a zero value means that the tails on both sides of the mean balance out overall; this is the case for a symmetric distribution, but is also true for an asymmetric distribution where the asymmetries even out, such as one tail being long but thin, and the other being short but fat.

Finally, the larger the sample size, the tighter is the fit around the mean.

Let us know look at what the central limit theorem give us. Because

$$E[X_i] = p, Var(X_i) = p(1-p)$$

it follows from the central limit theorem that for n large

$$\frac{X - np}{\sqrt{np(1-p)}}$$

will approximately be a standard normal random variable.

Let $X_1, \ldots, X_n$ be a sample from a population having mean μ and variance σ^2. The central limit theorem can be used to approximate the distribution of the sample mean

$$\bar{X} = \sum_{i=1}^{n} X_i/n$$

Since a constant multiple of a normal random variable is also normal, it follows from the central limit theorem that X will be approximately normal when the sample size n is large. Since the sample mean has expected value and standard deviation $\sigma/\sqrt{n}$, it then follows that

$$\frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$$

has approximately a standard normal distribution.

6.4.10 The sample variance

If we have a random sample $X_1, X_2, \ldots, X_n$ from a distribution with mean μ and variance σ^2, the statistics S^2, i.e. the sample variance, is given by:

$$S^2 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{n-1}$$

The sample standard deviation is then the square root of S^2.

6.5 Null hypothesis significance testing

6.5.1 Tests concerning the mean of a normal population

Let us suppose that a random sample from a population distribution, specified except for a vector of unknown parameters, is to be observed. However, rather than wishing to explicitly estimate the unknown parameters, let us now suppose that we are primarily concerned with using the resulting sample to test some particular hypothesis concerning them. As an illustration, suppose a manufacturer claims that the

minimum lifetime of a light bulb is more than 15000 hours. In a sample of 40 light bulbs it was found that they only lasted 14930 hours on average. Assume the populations standard deviations is 240 hours.

$$\mu_0 = 15000$$
$$n = 40$$
$$\bar{x}(\text{or } \mu) = 14930$$
$$\sigma = 240$$

How do we go about testing whether there really is a difference or not? First, we can express our *hypothesis* or our idea:

$$H_0 : \mu \geq \mu_0$$

Here we expressed our *null hypothesis* or *not hypothesis*. A statistical hypothesis is usually a statement about a set of parameters of a population distribution. It is called a hypothesis because it is not known whether or not it is true. Now, as was previously mentioned, the objective of a statistical test of H0 is not to explicitly determine whether or not H0 is true but rather to determine if its validity is consistent with the resultant data. Hence, with this objective it seems reasonable that H_0 should only be rejected if the resultant data are very unlikely when H_0 is true. The classical way of accomplishing this is to specify a value α and then require the test to have the property that whenever H_0 is true its probability of being rejected is never greater than α. The value α, called the *level of significance of the test*, is usually set in advance. In other words, α is the probability of rejecting the null hypothesis given that it is true. In natural sciences standard α used the years is 0.05. That is, there is a 5% chance that we might reject the null hypothesis even if it is true.

Let us define the test statistic z in terms of the sample mean, sample size (n), and the population standard deviation.

$$z = \frac{\mu - \mu_0}{\sigma/\sqrt{n}}$$

Now, if we plug the previously presented values in out formula we have:

$$z = \frac{14930 - 15000}{240/\sqrt{40}} = -1.844662$$

We have what is known as a *one-tailed test* in statistics. That is, we expecting a difference in one direction. What we are looking for is to check if our z statistic is lower or higher than what is called the *critical value*. In the case presented in Figure 6.10 is must lower, since we are looking for values that are to the left of the critical value. It is visualized in blue. Then, how do we get that critical value? Tables of critical values for different distributions may be found online, and in the bottom row of such tables the least values based on α are presented. Then, the sample size usually is larger than 29 and then we can see that from our value of α, the critical value is -1.645. We notice that -1.8 is lower than -1.6, so we "are" in the blue region for this test. Then, we can conclude that with at 0.05 we reject the hypothesis of the mean lifetime of the light time is above 15000 hours.

In the previous example we had a one-tailed test. In short, we expect the population mean to be larger or equal to a value. But, if we say it can larger or lower than a hypothesis mean? In that particular case (a *two-tailed test*), we have no clue if it should be larger or lower than a hypothesis value. Let's use the same α value from the previous example in the next one, depicted in Figure 6.11. Now, the areas are distributed to both sides, and since there is a smaller area at both sides the critical values also move. If $\alpha = 0.05$, $\alpha/2 = 0.025$, and therefore in this particular case the hypothesis is accepted with values of z that re between ± 1.96. Here, our null hypothesis is that the true population mean and the hypothesis population population mean are equal. We can also always formulate an alternative hypothesis, in our case μ_1.

$$H_0 : \mu = \mu_0$$
$$H_1 : \mu \neq \mu_0$$

Let's say that we have conducted an experiment where we want to see if there is a *statically significant difference* (SSD) when introducing a faster and faster test technique for finding faults compared with old

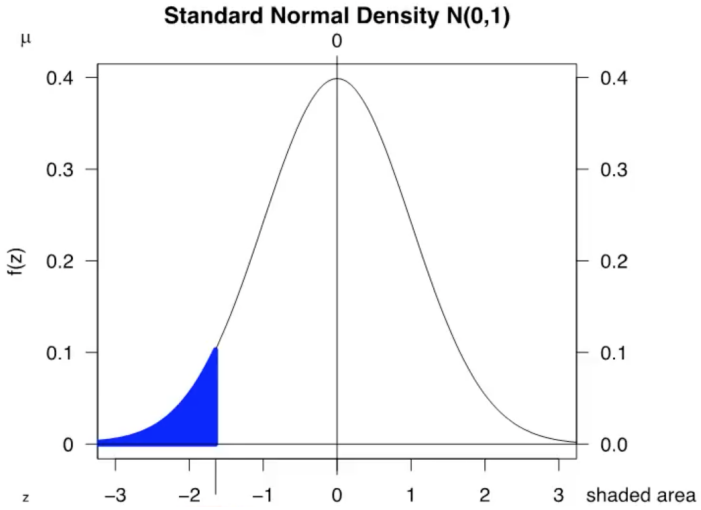


Figure 6.10: One-tailed example

way of doing things. So, we let 30 engineers try the new test technique and then we compare the results with our hypothesis value. The test engineris found on average 14.6 faults using the new technique. Interestingly, the old way of doing things provided a mean of 15.4 with a σ of 2.5. Is important to remark that we have no idea if the new test technique is better or worse, so we presume a test goes in both directions. That is, it can be better or worse.

$$\bar{x} = 14.6$$
$$\mu_0 = 15.4$$
$$\sigma = 2.5$$
$$n = 30$$

We first calculate the z statistic

$$\frac{\mu - \mu_0}{\sigma/\sqrt{n}} \approx -1.75$$

In the case of a case of known variance concerning the mean of a normal population, the acceptance criteria is:

$$\text{reject } H_0 \text{ if } \frac{\sqrt{n}}{\sigma}|\bar{X} - \mu_0| > z_{\alpha/2} = 1.96$$
$$\text{accept } H_0 \text{ if } \frac{\sqrt{n}}{\sigma}|\bar{X} - \mu_0| \leq z_{\alpha/2} = 1.96$$

The value is not in the blue region of Figure 6.11, since -1.75 is to the right of -1.96. Hence, we cannot reject the null hypothesis. In other words, we see no statistically significant difference between the new and old test techniques on the 0.05 significance level. But, let us for a short while not think about the null hypothesis, critical values, z statistics, significance levels, etc. What if both techniques are equal in every way concerning their capabilities of finding faults or of finding certain types of faults and the only thing that differs is that engineers use a new technique they are more efficient. In other words, they do more in a given time unit compared to using the old test technique.

