chapter 2

Facts about Functions

All my discoveries were simply improvements in notation.
—Gottfried Wilhelm von Leibniz (1646–1716)[1]

Functions can often make life simpler. In this chapter we'll start with the basic notions and notations for functions. Then we'll introduce some functions that are especially important in computer science. Since programs can be functions and functions can be programs, we'll spend some time discussing techniques for constructing new functions from simpler ones. We'll also discuss other properties of functions that are useful in problem solving.

chapter guide

Section 2.1 introduces the basic ideas of functions—what they are, and how to represent them. We'll give many examples, including several functions that are especially useful to computer scientists.

Section 2.2 introduces the important idea of composition as a way to combine functions to construct new functions. We'll see that the map function is a useful tool for constructing functions that calculate lists.

Section 2.3 introduces three important properties of functions—injective, surjective, and bijective. We'll see how these properties are used when we discuss the pigeonhole principle, cryptology, and hash functions.

Section 2.4 gives a brief introduction to techniques for comparing infinite sets. We'll discuss the ideas of countable and uncountable sets. We'll introduce the diagonalization technique, and we'll discuss whether we can compute everything.

[1]Leibniz introduced the word “function” around 1692. He is responsible for such diverse ideas as binary arithmetic, symbolic logic, combinatorics, and calculus. Around 1694 he built a calculating machine that could add and multiply.

2.1 Definitions and Exampless

In this section we'll give the definition of a function along with various ways to describe functions. We'll also spend some time with functions that are very useful in computer science.

2.1.1 Definition of a Function

Suppose A and B are sets and for each element in A we associate exactly one element in B. Such an association is called a *function* from A to B. The main idea is that each element of A is associated with *exactly one* element of B. In other words, if $x \in A$ is associated with $y \in B$, then x is not associated with any other element of B.

Functions are normally denoted by letters like f, g, and h or other descriptive names or symbols. If f is a function from A to B and f associates the element $x \in A$ with the element $y \in B$, then we write $f(x) = y$ or $y = f(x)$. The expression $f(x)$ is read, "f of x," or "f at x," or "f applied to x." When $f(x) = y$, we often say, "f maps x to y." Some other words for "function" are *mapping*, *transformation*, and *operator*.

Describing Functions

Functions can be described in many ways. Sometimes a formula will do the job. For example, the function f from N to N that maps every natural number x to its square can be described by the following formula:

$$f(x) = x^2.$$

Other times, we'll have to write down all possible associations. For example, the following associations define a function g from $A = \{a, b, c\}$ to $B = \{1, 2, 3\}$:

$$g(a) = 1, g(b) = 1, \text{ and } g(c) = 2.$$

We can also describe a function by a drawing a figure. For example, Figure 2.1 shows three ways to represent the function g. The top figure uses Venn diagrams together with a digraph. The lower left figure is a digraph. The lower-right figure is the familiar Cartesian graph, in which each ordered pair $(x, g(x))$ is plotted as a point.

Figure 2.2 shows two associations that are not functions. Be sure to explain why these associations do not represent functions from A to B.

Terminology

To communicate with each other about functions, we need to introduce some more terminology. If f is a function from A to B, we denote this by writing

$$f : A \to B.$$

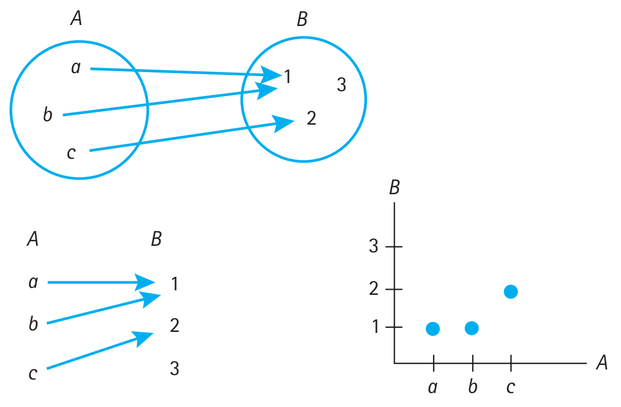


Figure 2.1 Three ways to describe the same function.

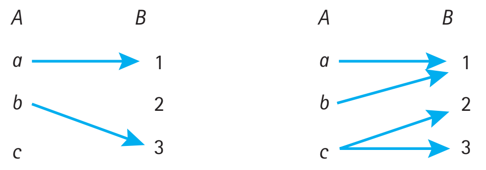


Figure 2.2 Truth tables.

The set A is the *domain* of f and the set B is the *codomain* of f. We also say that f has *type* $A \to B$. The expression $A \to B$ denotes the set of all functions from A to B.