Up to now we have supposed that the only unknown parameter of the normal population distribution is its mean. However, the more common situation is one where the mean μ and variance σ^2 are both unknown. Let us suppose this to be the case and again consider a test of the hypothesis that the mean is equal to some specified value μ_0. That is, consider a test of

$$H_0 : \mu = \mu_0$$

versus the alternative

$$H_1 : \mu \neq \mu_0$$

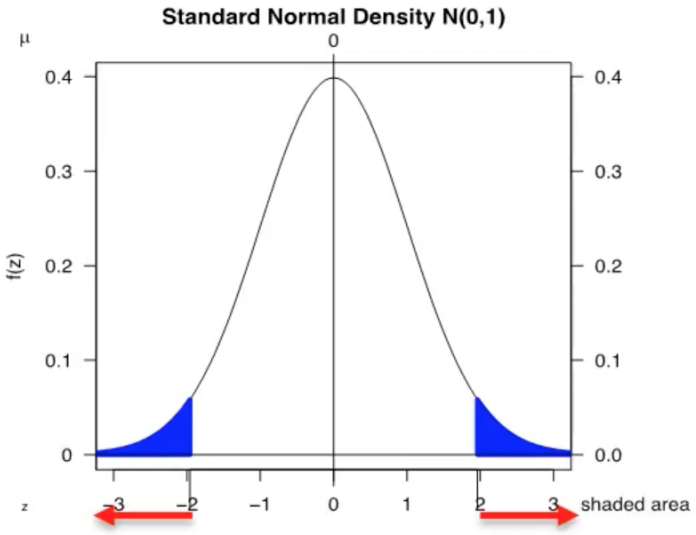


Figure 6.11: Two-tailed example

It should be noted that the null hypothesis is not a simple hypothesis since it does not specify the value of σ^2. As before, it seems reasonable to reject H_0 when the sample mean X is far from μ_0. However, how far away it need be to justify rejection will depend on the variance σ^2. Now when σ^2 is no longer known, it seems reasonable to estimate it by

$$S^2 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{n-1}$$

and then to reject H_0 when

$$\left| \frac{\bar{X} - \mu_0}{S/\sqrt{n}} \right| \gg 0$$

To determine how large a value of the statistic

$$T = \frac{\sqrt{n}(\bar{X} - \mu_0)}{S}$$

to require for rejection, in order that the resulting test have significance level α, we must determine the probability distribution of this statistic when H_0 is true.

6.5.2 Testing the equality of means of two normal populations

A common situation faced by a practicing engineer is one in which she must decide whether two different approaches lead to the same solution. Often such a situation can be modeled as a test of the hypothesis that two normal populations have the same mean value.

In the case variances are known, suppose that $X_1, \ldots, X_{n_A}$ and $Y_1, \ldots, Y_{n_B}$ are independent samples from normal populations having unknown means μ_x and μ_y but known variances σ_x^2 and σ_y^2. Let us consider the problem of testing the hypothesis

$$H_0 : \mu_x = \mu_y$$

versus the alternative

$$H_1 : \mu_x \neq \mu_y$$

Since X is an estimate of μ_x and Y of μ_y, it follows that X Y can be used to estimate $\mu_x - \mu_y$. Hence, because the null hypothesis can be written as $H_0 : \mu_x - \mu_y = 0$, it seems reasonable to reject it when X

- Y is far from zero. We obtain that the significance level α test is

$$\text{accept } H_0 \text{ if } \frac{|\bar{X} - \bar{Y}|}{\sqrt{\sigma_x^2/n + \sigma_y^2/m}} \leq z_{\alpha/2}$$
$$\text{reject } H_0 \text{ if } \frac{|\bar{X} - \bar{Y}|}{\sqrt{\sigma_x^2/n + \sigma_y^2/m}} > z_{\alpha/2}$$

A test of the hypothesis $H_0 : \mu_x = \mu_y (or H_0 : \mu_x \leq \mu_y)$ against the one-sided alternative $H_1 : \mu_x > \mu_y$ would be to

$$\text{accept } H_0 \text{ if } |\bar{X} - \bar{Y}| \leq z_\alpha \sqrt{\frac{\sigma_x^2}{n_A} + \frac{\sigma_y^2}{n_B}}$$
$$\text{reject } H_0 \text{ if } |\bar{X} - \bar{Y}| > z_\alpha \sqrt{\frac{\sigma_x^2}{n_A} + \frac{\sigma_y^2}{n_B}}$$

Suppose again that $X_1, \ldots, X_{n_A}$ and $Y_1, \ldots, Y_{n_B}$ are independent samples from normal populations having respective parameters (μ_x, σ_x^2) and (μ_y, σ_y^2), but now suppose that all four parameters are unknown. We will once again consider a test of

$$H_0 : \mu_x = \mu_y$$

versus the alternative

$$H_1 : \mu_x \neq \mu_y$$

To determine a significance level α test of H_0 we will need to make the additional assumption that the unknown variances σ_x^2 and σ_y^2 are equal. Let σ^2 denote their value, that is,

$$\sigma^2 = \sigma_x^2 = \sigma_y^2$$

As before, we would like to reject H_0 when$\bar{X} - \bar{Y}$is "far" from zero. To determine how far from zero it need be, let

$$S_x^2 = \frac{\sum_{i=1}^{n_A} (X_i - \bar{X})^2}{n_A - 1}$$
$$S_y^2 = \frac{\sum_{i=1}^{n_B} (X_i - \bar{Y})^2}{n_B - 1}$$

Then we conclude with S_p^2, the pooled estimator of the common variance σ^2, is given by

$$S_p^2 = \frac{(n_A - 1)S_x^2 + (n_B - 1)S_y^2}{n_A + n_B - 2}$$

Hence, when H_0 is true, and so $\mu_x - \mu_y = 0$, the statistic

$$T \equiv \frac{\bar{X} - \bar{Y}}{\sqrt{S_p^2(1/n_A + 1/n_B))}}$$

From this, it follows that we can test the hypothesis that $\mu_x = \mu_y$ as follows:

$$\text{accept } H_0 \text{ if } |T| \leq t_{\alpha/2, n_A + n_B - 2}$$
$$\text{reject } H_0 \text{ if } |T| > t_{\alpha/2, n_A + n_B - 2}$$

If we are interested in testing the one-sided hypothesis

$$H_0 : \mu_x \leq \mu_y$$

versus the alternative

$$H_1 : \mu_x > \mu_y$$

then H_0 will be rejected at large values of T . Thus the significance level α test is to

$$\text{reject } H_0 \text{ if } T \geq t_{\alpha, n_A+n_B-2}$$
$$\text{not reject } H_0 \text{ otherwise}$$

Example 6.5.1. Twenty-two volunteers at a cold research institute caught a cold after having been exposed to various cold viruses. A random selection of 10 of these volunteers was given tablets containing 1 gram of vitamin C. These tablets were taken four times a day. The control group consisting of the other 12 volunteers was given placebo tablets that looked and tasted exactly the same as the vitamin C tablets. This was continued for each volunteer until a doctor, who did not know if the volunteer was receiving the vitamin C or the placebo tablets, decided that the volunteer was no longer suffering from the cold. The length of time the cold lasted was then recorded. At the end of this experiment, the following data resulted.

Treated with Vitamin C	Treated with Placebo
5.5	6.5
6.0	6.0
7.0	8.5
6.0	7.0
7.5	6.5
6.0	8.0
7.5	7.5
5.5	6.5
7.0	7.5
6.5	6.0
	8.5
	7.0

Figure 6.12: Two normal populations example

Do the data listed prove that taking 4 grams daily of vitamin C reduces the mean length of time a cold lasts? At what level of significance?

Solution To prove the above hypothesis, we would need to reject the null hypothesis in a test of

$$H_0 : \mu_p \leq \mu_c$$

versus the alternative

$$H_1 : \mu_p > \mu_c$$

where μ_c is the mean time a cold lasts when the vitamin C tablets are taken and μ_p is the mean time when the placebo is taken. Assuming that the variance of the length of the cold is the same for the vitamin C patients and the placebo patients, we test the above. H_0 would be rejected at the 5 percent level of significance. We can compute then

$$\bar{X} = 6.450, \bar{Y} = 7.125$$
$$S_x^2 = 0.581, S_y^2 = 0.778$$

Therefore,

$$S_p^2 = \frac{9}{20} S_x^2 + \frac{11}{20} S_y^2 = 0.689$$

and the value of the test statistic is

$$TS = \frac{-0.675}{\sqrt{0.689(1/10 + 1/12)}} = -1.90$$

Since $t_{0.5,20} = 1.725$, the null hypothesis is rejected at the 5 percent level of significance. That is, at the 5 percent level of significance the evidence is significant in establishing that vitamin C reduces the mean time that a cold persists.

For the sake of completeness, we can see the example when the variance is known. In this case, we do not need to have a pool estimate of the common variance, but we instead use the known variance.

$$T = \frac{\bar{X} - \bar{Y}}{\sqrt{\frac{\sigma_x^2}{n_A} + \frac{\sigma_y^2}{n_B}}}$$

We went through examples where the variance is either known or unknown. If we did not know the variance we calculate a pooled estimate of the variance which we could use then to calculate our statistic. There are cases where we cannot assume that the variances are equal. Before we talked about cases when the variance was known or not, but knowing if the variance is equal or not is also important. Just as any other test we discussed so far one can either compare two variances, that is, from two samples or compare one variance that is from one sample with a reference value (i.e. some specified value we have) typically referred as σ_0. In order for us to determine if the variance is equal or not we need two types of random distributions, the $\mathcal{X}^2$ and the F distribution; depending on if we compare against a reference value or if we compare two samples.

When comparing against a reference value, we might use the the F distribution, where the F statistic is given by:

$$T = \frac{(n-1)}{(S/\sigma_0)^2}$$

with degrees of freedom n-1.

On the other hand, when comparing two sample variances, we might use the the $\mathcal{X}^2$ distribution, where the T statistic is given by:

$$F = \frac{S_A^2}{S_B^2}$$

with $n_A - 1$ and $n_B - 1$ degrees of freedom.

We will illustrate the second case with an example, where we will compare two sample variances. Let us assume we have two sample sizes:

$$n_A = 6, n_B = 11$$

Let us also assume the sample variances:

$$S_A^2 = 65.1, S_B^2 = 61.9$$

Do the two samples have equal variance? If we divide the two variances we will have the statistic F:

$$F = \frac{S_A^2}{S_B^2} = \frac{65.1}{61.9} = 1.0517$$

If we look at the table of the F distribution we know that the numerator is $6-1 = 5$ and the denominator is $11 - 1 = 10$. Then, we can see what the critical values are for $\alpha = 0.05$. We now know:

$$F_{0.975,5,10} = 4.236086$$
$$F_{0.0025,5,10} = 0.0545806$$

We now have what is needed to make a decision. Our F statistic is not to the left of our leftmost region nor is it to the right of our rightmost region; or, in other words, the F statistic is not lower than the low boundary nor it is higher than our high boundary. To summarize, we cannot reject the null hypothesis that the two sample variances are equal using a α of 0.05. In other words, the variances are equal or there is not significant difference between the two variances when we compare them.

6.5.3 Nonparametric hypothesis tests

For all the examples and tests we have talked about we made several assumptions. The most important assumption was that the tests assumed a certain type of distribution—very often a normal one. However, in reality we often see that the data we have cannot be assumed to follow a normal distribution or any other known distribution for that matter. In those cases we cannot rely on parametric tests, we need to look at *nonparametric* ones. Let's review some of the advantages of using nonparametric methods.

1. They make fewer untested assumptions. There is rarely any substantial evidence that the data follows a certain distribution, normal or otherwise.
2. Nonparametric methods can be used for classification. That is, using data types that are nominal or ordinal. Generally speaking parametric tests cannot.
3. Nonparametric methods are usually easier to understand and use.
4. Nonparametric tests are better to use when dealing with small sample sizes.

Nonparametric methods, however, do come with some disadvantages which, according to some, parametric methods do not have.

1. There are more parametric methods that we can use for analyzing more source of experimental results. In addition, many problems dealing with the estimation of population parameters from a sample of observations have only being dealt with bi-parametric methods.
2. Some claim that nonparametric tests are not as powerful as parametric tests. When we say that a test has a high statistically power we mean that it is more sensitive to finding a real difference. This statement is debatable. First of all, if all assumptions for a parametric test are true, then yes, parametric methods can be more powerful. However, we rarely know if this is true. In the cases when they are not true, nonparametric tests are the better choice. Even if all assumptions do hold, nonparametric tests are often only slightly less powerful.

We will now illustrate nonparametric tests with a two-sample problem.

Example 6.5.2. From `http://sphweb.bumc.bu.edu/otlt/mph-modules/bs/bs704_nonparametric/BS704_Nonparametric4.html`

Consider a Phase II clinical trial designed to investigate the effectiveness of a new drug to reduce symptoms of asthma in children. A total of n=10 participants are randomized to receive either the new drug or a placebo. Participants are asked to record the number of episodes of shortness of breath over a 1 week period following receipt of the assigned treatment. The data are shown below.

Placebo	7	5	6	4	12
New Drug	3	6	4	2	1

Is there a difference in the number of episodes of shortness of breath over a 1 week period in participants receiving the new drug as compared to those receiving the placebo? By inspection, it appears that participants receiving the placebo have more episodes of shortness of breath, but is this statistically significant?

In addition, the sample size is small ($n_1 = n_2 = 5$), so a nonparametric test is appropriate. The hypothesis is given below, and we run the test at the 5% level of significance (i.e., $\alpha = 0.05$).

$$H_0 : \mu_1 = \mu_2$$

versus the alternative

$$H_1 : \mu_1 \neq \mu_2$$

Note that if the null hypothesis is true (i.e., the two populations are equal), we expect to see similar numbers of episodes of shortness of breath in each of the two treatment groups, and we would expect to

see some participants reporting few episodes and some reporting more episodes in each group. This does not appear to be the case with the observed data. A test of hypothesis is needed to determine whether the observed data is evidence of a statistically significant difference in populations.

The first step is to assign ranks and to do so we order the data from smallest to largest. This is done on the combined or total sample (i.e., pooling the data from the two treatment groups (n=10)), and assigning ranks from 1 to 10, as follows. We also need to keep track of the group assignments in the total sample.

		Total Sample (Ordered Smallest to Largest)		Ranks	
Placebo	New Drug	Placebo	New Drug	Placebo	New Drug
7	3		1		1
5	6		2		2
6	4		3		3
4	2	4	4	4.5	4.5
12	1	5		6	
		6	6	7.5	7.5
		7		9	
		12		10	

Note that the lower ranks (e.g., 1, 2 and 3) are assigned to responses in the new drug group while the higher ranks (e.g., 9, 10) are assigned to responses in the placebo group. Again, the goal of the test is to determine whether the observed data support a difference in the populations of responses. Recall that in parametric tests, when comparing means between two groups, we analyzed the difference in the sample means relative to their variability and summarized the sample information in a test statistic. A similar approach is employed here. Specifically, we produce a test statistic based on the ranks.

First, we sum the ranks in each group. In the placebo group, the sum of the ranks is 37; in the new drug group, the sum of the ranks is 18.

For the test, we call the placebo group 1 and the new drug group 2. We let R_1 denote the sum of the ranks in group 1 (i.e., $R_1 = 37$), and R_2 denote the sum of the ranks in group 2 (i.e., $R_2 = 18$). If the null hypothesis is true (i.e., if the two populations are equal), we expect R_1 and R_2 to be similar. In this example, the lower values (lower ranks) are clustered in the new drug group (group 2), while the higher values (higher ranks) are clustered in the placebo group (group 1). This is suggestive, but is the observed difference in the sums of the ranks simply due to chance? To answer this we will compute a test statistic to summarize the sample information and look up the corresponding value in a probability distribution.

The test statistic for the Mann Whitney U Test is denoted U and is the smaller of U_1 and U_2, defined below.

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1 = 5(5) + \frac{5(6)}{2} - 37 = 3$$
$$U_2 = n_1 n_2 + \frac{n_2(n_2+1)}{2} - R_2 = 5(5) + \frac{5(6)}{2} - 18 = 22$$

In every test, we must determine whether the observed U supports the null or research hypothesis. This is done following the same approach used in parametric testing. Specifically, we determine a critical value of U such that if the observed value of U is less than or equal to the critical value, we reject H_0 in favor of H_1 and if the observed value of U exceeds the critical value we do not reject H_0.

The critical value of U can be found in tables online. To determine the appropriate critical value we need sample sizes (in our example: $n_1 = n_2 = 5$) and our two-sided level of significance ($\alpha = 0.05$).

For this example the critical value is 2, and the decision rule is to reject H_0 if $U < 2$. We do not reject H_0 because $3 > 2$. We do not have statistically significant evidence at $\alpha = 0.05$, to show that the two populations of numbers of episodes of shortness of breath are not equal. However, in this example, the failure to reach statistical significance may be due to low power. The sample data suggest a difference, but the sample sizes are too small to conclude that there is a statistically significant difference.

6.5.4 Goodness of fit tests

We are often interested in determining whether or not a particular probabilistic model is appropriate for a given random phenomenon. This determination often reduces to testing whether a given random sample comes from some specified, or partially specified, probability distribution. *Goodness of fit tests* are statistical tests that examine or pass judgment on if data is following a certain kind of distribution. In the end, we might a priori believe that data should be normally distributed, but using a goodness of fit provides a probability that this is the case. The classical approach to obtaining a goodness of fit test of a null hypothesis that a sample has a specified probability distribution is to partition the possible values of the random variables into a finite number of regions. The numbers of the sample values that fall within each region are then determined and compared with the theoretical expected numbers under the specified probability distribution, and when they are significantly different the null hypothesis is rejected.