If $f(x) = y$, then x is called an *argument* of f, and y is called a *value* of f. If the domain of f is the Cartesian product $A_1 \times \cdots \times A_n$, we say f has *arity* n or f has n *arguments*. In this case, if $(x_1, \ldots, x_n) \in A_1 \times \cdots \times A_n$. then

$$f(x_1, \ldots, x_n)$$

denotes the value of f at $(x_1, \ldots, x_n)$. A function f with two arguments is called a *binary* function and we have the option of writing $f(x, y)$ in the popular *infix* form $x\ f\ y$. For example, $4 + 5$ is usually preferable to $+(4, 5)$.

Ranges, Images, and Pre-Images

At times it is necessary to discuss certain subsets of the domain and codomain of a function $f : A \to B$. The *range* of f, denoted by range(f), is the set of elements in the codomain B that are associated with some element of A. In other words, we have

$$\text{range}(f) = \{f(a) \mid a \in A\} .$$

For any subset $S \subset A$, the *image* of S under f, denoted by $f(S)$, is the set of elements in B that are associated with some element of S. In other words, we have

$$f(S) = \{f(x) \mid x \in S\} .$$

Notice that we always have the special case $f(A) = \text{range}(f)$. Notice also that images allow us to think of f not only as a function from A to B, but also as a function from power(A) to power(B).

For any subset $T \subset B$ the *pre-image* of T under f, denoted by $f^{-1}(T)$, is the set of elements in A that associate with elements of T. In other words, we have

$$f^{-1}(T) = \{a \in A \mid f(a) \in T\} .$$

Notice that we always have the special case $f^{-1}(B) = A$. Notice also that pre-images allow us to think of f^{-1} as a function from power(B) to power(A).

example 2.1 Sample Notations

Consider the function $f : \{a, b, c\} \to \{1, 2, 3\}$ defined by $f(a) = 1$, $f(b) = 1$, and $f(c) = 2$. We can make the following observations.

f has type $\{a, b, c\} \to \{1, 2, 3\}$.
The domain of f is $\{a, b, c\}$.
The codomain of f is $\{1, 2, 3\}$.
The range of f is $\{1, 2\}$.

Some sample images are

$f(\{a\}) = \{1\}$,
$f(\{a, b\}) = \{1\}$,
$f(A) = f(\{a, b, c\}) = \{1, 2\} = \text{range}(f)$.

Some sample pre-images are

$f^{-1}(\{1, 3\}) = \{a, b\}$,
$f^{-1}(\{3\}) = \varnothing$,
$f^{-1}(B) = f^{-1}(\{1, 2, 3\}) = \{a, b, c\} = A$.

end example

example 2.2 Functions and Not Functions

Let P be the set of all people, alive or dead. We'll make some associations and discuss whether each is function of type $P \to P$.

1. $f(x)$ is a parent of x.

   In this case f is *not* a function of type $P \to P$ because people have two parents. For example, if q has mother m and father p, then $f(q) = m$ and $f(q) = p$, which is contrary to the requirement that each domain element be associated with exactly one codomain element.

2. $f(x)$ is the mother of x.

   In this case f *is* a function of type $P \to P$ because each person has exactly one mother. In other words, each $x \in P$ maps to exactly one person, the mother of x. If m is a mother, what is the pre-image of the set $\{m\}$ under f?

3. $f(x)$ is the oldest child of x.

   In this case f is *not* a function of type $P \to P$ because some person has no children. Therefore, $f(x)$ is not defined for some $x \in P$.

4. $f(x)$ is the set of all children of x.

   In this case f is *not* a function of type $P \to P$ because each person is associated with a set of people rather than a person. However, f *is* a function of type $P \to \text{power}(P)$. Can you see why?

end example

example

2.3 Tuples Are Functions

Any ordered sequence of objects can be thought of as a function. For example, the tuple (22, 14, 55, 1, 700, 67) can be thought of as a listing of the values of the function

$$f : \{0, 1, 2, 3, 4, 5\} \to N$$

where f is defined by the equality

$$f(0), f(1), f(2), f(3), f(4), f(5)) = (22, 14, 55, 1, 700, 67).$$

Similarly, any infinite sequence of objects can also be thought of as a function. For example, suppose that $(b_0, b_1, \ldots, b_n, \ldots)$ is an infinite sequence of objects from a set S. Then the sequence can be thought of as a listing of values in the range of the function $f : \mathrm{N} \to S$ defined by $f(n) = b_n$.

end example

example

2.4 Functions and Binary Relations

Any function can be defined as a special kind of binary relation. A function $f : A \to B$ is a binary relation from A to B such that no two ordered pairs have the same first element. We can also describe this uniqueness condition as: If $(a, b), (a, c) \in f$, then $b = c$. The notation $f(a) = b$ is normally preferred over the relational notations $f(a, b)$ and $(a, b) \in f$.

end example

Equality of Functions

Two functions are equal if they have the same type and the same values for each domain element. In other words, if f and g are functions of type $A \to B$, then f and g are said to be *equal* if $f(x) = g(x)$ for all $x \in A$. If f and g are equal, we write

$$f = g.$$

For example, suppose f and g are functions of type $N \to N$ defined by the formulas $f(x) = x + x$ and $g(x) = 2x$. It's easy to see that $f = g$.

Defining a Function by Cases

Functions can often be defined by cases. For example, the absolute value function "abs" has type $R \to R$, and it can be defined by the following rule:

$$\text{abs}(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0. \end{cases}$$

A definition by cases can also be written in terms of the *if-then-else* rule. For example, we can write the preceding definition in the following form:

$$\text{abs}(x) = \text{if } x \geq 0 \text{ then } x \text{ else } -x.$$

The if-then-else rule can be used more than once if there are several cases to define. For example, suppose we want to classify the roots of a quadratic equation having the following form:

$$ax^2 + bx + c = 0.$$

We can define the function "classifyRoots" to give the appropriate statements as follows:

```
classifyRoots($a, b, c$) = if $b^2 - 4ac > 0$ then
                               "The roots are real and distinct."
                           else if $b^2 - 4ac < 0$ then
                               "The roots are complex conjugates."
                           else
                               "The roots are real and repeated."
```

2.1.2 Some Useful Functions

Now let’s look at some functions that are especially useful in computer science. These functions are used for tasks such as analyzing properties of data, analyzing properties of programs, and constructing programs.

The Floor and Ceiling Functions

Let’s discuss two important functions that “integerize” real numbers by going down or up to the nearest integer. The *floor* function has type $R \to Z$ and is defined by setting floor(x) to the closest integer less than or equal to x. For example, floor(8) = 8, floor(8.9) = 8, and floor(–3.5) = –4. A useful shorthand notation for floor(x) is

$$\lfloor x \rfloor.$$

The *ceiling* function also has type $R \to Z$ and is defined by setting ceiling(x) to the closest integer greater than or equal to x. For example, ceiling(8) = 8, ceiling(8.9) = 9, and ceiling(–3.5) = –3. The shorthand notation for ceiling(x) is

$$\lceil x \rceil.$$

Figure 2.3 gives a few sample values for the floor and ceiling functions.

Can you find some relationships between floor and ceiling? For example, is $\lfloor x \rfloor = \lceil x - 1 \rceil$? It’s pretty easy to see that if x is an integer, then the statement is false. But if x is not an integer, then there is some integer n such that $n < x < n + 1$ and thus also $n - 1 < x - 1 < n$. In this case it follows that $\lfloor x \rfloor = n = \lceil x - 1 \rceil$. So we can say that $\lfloor x \rfloor = \lceil x - 1 \rceil$ if and only if $x \notin Z$. This property and some others are listed below. The proofs are similar to the argument we just made.

Floor and Ceiling Properties (2.1)

a. $\lfloor x + 1 \rfloor = \lfloor x \rfloor + 1$.

b. $\lceil x - 1 \rceil = \lceil x \rceil - 1$.

c. $\lceil x \rceil = \lfloor x \rfloor$ if and only if $x \in Z$.

d. $\lfloor x \rfloor = \lceil x - 1 \rceil$ if and only if $x \notin Z$.

e. $\lfloor x \rfloor = \lceil x - 1 \rceil$ if and only if $x \notin Z$.

x	–2.0	–1.7	–1.3	–1.0	–0.7	–0.3	0.0	0.3	0.7	1.0	1.3	1.7	2.0
$\lfloor x \rfloor$	–2	–2	–2	–1	–1	–1	0	0	0	1	1	1	2
$\lceil x \rceil$	–2	–1	–1	–1	0	0	0	1	1	1	2	2	2

Figure 2.3 Some floor and ceiling values.

Greatest Common Divisor

Let's recall from Section 1.1 that an integer d *divides* an integer n if $d \neq 0$ and there is an integer k such that $n = dk$, and we denote this fact with $d \mid n$. Our focus here will be on the largest of all common divisions for two integers.