We will consider the case of a goodness of fit test when all parameters are specified. To test the foregoing hypothesis, let $X_i, i = 1, ..., k$, denote the number of the Y_j's—independent random variables—that equal i. Then as each Y_j will independently equal i with probability $P\{Y = i\}$, it follows that, under H_0, X_i is binomial with parameters n and p_i. Hence, when H_0 is true,

$$E[X_i] = np_i$$

When this is large it is an indication that H_0 is not correct. Indeed such reasoning leads us to consider the following test statistic:

$$T = \sum_{i=1}^{k} \frac{(X_i - np_i)^2}{np_i} = \sum_{i} \frac{X_i^2}{np_i} - n$$

and to reject the null hypothesis when T is large. Finally, the approximate α level test is:

$$\text{reject } H_0 \text{ if } T \geq \mathcal{X}_{\alpha,k-1}$$
$$\text{not reject } H_0 \text{ otherwise}$$

Example 6.5.3. In recent years, a correlation between mental and physical well-being has increasingly become accepted. An analysis of birthdays and death days of famous people could be used as further evidence in the study of this correlation. To use these data, we are supposing that being able to look forward to something betters a persons mental state; and that a famous person would probably look forward to his or her birth- day because of the resulting attention, affection, and so on. If a famous person is in poor health and dying, then perhaps anticipating his birthday would "cheer him up and therefore improve his health and possibly decrease the chance that he will die shortly before his birthday". The data might therefore reveal that a famous person is less likely to die in the months before his or her birthday and more likely to die in the months afterward.

	6 Months Before	5 Months Before	4 Months Before	3 Months Before	2 Months Before	1 Month Before	The Months	1 Month After	2 Months After	3 Months After	4 Months After	5 Months After
Number of deaths	90	100	87	96	101	86	119	118	121	114	113	106

For instance, let us determine what the result is if we code the data into 4 possible outcomes as follows:

$$\begin{aligned}\text{outcome } 1 &= -6, -5, -4\\ \text{outcome } 2 &= -3, -2, -1\\ \text{outcome } 3 &= 0, 1, 2\\ \text{outcome } 4 &= 3, 4, 5\end{aligned}$$

That is, for instance, an individual whose death day occurred 3 months before his or her birthday would be placed in outcome 2. With this classification, the data would be as follows:

Outcome	Number of Times Occurring
1	277
2	283
3	358
4	333

$n = 1{,}251$
$n/4 = 312.75$

Solution The test statistic for testing $H_0 = p_i = 1/4, i = 1, 2, 3, 4$ (so $np_i = 1251/4 = 312.75$) is

$$T = \frac{(277)^2 + (283)^2 + (358)^2 + (333)^2}{312.75} - 1251 = 14.775$$

Hence, as $\mathcal{X}^2_{.01,3} = 11.345$ (check any table online), the null hypothesis would be rejected even at the 1 percent level.

6.6 Regression

Many engineering and scientific problems are concerned with determining a relationship between a set of variables. Knowledge of such a relationship would enable us to predict the output for various values. In many situations, there is a single *response* variable Y, also called the *dependent* variable, which depends on the value of a set of *input*, also called *independent*, variables $x_1, \ldots, x_r$. The simplest type of relationship between the dependent variable Y and the input variables $x_1, \ldots, x_r$ is a linear relationship. A simple linear regression model supposes a linear relationship between the mean response and the value of a single independent variable. It can be expressed as

$$Y = \alpha + \beta x + e$$

where x is the value of the independent variable, also called the input level, Y is the response, and e, representing the random error, is a random variable having mean 0. In other words, the aim of *linear regression* is to model a continuous variable Y as a mathematical foundation of one or more x variables. The idea is that we will be able to forecast the Y when only the x is known.

In the previous equation, α is called the *intercept* and β the *slope*. A visual definition of those terms is presented in Figure 6.13.

Suppose that the responses Y_i corresponding to the input values $x_i, i = 1, \ldots, n$ are to be observed and used to estimate α and β in a simple linear regression model. To determine estimators of α and β we reason as follows: If $\hat{\alpha}$ is the estimator of α and $\hat{\beta}$ of β, then the estimator of the response corresponding to the input variable x_i would be $\hat{\alpha} + \hat{\beta}x_i$. In order for us to build a statistical model of the data (in

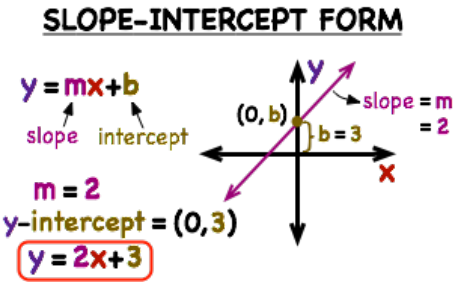


Figure 6.13: Figure found at `http://virtualnerd.com/worksheetHelper.php?tutID=Alg1_10_2_8`

this case a linear model) we need to find the least squares estimators of α and β for a datasets. The estimator of Y can be expressed as

$$\hat{Y} = \alpha + \beta x$$

We can also define

$$\bar{Y} = \sum_i Y_i/n, \ \bar{X} = \sum_i X_i/n,$$

Definition 54. *The least squares estimators of α and β corresponding to the data set $x_i, Y_i, i = 1, \ldots, n$ are, respectively,*

$$\hat{\beta} = \frac{\sum_{i=1}^n x_i Y_i - \bar{x}\sum_{i=1}^n Y_i}{\sum_{i=1}^n x_i^2 - n\bar{x}^2}$$
$$\hat{\alpha} = \bar{Y} - B\bar{x}$$

The straight line $A + Bx$ is called the estimated regression line.

We can now focus on how large errors our model makes when it predicts things. On top of this, we need the standard error in order to devise our hypothesis test under regression model. In order to do so, we can express a new notation to simplify the calculation of the least squares estimators:

$$S_x = \sum x_i, \qquad S_y = \sum y_i$$
$$S_{xx} = \sum x_i^2, \qquad S_{yy} = \sum y_i^2$$
$$S_{xy} = \sum x_i y_i, \qquad n = \text{sample size}$$

The equation used to calculate the estimators of β and α can be then redefined as

$$\hat{\beta} = \frac{nS_{xy} - S_x S_y}{nS_{xx} - S_x^2}$$
$$\hat{\alpha} = \frac{1}{n}S_y - \hat{\beta}\frac{1}{n}S_x$$

Then, in order to calculate the uncertainty (*standard error* SE) of the regression we use the following formula

$$SE_{regr} = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{n-2}}$$

The reason of why we subtract two to the sample size in the equation is that we make two assumptions:

1. The sample is representative of the population;
2. $\bar{Y}$ is representative of Y.

We might also calculate the SE for the coefficients α and β

$$SE_{\hat{\alpha}} = SE_{regr}\sqrt{\frac{\sum x_i^2}{n\sum(x_i-\bar{x})^2}}$$
$$SE_{\hat{\beta}} = \frac{SE_{regr}}{\sqrt{\sum(x_i-\bar{x})^2}}$$

Another statistic that can be calculated in order to understand the uncertainty of our model is the t statistic. For our coefficients, the t statistic is defined by

$$t_{\hat{\alpha}} = \frac{\hat{\alpha}}{SE_{\hat{\alpha}}},\ t_{\hat{\beta}} = \frac{\hat{\beta}}{SE_{\hat{\beta}}}$$

A larger value of the t statistic tell us that is less likely that the coefficient is not equal to 0 purely by chance. So, the higher the t value, the better.

From information theory, we know that the actual information in data is the variation that contains. But R^2 measures is the amount of variation in the dependent variable, in our case Y. Here we use the error sum of squares (SSE) and the total sum of squares to calculate R^2.

$$R^2 = 1 - \frac{SSE}{SST}$$

The closest to 1, the better. But, generally speaking, a value above .7 is considered acceptable. SSE and SST can be calculated with the following formulas:

$$SSE = \sum_1^n (y_i - \hat{y}_i)^2$$
$$SST = \sum_1^n (y_i - \bar{y}_i)^2$$

We can use these new terms to determine how much variation is explained by the regression line. If the points are perfectly linear, then the SSE is 0. However, the coefficient that is normally used is the adjusted R^2, R^2_{adj}. The reason is that it is much better with dealing with more variables. The adjusted R^2_{adj} penalizes total value for the number of predictors (or observations), that is, n. So it is better practice to use adjusted R since adding extra variables always increases your explanatory power. Adjusted R is expressed as follows

$$R^2_{adj} = 1 - \frac{MSE}{MST}$$

The terms mean square error (MSE) and the mean square total (MST) are defined by

$$MSE = \frac{SSE}{n-q}$$
$$MST = \frac{SST}{n-1}$$

being q the number of independent variables (or coefficients in our model).