Definiton of Greatest Common Divisor

The *greatest common divisor* of two integers, not both zero, is the largest integer that divides them both. We denote the greatest common divisor of a and b by

$$\gcd(a, b).$$

For example, the common divisors of 12 and 18 are ±1, ±2, ±3, ±6. So the greatest common divisor of 12 and 18 is 6, so we write $\gcd(12, 18) = 6$. Other examples are $\gcd(-44, -12) = 4$ and $\gcd(5, 0) = 5$. If $a \neq 0$, then $\gcd(a, 0) = |a|$. An important and useful special case occurs when $\gcd(a, b) = 1$. In this case a and b are said to be *relatively prime*. For example, 9 and 4 are relatively prime.

Here are some properties of the greatest common divisor function.

Greatest Common Divisor Properties (2.2)

a. $\gcd(a, b) = \gcd(b, a) = \gcd(a, -b)$.

b. $\gcd(a, b) = \gcd(b, a - bq)$ for any integer q.

c. If $g = \gcd(a, b)$, then there are integers x and y such that $g = ax + by$.

d. If $d \mid ab$ and $\gcd(d, a) = 1$, then $d \mid b$.

Property (2.2a) confirms that the ordering of the arguments doesn't matter and that negative numbers have positive greatest common divisors. For example, $\gcd(-4, -6) = \gcd(-4, 6) = \gcd(6, -4) = \gcd(6, 4) = 2$. We'll see shortly how property (2.2b) can help us compute greatest common divisors. Property (2.2c) says that we can write $\gcd(a, b)$ in terms of a and b. For example, $\gcd(15, 9) = 3$, and we can write 3 in terms of 15 and 9 as

$$3 = \gcd(15, 9) = 15(2) + 9(-3).$$

Property (2.2d) is a divisibility property that we'll be using later.

Now let's get down to brass tacks and describe an algorithm to compute the greatest common divisor. Most of us recall from elementary school that we can divide an integer a by a nonzero integer b to obtain two other integers, a quotient q and a remainder r, which satisfy an equation like the following:

$$a = bq + r.$$

For example, if $a = -16$ and $b = 3$, then we can write many equations, each with different values for q and r. For example, the following four equations all have the form $a = bq + r$:

$$
\begin{aligned}
-16 &= 3 \cdot (-4) + (-4) \\
-16 &= 3 \cdot (-5) + (-1) \\
-16 &= 3 \cdot (-6) + 2 \\
-16 &= 3 \cdot (-7) + 5
\end{aligned}
$$

In mathematics and computer science the third equation is by far the most useful. In fact it's a result of a theorem called the *division algorithm*, which we'll state for the record.

Division Algorithm

If a and b are integers and $b \neq 0$, then there are unique integers q and r such that $a = bq + r$, where $0 \leq r < |b|$.

The division algorithm together with property (2.2b) gives us the seeds of an algorithm to compute greatest common divisors. Suppose a and b are integers and $b \neq 0$. The division algorithm gives us the equation $a = bq + r$, where $0 \leq r < |b|$. Solving the equation for r gives $r = a - bq$. This fits the form of (2.2b). So we have the nice equation

$$\gcd(a, b) = \gcd(b, a - bq) = \gcd(b, r).$$

The important point about this equation is that the numbers in $\gcd(b, r)$ are getting closer to zero. Let's see how we can use this equation to compute the greatest common divisor. For example, to compute gcd(315, 54), we apply the division algorithm to obtain the equation $315 = 54 \cdot 5 + 45$. Thus we know that

$$\gcd(315, 54) = \gcd(54, 45).$$

Now apply the division algorithm again to obtain $54 = 45 \cdot 1 + 9$. So we have

$$\gcd(315, 54) = \gcd(54, 45) = \gcd(45, 9).$$

Continuing, we have $45 = 9 \cdot 5 + 0$, which extends our computation to

$$\gcd(315, 54) = \gcd(54, 45) = \gcd(45, 9) = \gcd(9, 0) = 9.$$

The algorithm that we have been demonstrating is called *Euclid's algorithm*. Since greatest common divisors are always positive, we'll describe the algorithm to calculate $\gcd(a, b)$ for the case in which a and b are natural numbers that are not both zero.

Euclid’s Algorithm (2.3)

Input natural numbers a and b, not both zero, and output gcd(a, b).

```
while $b > 0$ do
    Construct $a = bq + r$, where $0\ ^2\ r < b$;      (by the division algorithm)
    $a := b$;
    $b := r$
od;
Output $a$.
```

We can use Euclid’s algorithm to show how property (2.2c) is satisfied. The idea is to keep track of the equations $a = bq + r$ from each execution of the loop. Then work backwards through the equations to solve for gcd(a, b) in terms of a and b. For example, in our calculation of gcd(315, 54) we obtained the three equations

$$\begin{aligned} 315 &= 54 \cdot 5 + 45 \\ 54 &= 45 \cdot 1 + 9 \\ 45 &= 9 \cdot 5 + 0. \end{aligned}$$

Starting with the second equation, we can solve for 9. Then we can use the first equation to replace 45. The result is an expression for $9 = \gcd(315, 54)$ written in terms of 315 and 54 as $9 = 315 \cdot (-1) + 54 \cdot 6$.

The Mod Function

If a and b are integers, where $b > 0$, then the division algorithm states that there are two unique integers q and r such that

$$a = bq + r \qquad \text{where} \qquad 0 \le r < b.$$

We say that q is the quotient and r is the remainder upon division of a by b. The remainder $r = a - bq$ is the topic of interest.

Definition of Mod Function

If a and b are integers with $b > 0$, then the remainder upon the division of a by b is denoted

$$a \ mod \ b$$

If we agree to fix n as a positive integer, then x mod n takes values in the set $\{0, 1, \ldots, n - 1\}$, which is the set of possible remainders obtained upon division of any integer x by n. For example, each row of the table in Figure 2.4 gives some sample values for x mod n.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x mod 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x mod 2	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x mod 3	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0
x mod 4	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
x mod 5	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0

Figure 2.4 Sample values of mod function.

We sometimes let N_n denote the set

$$N_n = \{0, 1, 2, \ldots, n - 1\}.$$

For example, $N_0 = \varnothing$, $N_1 = \{0\}$, and $N_2 = \{0, 1\}$. So for fixed n, the function f defined by $f(x) = x \text{ mod } n$ has type $Z \to N_n$.

A Formula for Mod

Can we find a formula for a mod b in terms of a and b? Sure. We have the following formula

$$a \text{ mod } b = r = a - bq, \qquad \text{where} \qquad 0 \le r \le b.$$

So we'll have a formula for a mod b if we can find a formula for the quotient q in terms of a and b. Starting with the the inequality $0 \le r < b$ we have the following sequence of inequalities.

$$\begin{aligned}
0 &\le r < b. \\
-b &< -r \le 0, \\
a - b &< a - r \le a, \\
\frac{a-b}{b} &< \frac{a-r}{b} \le \frac{a}{b}, \\
\frac{a}{b} - 1 &< \frac{a-r}{b} \le \frac{a}{b}, \\
\frac{a}{b} - 1 &< q \le \frac{a}{b}, \qquad \text{since } q = \frac{a-r}{b}.
\end{aligned}$$

Since q is an integer, the last inequality implies that q can be written as the floor expression

$$q = \lfloor a/b \rfloor$$

Since $r = a - b \cdot q$, we have the following representation of r when $b > 0$.

$$r = a - b\lfloor a/b \rfloor.$$

This gives us a formula for the mod function.

Formula for Mod Function

$$a \text{ mod } b = a - b\lfloor a/b \rfloor.$$

Properties of Mod

The mod function has many properties. For example, the definition of mod tells us that $0 < x \text{ mod } n < n$ for any integer x. So for any integer x we have

$$(x \text{ mod } n) \text{ mod } n = x \text{ mod } n$$

and

$$x \text{ mod } n = x \text{ iff } 0 \leq x < n.$$

The following list contains some of the most useful properties of the mod function.

Mod Function Properties (2.4)

a. $x \text{ mod } n = y \text{ mod } n$ iff n divides $x - y$ iff $(x - y) \text{ mod } n = 0$.

b. $(x + y) \text{ mod } n = ((x \text{ mod } n) + (y \text{ mod } n)) \text{ mod } n$.

c. $(xy) \text{ mod } n = ((x \text{ mod } n)(y \text{ mod } n)) \text{ mod } n$.

d. If $ax \text{ mod } n = ay \text{ mod } n$ and $\gcd(a, n) = 1$, then $x \text{ mod } n = y \text{ mod } n$.

e. If $\gcd(a, n) = 1$, then $1 \text{ mod } n = ax \text{ mod } n$ for some integer x.

Proof: We'll prove parts (a) and (d) and discuss the other properties in the exercises. Using the definition of mod we can write

$$x \text{ mod } n = x - nq_1 \text{ and } y \text{ mod } n = y - nq_2$$

for some integers q_1 and q_2. Now we have a string of iff statements.

$$\begin{aligned} x \text{ mod } n = y \text{ mod } n \quad & \text{iff} \quad x - nq_1 = y - nq_2 \\ & \text{iff} \quad x - y = n(q_1 - q_2) \\ & \text{iff} \quad n \text{ divides } (x - y) \\ & \text{iff} \quad (x - y) \text{ mod } n = 0. \end{aligned}$$

So part (a) is true.