The F-test can also be used for regression. For instance, say you add a number of variables into a regression model and you want to see if, as a group, they are significant in explaining variation in your dependent variable Y. The F-test tells you whether a group of variables, or even an entire model, is jointly significant. The F statistic can be calculated as follows

$$F = \frac{MSR}{MSE}$$

being MSR the mean square regression

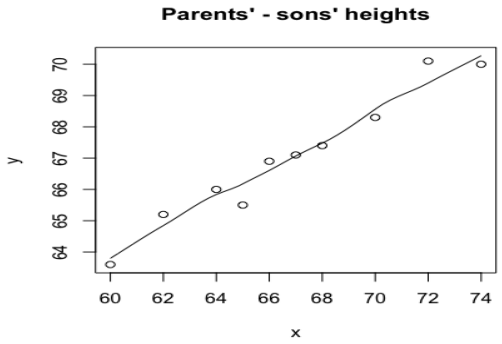
$$MSR = \frac{SST - SSE}{q - 1}$$

In the following we illustrate the terms introduced in this topic with an example.

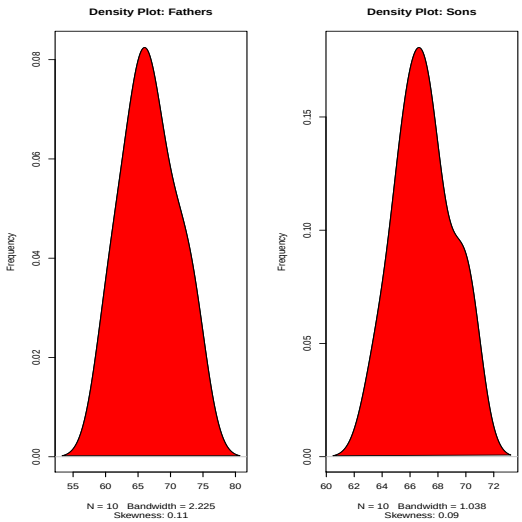
Example 6.6.1. In the following table the heights (in inches) of 10 randomly chosen sons versus that of their fathers are represented.

Fathers' height	60	62	64	65	66	67	68	70	72	74
Sons' height	63.6	65.2	66	65.5	66.9	67.1	67.4	68.3	70.1	70

For the first part of the exercise we represent the data in three different ways using R, all of them represented in Figure 6.14. The scatter diagram of Figure 6.14a suggests that there is a linearly, increasing relationship between father's and son's heights. It is additive and linear. Figure 6.14b represents the density plot for fathers' and sons' heights, and both look approximately normally distributed.



(a) Scatter diagram



(b) Density plot

Figure 6.14: Representing the data

Now we proceed by building the linear model of our data, starting by calculating $\hat{\alpha}$ and $\hat{\beta}$. The table in Figure 6.15 includes data which was calculated step by step (i.e., each column). The table is represented in the following.

	Fathers' height (x)	Sons' height (y)	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})^2$	$(y_i - \bar{y})^2$	$(x_i - \bar{x})(y_i - \bar{y})$	$\hat{y}_i$	$(y_i - \hat{y}_i)$	$(y_i - \hat{y}_i)^2$
	60	63.6	-6.8	-3.41	46.24	11.6281	23.188	63.85093	-0.250932401	6.296707e-02
	62	65.2	-4.8	-1.81	23.04	3.2761	8.688	64.78007	0.419930070	1.763413e-01
	65	66	-2.8	-1.01	7.84	1.0201	2.828	65.70921	0.290792541	8.456030e-02
	64	65.5	-1.8	-1.51	3.24	2.2801	2.718	66.17378	-0.673776224	4.539744e-01
	66	66.9	-0.8	-0.11	0.64	0.0121	0.088	66.63834	0.261655012	6.846335e-02
	67	67.1	0.2	0.09	0.04	0.0081	0.018	67.10291	-0.002913753	8.489956e-06
	68	67.4	1.2	0.39	1.44	0.1521	0.468	67.56748	-0.167482517	2.805039e-02
	70	68.2	3.2	1.29	10.24	1.6641	4.128	68.49662	-0.196620047	3.865944e-02
	74	70.1	5.2	3.09	27.04	9.5481	16.068	69.42576	0.674242424	4.546028e-01
	72	70	7.2	2.99	51.84	8.9401	21.528	70.35490	-0.354895105	1.259505e-01
Sum	**668**	**670.1**			**171.6**	**38.529**	**79.72**			**1.493578**
Mean	**66.8**	**67.01**								

Figure 6.15: Step by step calculations for the linear regression exercise.

Now, we can easily calculate $\hat{\beta}$ using the data contained in the table, since we calculated already the sum of $(x_i - \bar{x})^2$ and $(x_i - \bar{x})(y_i - \bar{y})$.

$$\begin{aligned}\hat{\beta} &= \frac{\sum_{i=1}^{n} x_i Y_i - \bar{x}\sum_{i=1}^{n} Y_i}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2} \\ &= \frac{79.72}{171.6} = 0.4645688\end{aligned}$$

We can also easily calculate $\hat{\alpha}$ knowing the value of $\hat{\beta}$ and the means of Y and x

$$\begin{aligned}\hat{\alpha} &= \bar{Y} - \hat{\beta}\bar{x} \\ &= 67.01 - \beta * 66.8 = 35.97681\end{aligned}$$

Once we have the values of $\hat{\beta}$ and $\hat{\alpha}$, the estimation of Y in our model can be obtained

$$\hat{Y} = 35.9768 + 0.4646x$$

The results of the estimation are calculated and shown in the table of Figure 6.15 under the column labeled with $\hat{y}_i$.

For the sake of completeness, we will calculate $\hat{\beta}$ and $\hat{\alpha}$ with a more efficient procedure, as introduced previously in this section.

$$\begin{aligned} S_x &= \sum x_i = 668, & S_y &= \sum y_i = 670.1 \\ S_{xx} &= \sum x_i^2 = 44794, & S_{yy} &= \sum y_i^2 = 44941.93 \\ S_{xy} &= \sum x_i y_i = 44842.4, & n &= \text{sample size} = 10 \end{aligned}$$

And then

$$\begin{aligned}\hat{\beta} &= \frac{nS_{xy} - S_x S_y}{nS_{xx} - S_x^2} = 0.4645688 \\ \hat{\alpha} &= \frac{1}{n}S_y - \hat{\beta}\frac{1}{n}S_x = 35.97681\end{aligned}$$

Let's proceed by checking whether our coefficients are statistically significant. To do so, we calculate the standard errors.

$$\begin{aligned}SE_{regr} &= \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{n-2}} \\ &= \sqrt{\frac{1.493578}{10-2}} = 0.4320848\end{aligned}$$

We will now calculate the standard error for the coefficients α and β

$$\begin{aligned} SE_{\hat{\alpha}} &= SE_{regr}\sqrt{\frac{\sum x_i^2}{n\sum(x_i-\bar{x})^2}} \\ &= 2.207599 \\ SE_{\hat{\beta}} &= \frac{SE_{regr}}{\sqrt{\sum(x_i-\bar{x})^2}} \\ &= 0.03298453 \end{aligned}$$

The last calculations allows us to obtain the t-value for both coefficients

$$\begin{aligned} t_{\hat{\alpha}} &= \frac{\hat{\alpha}}{SE_{\hat{\alpha}}} = 16.2968 \\ t_{\hat{\beta}} &= \frac{\hat{\beta}}{SE_{\hat{\beta}}} = 14.08444 \end{aligned}$$

In our case, the critical values are $t_{.025,8} = \pm 2.306$, so we can clearly see that both α and β are statistically significant (both are inside of what we called in previous sections the "blue region"). In this part of the example we performed a t-test and evaluated whether α and β are significant.

As explained before, there are other ways to measure the quality of the model, that is, the goodness of fit. Even though we discussed that is better to calculate R^2_{adj} rather than R^2, we will do it for the sake of completeness.

$$\begin{aligned} SSE &= \sum_1^n (y_i - \hat{y}_i)^2 = 1.493578 \\ SST &= \sum_1^n (y_i - \bar{y}_i)^2 = 38.529 \\ R^2 &= 1 - \frac{SSE}{SST} = 0.961235 \end{aligned}$$

The value of R^2 is very high (the closest to one, the better), but we must also check the adjusted value of R^2. The reason is that it is much better equipped to deal with more variables. In the following equation, q represents the number of coefficients in our model—in our case, two.

$$\begin{aligned} MSE &= \frac{SSE}{n-q} = 0.1866973 \\ MST &= \frac{SST}{n-1} = 4.281 \\ R^2_{adj} &= 1 - \frac{MSE}{MST} = 0.9563893 \end{aligned}$$

With only two coefficients there is no much difference between R^2_{adj} and R^2, and both are quit high values. It means that our model is quite powerful in terms of explanation.

Finally, let's check the quality of the model by calculating the F-statistic.

$$\begin{aligned} MSR &= \frac{SST - SSE}{q-1} = 37.03542 \\ F &= \frac{MSR}{MSE} = 198.3715 \end{aligned}$$

If we look at the F table for one and 8 degrees of freedom and an alpha of .05 we obtain a critical value of 5.32. It is obvious that our value of F is significantly higher; so we have a significant result, that is, the model as a whole has a significant fit in a way.

6.6.1 The one-way analysis of variance (ANOVA)

The one-way analysis of variance (ANOVA) is used to determine whether there are any statistically significant differences between the means of two or more independent (unrelated) groups. In this section, we suppose that we have been provided samples of size n from m distinct populations and that we want to use these data to test the hypothesis that the m population means are equal. Since the mean of a random variable depends only on a single factor, namely, the sample the variable is from, this scenario is said to constitute a *one-way analysis of variance*. In all of the models considered in this section, we assume that the data are normally distributed with the same (although unknown) variance σ^2.

Consider m independent samples, each of size n, where the members of the ith sample—$X_{i1}, X_{i2}, \ldots, X_{in}$—are normal random variables with unknown mean μ_i and unknown variance σ^2. That is,

$$X_{ij} \sim N(\mu_i, \sigma^2), i = 1, \ldots, m, j = 1, \ldots, n$$

We will be interested in testing

$$H_0 : \mu_1 = \mu_2 = \cdots = \mu_m$$

versus

$$H_1 : \text{not all the means are equal}$$

That is, we will be testing the null hypothesis that all the population means are equal against the alternative that at least two of them differ. Since there are a total of nm independent normal random variables X_{ij}, it follows that the sum of the squares of their standardized versions will be a chi-square random variable with nm degrees of freedom. To obtain estimators for the m unknown parameters $\mu_1, \ldots, \mu_m$, let $X_i.$ denote the average of all the elements in sample i; that is,

$$X_i. = \sum_{j=1}^{n} X_{ij}/n$$

The statistic

$$SS_W = \sum_{i=1}^{m} \sum_{j=1}^{n} (X_{ij} - X_i.)^2$$

is called the *within samples sum of squares*. So, $SS_W/(nm - m)$ is an estimator of σ^2.

Now, when all the population means are equal to μ, then the estimator of μ is the average of all the nm data values. That is, the estimator of μ is X.., given by

$$X.. = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} (X_{ij})}{nm} = \frac{\sum_{i=1}^{m} X_i.}{m}$$

Our second estimator of σ^2 will only be a valid estimator when the null hypothesis is true. So let us assume that H_0 is true and so all the population means μ_i are equal, say, $\mu_i = \mu$ for all i. In turn, the statistic

$$SS_b = n \sum_{i=1}^{m} (X_i. - X..)$$

is called the *between samples sum of squares*. When H_0 is true, $SS_b/(m - 1)$ is an estimator of σ^2.

Definition 55. *Because—if the reader is interested, a proof is given in the correspondent bibliography—it can be shown that $SS_b/(m - 1)$ will tend to exceed σ^2 when H_0 is not true, it is reasonable to let the test statistic be given by*

$$TS = \frac{SS_b/(m - 1)}{SS_W/(nm - m)}$$

and to reject H_0 when TS is sufficiently large.

To determine how large TS needs to be to justify rejecting H_0, we use the fact that it can be shown that if H_0 is true then SS_b and SS_W are independent. It follows from this that, when H_0 is true, TS has an F-distribution with m - 1 numerator and nm - m denominator degrees of freedom. Let $F_{m-1,nm-m,\alpha}$ denote the $100(1-\alpha)$ percentile of this distribution—that is,

$$P\{F_{m-1,nm-m} > F_{m-1,nm-m,\alpha}\} = \alpha$$

The significance level test of H0 is as follows:

$$\text{reject } H_0 \text{ if } \frac{SS_b/(m-1)}{SS_W/(nm-m)} > F_{m-1,nm-m,\alpha}$$
$$\text{accept } H_0 \text{ otherwise}$$

Example 6.6.2. An auto rental firm is using 15 identical motors that are adjusted to run at a fixed speed to test 3 different brands of gasoline. Each brand of gasoline is assigned to exactly 5 of the motors. Each motor runs on 10 gallons of gasoline until it is out of fuel. The following represents the total mileages obtained by the different motors:

Gas 1 :	220	251	226	246	260
Gas 2 :	244	235	232	242	225
Gas 3 :	252	272	250	238	256

Test the hypothesis that the average mileage obtained is not affected by the type of gas used. Use the 5 percent level of significance.

The first thing to note is that subtracting a constant from each data value will not affect the value of the test statistic. So we subtract 220 from each data value to get the following information:

Gas	Mileage					$\sum_j X_{ij}$	$\sum_j X_{ij}^2$
1	0	31	6	26	40	103	3,273
2	24	15	12	22	5	78	1,454
3	32	52	30	18	36	168	6,248

Now m = 3 and n = 5 and

$$X_1. = 103/5 = 20.6$$
$$X_2. = 78/5 = 15.6$$
$$X_3. = 168/5 = 33.6$$

$$X.. = (103 + 78 + 168)/15 = 23.2667$$
$$X_{..}^2 = 541.3393$$

Thus,

$$SS_b = 5[(20.6 - 23.2667)^2 + (15.6 - 23.2667)^2 + (33.6 - 23.2667)^2] = 863.3335$$

Also,

$$\sum\sum X_{ij}^2 = 3273 + 1454 + 6248 = 10975$$

and, from the sum of squares identity,

$$SS_W = 10975 - 15(541.3393) - 863.3335 = 1991.5785$$

The value of the test statistic is thus

$$TS = \frac{863.3335/2}{1991.5785/12} = 2.60$$

Now, if we look in a F distribution table[1], we see that $F_{2,12,.05} \approx 3.89$. Hence, because the value of the test statistic does not exceed 3.89, we cannot, at the 5 percent level of significance, reject the null hypothesis that the gasolines give equal mileage.

[1] See, for instance `http://www.socr.ucla.edu/applets.dir/f_table.html`

2.977	3.155	3.920	3.412	4.236	2.593	3.270	3.813	4.042	3.387

Table 6.1: Birthweight of ten babies (in kilograms).

Participant	A	B	C	D	E	F
Height [in]	67	72	61	72	60	67
Earnings [kUSD]	49	119	24	68	46	53

Table 6.2: Data based on Judge & Cable, 2004; earnings adjusted to 2002 US dollars.

6.7 Supervision session exercises

This Section includes all the exercises that will be focused on during this week's supervision session. The provided links include many more exercises (with solutions). You can also try to solve exercises from the book [5].

Exercise 6.1. Calculate the mean and standard deviation of the set of data shown in Table 6.1 [2].