For part (d), assume that $ax \text{ mod } n = ay \text{ mod } n$ and $\gcd(a, n) = 1$. By part (a) we can say that n divides $(ax - ay)$. So n divides the product $a(x - y)$. Since $\gcd(a, n) = 1$, it follows from (2.2d) that n divides $(x - y)$. So again by part (a) we have $x \text{ mod } n = y \text{ mod } n$. QED.

example **2.5 Converting Decimal to Binary**

How can we convert a decimal number to binary? For example, the decimal number 53 has the binary representation 110101. The rightmost bit (binary digit) in this representation of 53 is 1 because 53 is an odd number. In general, we can find the rightmost bit (binary digit) of the binary representation of a natural decimal number x by evaluating the expression x mod 2. In our example, 53 mod 2 = 1, which is the rightmost bit.

So we can apply the division algorithm, dividing 53 by 2, to obtain the rightmost bit as the remainder. This gives us the equation

$$53 = 2 \cdot 26 + 1.$$

Now do the same thing for the quotient 26 and the succeeding quotients.

$$\begin{aligned} 53 &= 2 \cdot 26 + 1 \\ 26 &= 2 \cdot 13 + 0 \\ 13 &= 2 \cdot 6 + 1 \\ 6 &= 2 \cdot 3 + 0 \\ 3 &= 2 \cdot 1 + 1 \\ 1 &= 2 \cdot 0 + 1 \\ 0. & \quad \text{(done)} \end{aligned}$$

We can read off the remainders in the above equations from bottom to top to obtain the binary representation 110101 for 53. An important point to notice is that we can represent any natural number x in the form

$$x = 2\lfloor x/2 \rfloor + x \bmod 2.$$

So an algorithm to convert x to binary can be implemented with the floor and mod functions.

end example

The Log Function

The "log" function—which is shorthand for logarithm—measures the size of exponents. We start with a positive real number $b \neq 1$. If x is a positive real number, then

$$\log_b x = y \text{ means } b^y = x,$$

and we say, "log base b of x is y."

The base-2 log function $\log_2$ occurs frequently in computer science because many algorithms make binary decisions (two choices) and binary trees are useful data structures. For example, suppose we have a binary search tree with 16 nodes having the structure shown in Figure 2.5.

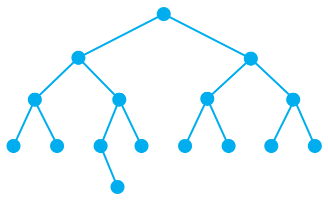


Figure 2.5 Sample binary tree.

x	1	2	4	8	16	32	64	128	256	512	1024
$\log_2 x$	0	1	2	3	4	5	6	7	8	9	10

Figure 2.6 Sample log values.

The depth of the tree is 4, so a maximum of 5 comparisons are needed to find any element in the tree. Notice in this case that $16 = 2^4$, so we can write the depth in terms of the number of nodes: $4 = \log_2 16$. Figure 2.6 gives a few choice values for the $\log_2$ function.

Of course, $\log_2$ takes real values also. For example, if $8 < x < 16$, then

$$3 < \log_2 x < 4.$$

For any real number $b > 1$, the function $\log_b$ is an increasing function with the positive real numbers as its domain and the real numbers as its range. In this case the graph of $\log_b$ has the general form shown in Figure 2.7. What does the graph look like if $0 < b < 1$?

The log function has many properties. For example, it's easy to see that

$$\log_b 1 = 0 \text{ and } \log_b b = 1.$$

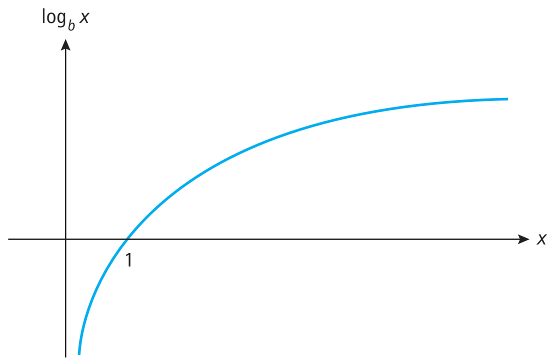


Figure 2.7 Graph of a log function.

The following list contains some of the most useful properties of the log function. We'll leave the proofs as exercises in applying the definition of log.