Exercise 6.2. a) How many unique ways are there to arrange the letters in the word PRIOR? [3]

b) In Winden, Germany it rains on one third of the days. The local evening newspaper attempts to predict whether or not it will rain the following day. Three quarters of rainy days and three fifths of dry days are correctly predicted by the previous evenings paper. Given that this evenings paper predicts rain, what is the probability that it will actually rain tomorrow? [4]

Exercise 6.3. Suppose that the data from Table 6.2 has been collected in a small survey to explore a possible relationship between people's physical height and their annual earnings. [5]

a) Calculate the mean and standard deviation of height and earnings for these six people.

b) The six survey participants have been selected at random from a much larger population. Calculate estimates for the mean and standard deviation of the whole population.

c) The equation below gives an estimate for the Pearson's correlation coefficient of a whole population based on sample values.

$$r_{x,y} = \frac{\sum_{i=1}^{n}(x_i - m_x)(y_i - m_y)}{(n-1)s_x s_y} \tag{6.2}$$

Use this to estimate the correlation coefficient between height and earnings in the population from which the sample above was taken.

d) You are asked to test the hypothesis that there is a positive correlation between height and earnings. Use your answer from (c) and Table 6.3 to answer the following questions, in each case explaining how you arrive at your answer:

1. Does this sample show positive correlation between height and earnings?
2. Is it statistically significant?

Exercise 6.4. A company manufactures an electronic device to be used in a very wide temperature range. The company knows that increased temperature shortens the life time of the device, and a study is therefore performed in which the life time is determined as a function of temperature. The data in Table 6.4 is found.

[2] Exercise source: `https://sydney.edu.au/stuserv/documents/maths_learning_centre/descstats2010web.pdf`

[3] Exercise source: `https://www.khanacademy.org/math/precalculus/x9e81a4f98389efdf:prob-comb/x9e81a4f98389efdf:combinations/e/permutations_and_combinations_2`

[4] Exercise source: `http://www.mas.ncl.ac.uk/~nmf16/teaching/mas3301/solutions109.pdf`

[5] Exercise source: `https://blog.inf.ed.ac.uk/da16/files/2016/01/inf1-da-16-t8.pdf`

two-tail	p = 0.20	p = 0.10	p = 0.01	p = 0.001
one-tail	p = 0.10	p = 0.05	p = 0.005	p = 0.0005
N = 4	0.800	0.900	0.990	0.999
N = 5	0.687	0.805	0.959	0.991
N = 6	0.608	0.729	0.917	0.974
N = 7	0.551	0.669	0.875	0.951

Table 6.3: Critical values for Pearson's correlation.

Temperature in Celsius (t)	10	20	30	40	50	60	70	90	-80
Life time in hours (y)	420	365	285	220	176	117	69	34	5

Table 6.4: Temperature and life time of electronic device.

a) Calculate the 95% confidence interval for the slope in the usual linear regression model, which expresses the life time as a linear function of the temperature.

b) Can a relation between temperature and life time be documented on level 5%?

Exercise 6.5. The effectiveness of advertising for two rival products (Brand X and Brand Y) was compared. Market research at a local shopping centre was carried out, with the participants being shown adverts for two rival brands of coffee, which they then rated on the overall likelihood of them buying the product (out of 10, with 10 being "definitely going to buy the product"). Half of the participants gave ratings for one of the products, the other half gave ratings for the other product (see Table 6.5). Is there a significant difference between the the ratings given to each brand in terms of the likelihood of buying the product (use significance level 0.05)? Which test do we use? [6]

Exercise 6.6. A psychologist was interested in whether different TV shows lead to a more positive outlook on life. People were split into 4 groups and then taken to a room to view a program. The four groups saw: The Muppet Show, Futurama, The News, No program. After the program a blood sample was taken and serotonin levels measured (remember more serotonin means more happy). Figure 6.16 shows the data gathered. Carry out a one-way ANOVA by hand to test the hypothesis that some TV shows make people happier than others.

[6] Exercises source: `http://users.sussex.ac.uk/~grahamh/RM1web/Mann-Whitney%20worked%20example.pdf`
[7] Exercise source: `http://www.discoveringstatistics.com/docs/exampractice.pdf`

Brand X		Brand Y	
Participant	Rating	Participant	Rating
1	3	1	9
2	4	2	7
3	2	3	5
4	6	4	10
5	2	5	6
6	5	6	8

Table 6.5: Participant rating of coffees X and Y from Exercise 6.5.

	The Muppet Show	Futurama	BBC News	No Program
	11	4	4	7
	7	8	3	7
	8	6	2	5
	14	11	2	4
	11	9	3	3
	10	8	6	4
	5			4
				4
Mean	**9.43**	**7.67**	**3.33**	**4.75**
Variance	**8.95**	**5.87**	**2.27**	**2.21**
Grand Mean			**6.30**	
Grand Variance			**10.06**	

Figure 6.16: Serotonin levels after watching different programs from Exercise 6.6[7].

Birthweight in Kg	Deviations from Mean $score - mean$	Squared Deviations $(score - mean)^2$
2.977	0.5035	0.2535
3.155	0.3255	0.1060
3.920	0.4395	0.1932
3.412	0.0685	0.0047
4.236	0.7555	0.5708
2.593	0.8875	0.7877
3.270	0.2105	0.0443
3.813	0.3325	0.1106
4.042	0.5615	0.3153
3.387	0.0935	0.0087
Sum = 34.805	Sum = 0	Sum = 2.3948

Table 6.6: Calculations for Exercise 6.1.

Solution 6.1. See Table 6.6 and equations below.

$$Mean(\mu) = \frac{\text{sum of observations}}{\text{number of observations}} = \frac{34.805}{10} = 3.4805 \tag{6.3}$$

$$Variance(\sigma^2) = \frac{\text{sum of squared deviations}}{\text{number of observations - 1}} = \frac{2.3948}{9} = 0.266 \tag{6.4}$$

$$\text{Standard Deviation}(\sigma) = \sqrt{Variance} = \sqrt{0.266} = 0.5158 \tag{6.5}$$

Solution 6.2. a)

$$\frac{5!}{2!} = 60 \tag{6.6}$$

b) Let R be "rain", $\overline{R}$ be "dry", P be "rain predicted". We need to calculate $Probability(R|P)$. By

```
D <- data.frame(t=c(10,20,30,40,50,60,70,80,90),
                y=c(420,365,285,220,176,117,69,34,5))
fit <- lm(y ~ t, data=D)
summary(fit)


Call:
lm(formula = y ~ t, data = D)

Residuals:
   Min     1Q Median     3Q    Max
-21.02 -12.62  -9.16  17.71  29.64

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  453.556     14.394    31.5  8.4e-09 ***
t             -5.313      0.256   -20.8  1.5e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 19.8 on 7 degrees of freedom
Multiple R-squared:  0.984, Adjusted R-squared:  0.982
F-statistic:  432 on 1 and 7 DF,  p-value: 0.000000151
```

Figure 6.17: R printout from Exercise 6.4[8].

Bayes' theorem, this is:

$$\begin{aligned} Pr(R|P) &= \frac{Pr(R)Pr(P|R)}{Pr(R)Pr(P|R) + Pr(\overline{R})Pr(P|\overline{R})} \\ &= \frac{\frac{1}{3} \times \frac{3}{4}}{\frac{1}{3} \times \frac{3}{4} + \frac{2}{3} \times \frac{2}{5}} \\ &= \frac{\frac{3}{4}}{\frac{3}{4} + \frac{4}{5}} \\ &= \frac{15}{31} = 0.4839 \end{aligned} \tag{6.7}$$

Solution 6.3. a) For this sample of six people, their heights have mean 66.5 inches and standard deviation 4.72 inches; their earnings have mean \$59, 800 and standard deviation \$29, 500.

b) The sample means are appropriate estimators of the population mean height 66.5 inches and mean earnings \$59, 800. Appropriate estimates for the population standard deviation are 5.18 inches height and \$32, 300 earnings. This is using an $(n-1)$ denominator to account for the fact that we are using a small sample to estimate the value for a larger population.

c) The estimate of correlation coefficient in the population as a whole is +0.78.

d) (i) Yes, the sample does show a positive correlation, as the coefficient is greater than zero. (ii) Yes, this is statistically significant, at the 95% level, as it exceeds the critical one-tail value for $p = 0.05$ over $N = 6$ samples.

Solution 6.4. a) One could do all the regression computations to find the $\hat{\beta}_1$ and then subsequently use the formula for the confidence interval for β_1:

$$\hat{\beta}_1 \pm t_{1-\alpha/2} * \hat{\sigma}_{\beta_1} = \hat{\beta}_1 \pm t_{\alpha/2} * \hat{\sigma} \sqrt{\frac{1}{\sum_{i=1}^{n} (x_i - \bar{x})^{2'}}} \tag{6.8}$$

or run *lm* in R to find the result shown in Figure 6.17.

Brand X			Brand Y		
Participant	Rating	Rank	Participant	Rating	Rank
1	3	3	1	9	11
2	4	4	2	7	9
3	2	1.5	3	5	5.5
4	6	7.5	4	10	12
5	2	1.5	5	6	7.5
6	5	5.5	6	8	10

Table 6.7: Ranking all scores from Exercise 6.5.

From Figure 6.17 one can gather the information necessary to obtain $\hat{\beta}_1$:

$$\hat{\sigma}_{\beta_1} = \hat{\sigma}\sqrt{\frac{1}{\sum_{i=1}^{n}(x_i - \bar{x})^2}} = 0.2558, \tag{6.9}$$

and $t_{0.025}(7) = 2.364$ (in R: *qt(.975,7)*). Finally we get $-5.31 \pm 2.365 * 0.2558$.

b) Since the confidence interval does not include 0, it can be documented that there is a relationship between life time and temperature, also the p-value is $1.5 * 10^{-7} < 0.05 = \alpha$, which also give strong evidence against the null-hypothesis.

Solution 6.5. We have two conditions, with each participant taking part in only one of the conditions. The data are ratings (ordinal data), and hence a non-parametric test is appropriate - the Mann-Whitney U test (the non-parametric counterpart of an independent measures t-test).

Step 1: Rank all scores together, ignoring which group they belong to (Table 6.7.

Step 2: Add up the ranks for Brand X, to get R_1 and R_2: $R_1 = 3 + 4 + 1.5 + 7.5 + 1.5 + 5.5 = 23$
$R_2 = 11 + 9 + 5.5 + 12 + 7.5 + 10 = 55$.

Step 3: Select the larger rank. In this case its R_2.

Step 4: Calculate n_1, n_2 and n_x These are the number of participants in each group, and the number of people in the group that gave the larger rank total. In this case, $n_1 = n_2 = n_x = 6$.

Step 5: Find U. Note: R_x is the larger rank total.

$$\begin{aligned} U &= n_1 * n_2 + n_x * \frac{n_x + 1}{2} - R_x \\ U &= 6 * 6 + 6 * \frac{6 + 1}{2} - 55 \\ U &= 2 \end{aligned} \tag{6.10}$$

Step 6: Use a table of critical U values for the Mann-Whitney U Test. For $n_1 = 6$ and $n_2 = 6$, the critical value of U is 5 for a two-tailed test at the 0.05 significance level. To be significant, our obtained U has to be equal to or less than this critical value. Our obtained U is less than the critical value of U for a 0.05 significance level ($2 < 5$). There is a highly significant difference between the ratings given to each brand in terms of the likelihood of buying the product.

Solution 6.6. Tables 6.18 and 6.19 give the solution. This solution only gives you the calculations needed for answering the question. We suggest you carry out calculations by hand to understand how to get there, and to interpret these results correctly (i.e., what does this mean - do some show make people happier?).

[8] Exercise source: `https://02402.compute.dtu.dk/filemanager/02402/sharelatex-public/files/solutions-chapter5.pdf`

Descriptives

Serotonin Level

	N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Minimum	Maximum
					Lower Bound	Upper Bound		
The Muppet Show	7	9.4286	2.9921	1.1309	6.6614	12.1958	5.00	14.00
Futurama	6	7.6667	2.4221	.9888	5.1248	10.2085	4.00	11.00
BBC News	6	3.3333	1.5055	.6146	1.7534	4.9133	2.00	6.00
No Program (Control)	8	4.7500	1.4880	.5261	3.5060	5.9940	3.00	7.00
Total	27	6.2963	3.1722	.6105	5.0414	7.5512	2.00	14.00

Figure 6.18: Descriptive statistics of gathered data from Exercise 6.6.

ANOVA

Serotonin Level

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	151.749	3	50.583	10.588	.000
Within Groups	109.881	23	4.777		
Total	261.630	26			

Figure 6.19: ANOVA for Exercise 6.6.

Appendices

Appendix A

Appendix I: Truth Tables

p	p	$\sim(p)$
T	T	F
F	F	T

p	q	$p \wedge q$	$\sim(p \wedge q)$
T	T	T	F
T	F	F	T
F	T	F	T
F	F	F	T

p	q	$p \vee q$	$\sim(p \vee q)$
T	T	T	F
T	F	T	F
F	T	T	F
F	F	F	T

p	q	$p \oplus q$	$\sim(p \oplus q)$
T	T	F	T
T	F	T	F
F	T	T	F
F	F	F	T

p	q	$p \implies q$	$\sim(p \implies q)$
T	T	T	F
T	F	F	T
F	T	T	F
F	F	T	F

p	q	$p \iff q$	$\sim(p \iff q)$
T	T	T	F
T	F	F	T
F	T	F	T
F	F	T	F

Equivalences:

Identity laws:

p	$p \wedge T$	p
T	T	T
F	F	F

p	$p \wedge F$	F
T	F	F
F	F	F

p	$p \vee T$	T
T	T	T
F	T	T

p	$p \vee F$	p
T	T	T
F	F	F

Idempotent laws:

p	$p \wedge p$	p
T	T	T
F	F	F

p	$p \vee p$	p
T	T	T
F	F	F

Negation laws:

p	$p \vee \neg p$	T
T	T	T
F	T	T

p	$p \wedge \neg p$	F
T	F	F
F	F	F

Associative laws:

p	q	r	$p \vee (q \vee r)$	$(p \vee q) \vee r$
T	T	T	T	T
T	T	F	T	T
T	F	T	T	T
T	F	F	T	T
F	T	T	T	T
F	T	F	T	T
F	F	T	T	T
F	F	F	F	F

p	q	r	$p \wedge (q \wedge r)$	$(p \wedge q) \wedge r$
T	T	T	T	T
T	T	F	F	F
T	F	T	F	F
T	F	F	F	F
F	T	T	F	F
F	T	F	F	F
F	F	T	F	F
F	F	F	F	F

Distributive laws:

p	q	r	$p \vee (q \wedge r)$	$(p \vee q) \wedge (p \vee r)$
T	T	T	T	T
T	T	F	T	T
T	F	T	T	T
T	F	F	T	T
F	T	T	T	T
F	T	F	F	F
F	F	T	F	F
F	F	F	F	F

p	q	r	$p \wedge (q \vee r)$	$(p \wedge q) \vee (p \wedge r)$
T	T	T	T	T
T	T	F	T	T
T	F	T	T	T
T	F	F	F	F
F	T	T	F	F
F	T	F	F	F
F	F	T	F	F
F	F	F	F	F

De Morgan laws:

p	q	$\neg(p \vee q)$	$\neg p \wedge \neg q$
T	T	F	F
T	F	F	F
F	T	F	F
F	F	T	T

p	q	$\neg(p \wedge q)$	$\neg p \vee \neg q$
T	T	F	F
T	F	T	T
F	T	T	T
F	F	T	T

Absorption laws:

p	q	$p \vee (p \wedge q)$	p
T	T	T	T
T	F	T	T
F	T	F	F
F	F	F	F

p	q	$p \wedge (p \vee q)$	p
T	T	T	T
T	F	T	T
F	T	F	F
F	F	F	F

Involving conditional and biconditional statements:

p	q	$p \implies q$	$\neg p \vee q$
T	T	T	T
T	F	F	F
F	T	T	T
F	F	T	T

p	q	$p \implies q$	$\neg q \implies \neg p$
T	T	T	T
T	F	F	F
F	T	T	T
F	F	T	T

p	q	$\neg(p \iff q)$	$p \iff \neg q$
T	T	F	F
T	F	T	T
F	T	T	T
F	F	F	F

p	q	$p \iff q$	$(p \implies q) \wedge (q \implies p)$	$\neg p \iff \neg q$	$(p \wedge q) \vee (\neg p \wedge \neg q)$
T	T	T	T	T	T
T	F	F	F	F	F
F	T	F	F	F	F
F	F	T	T	T	T

p	q	r	$(p \implies q) \wedge (p \implies r)$	$p \implies (q \wedge r)$
T	T	T	T	T
T	T	F	F	F
T	F	T	F	F
T	F	F	F	F
F	T	T	T	T
F	T	F	T	T
F	F	T	T	T
F	F	F	T	T

Bibliography

[1] Paul Dawkins. Calculus i, 2007.

[2] Kirsti Hemmi and Clas Löfwall. Why do we need proof. In *CERME 6, Congress of the European Society for Research in Mathematics Education*. Institut National de Recherche Pedagogique, 2010.

[3] Danielle Pierce and German Junior. Local students make honor roll at oregon state university.

[4] Kenneth H Rosen. Discrete mathematics and its applications. *AMC*, 10:12, 2007.

[5] Sheldon M. Ross. Introduction to probability and statistics for engineers and scientists, 2009.