Log Function Properties (2.5)

a. $\log_b (b^x) = x$.

b. $\log_b (x\, y) = \log_b x + \log_b y$.

c. $\log_b(x^y) = y \log_b x$.

d. $\log_b (x/y) = \log_b x - \log_b y$.

e. $\log_a x = (\log_a b)(\log_b x)$. (change of base)

These properties are useful in the evaluation of log expressions. For example, suppose we need to evaluate the expression $\log_2(2^7 3^4)$. Make sure you can justify each step in the following evaluation:

$$\log_2(2^7 3^4) = \log_2(2^7) + \log_2(3^4) = 7 \log_2(2) + 4 \log_2(3) = 7 + 4 \log_2(3).$$

At this point we're stuck for an exact answer. But we can make an estimate. We know that $1 = \log_2(2) < \log_2(3) < \log_2(4) = 2$. Therefore, $1 < \log_2(3) < 2$. Thus we have the following estimate of the answer:

$$11 < \log_2(2^7 3^4) < 15.$$

2.1.3 Large Partial Functions

A *partial function* from A to B is like a function except that it might not be defined for some elements of A. In other words, some elements of A might not be associated with any element of B. But we still have the requirement that if $x \in A$ is associated with $y \in B$, then x can't be associated with any other element of B. For example, we know that division by zero is not allowed. Therefore, $\div$ is a partial function of type $R \times R \to R$ because $\div$ is not defined for all pairs of the form $(x, 0)$.

When discussing partial functions, to avoid confusion we use the term *total function* to mean a function that is defined on all its domain. Any partial function can be transformed into a total function. One simple technique is to shrink the domain to the set of elements for which the partial function is defined. For example, $\div$ is a total function of type $R \times (R - \{0\}) \to R$.

A second technique keeps the domain the same but increases the size of the codomain. For example, suppose $f : A \to B$ is a partial function. Pick some symbol that is not in B, say $\# \notin B$, and assign $f(x) = \#$ whenever $f(x)$ is not defined. Then we can think of f as a total function of type $A \to B \cup \{\#\}$. In programming, the analogy would be to pick an error message to indicate that an incorrect input string has been received.

Exercises

Definitions and Examples

1. Describe all possible functions for each of the following types.

a. $\{a, b\} \to \{1\}$.
b. $\{a\} \to \{1, 2, 3\}$.
c. $\{a, b\} \to \{1, 2\}$.
d. $\{a, b, c\} \to \{1, 2\}$.

2. Suppose we have a function $f : N \to N$ defined by $f(x) = 2x + 1$. Describe each of the following sets, where E and O denote the sets of even and odd natural numbers, respectively.

a. range(f). b. $f(E)$. **c.** $f(O)$.
d. $f(\varnothing)$. **e.** $f^{-1}(E)$. f. $f^{-1}(O)$.

Some Useful Functions

3. Evaluate each of the following expressions.

a. $\lfloor -4.1 \rfloor$. b. $\lceil -4.1 \rceil$. **c.** $\lfloor 4.1 \rfloor$ d. $\lceil 4.1 \rceil$.

4. Evaluate each of the following expressions.

a. gcd(–12, 15). b. gcd(98, 35). **c.** gcd(872, 45).

5. Find gcd(296, 872) and write the answer in the form $296x + 872y$.

6. Evaluate each of the following expressions.

a. 15 mod 12. b. –15 mod 12.
c. 15 mod (–12). d. –15 mod (–12).

7. Let $f : N_6 \to N_6$ be defined by $f(x) = 2x \bmod 6$. Find the image under f of each of the following sets:

a. $\varnothing$. b. $\{0, 3\}$. **c.** $\{2, 5\}$.
d. $\{3, 5\}$. **e.** $\{1, 2, 3\}$. f. N_6.

8. For a real number x, let trunc(x) denote the truncation of x, which is the integer obtained from x by deleting the part of x to the right of the decimal point.

a. Write the floor function in terms of trunc.
b. Write the ceiling function in terms of trunc.

9. For integers x and $y \neq 0$, let $f(x, y) = x - y\ \mathrm{trunc}(x/y)$, where trunc is from Exercise 8. How does f compare to the mod function?

10. Does it make sense to extend the definition of the mod function to real numbers? What would be the range of the function $f : R \to R$ defined by $f(x) = x \bmod 2.5$?

11. Evaluate each of the following expressions.

a. $\log_5 625$. b. $\log_2 8192$. **c.** $\log_3 (1/27)$.

12. For a subset S of a set U, the *characteristic* function $\chi_S : U \to \{0, 1\}$ is a test for membership in S and is defined by

$$\chi_S(x) = \text{if } x \in S \text{ then } 1 \text{ else } 0.$$

a. Verify that the following equation is correct for subsets A and B of U.

$$\chi_{A\cup B}(x) = \chi_A(x) + \chi_B(x) - \chi_A(x)\chi_B(x)$$

b. Find a formula for $\chi_{A\cap B}(x)$ in terms of $\chi_A(x)$ and $\chi_B(x)$.

c. Find a formula for $\chi_{A-B}(x)$ in terms of $\chi_A(x)$ and $\chi_B(x)$.

13. Given a function $f : A \to A$. An element $a \in A$ is called a *fixed point* of f if $f(a) = a$. Find the set of fixed points for each of the following functions.

a. $f : A \to A$where $f(x) = x$.

b. $f : N \to N$ where $f(x) = x + 1$.

c. $f : N_6 \to N_6$ where $f6$.

d. $f : N_6 \to N_6$ where $f(x) = 3x \bmod 6$.

Proofs and Challenges

14. Prove each of the following statements about floor and ceiling.

a. $\lfloor x + 1\rfloor = \lfloor x\rfloor + 1$.

b. $\lceil x - 1\rceil = \lceil x\rceil - 1$.

c. $\lceil x\rceil = \lfloor x\rfloor$ if and only if $x \in Z$.

d. $\lceil x\rceil = \lfloor x\rfloor + 1$ if and only if $x \in Z$.

15. Use the definition of the logarithm function to prove each of the following facts.

a. $\log_b 1 = 0$.

b. $\log_b b = 1$.

c. $\log_b (b^x) = x$.

d. $\log_b (x\ y) = \log_b x + \log_b y$.

e. $\log_b(x^y) = y\log_b x$.

f. $\log_b(x/y) = \log_b x - \log_b y$.

g. $\log_a x = (\log_a b)\ (\log_b x)$. (change of base)

16. Prove each of the following facts about greatest common divisors.

a. $\gcd(a, b) = \gcd(b, a) = \gcd(a, -b)$.
b. $\gcd(a, b) = \gcd(b, a - bq)$ for any integer q.
c. If $d \,|ab$ and $\gcd(d, a) = 1$, then $d\,|b$. *Hint:* Use (2.2c).

17. Given the result of the division algorithm $a = bq + r$, where $0 \le r < |b|$, prove the following statement:

$$\text{If } b < 0 \text{ then } r = a - b\lceil a/b\rceil.$$

18. Let $f : A \to B$ be a function, and let E and F be subsets of A. Prove each of the following facts about images.

a. $f(E \cup F) = f(E) \cup f(F)$.
b. $f(E \cap F) \subset f(E) \cap f(F)$.
c. Find an example to show that part (b) can be a proper subset.

19. Let $f : A \to B$ be a function, and let G and H be subsets of B. Prove each of the following facts about pre-images.

a. $f^{-1}(G \cup H) = f^{-1}(G) \cup f^{-1}(H)$.
b. $f^{-1}(G \cap H) = f^{-1}(G) \cap f^{-1}(H)$.
c. $E \subset f^{-1}(f(E))$.
d. $f(f^{-1}(G)) \subset G$.
e. Find examples to show that parts (c) and (d) can be a proper subsets.

20. Prove each of the following properties of the mod function. *Hint:* Use (2.4a) for parts (a) and (b), and use (2.2c) and parts (a) and (b) for part (c).

a. $(x + y) \bmod n = ((x \bmod n) + (y \bmod n)) \bmod n$.
b. $(xy) \bmod n = ((x \bmod n)(y \bmod n)) \bmod n$.
c. If $\gcd(a, n) = 1$, then there is an integer b such that $1 = ab \bmod n$.

21. We'll start a proof of (2.2c): If $g = \gcd(a, b)$, then there are integers x and y such that $g = ax + by$. Proof: Let $S = \{ax + by \mid x, y \in \text{Z and } ax + by > 0\}$ and let d be the smallest number in S. Then there are integers x and y such that $d = ax + by$. The idea is to show that $g = d$. Since $g\,|a$ and $g\,|b$, it follows from (1.1b) that $g_j d$. So $g \le d$. If we can show that $d\,|a$ and $d\,|b$, then we must conclude that $d = g$ because g is the greatest common divisor of a and b. Finish the proof by showing that $d_j a$ and $d_j b$. *Hint:* To show $d\,|a$, write $a = dq + r$, where $0 \le r < d$. Argue that r must be 0.

2.2 Constructing Functions

We often construct a new function by combining other simpler functions in some way. The combining method that we'll discuss in this section is called composition. We'll see that it is a powerful tool to create new functions. We'll also introduce the map function as a useful tool for displaying a list of values of a function. Many programming systems and languages rely on the ideas of this section.

2.2.1 Composition of Functions

Composition of functions is a natural process that we often use without even thinking. For example, the expression floor($\log_2(5)$) involves the composition of the two functions floor and $\log_2$. To evaluate the expression, we first evaluate $\log_2(5)$, which is a number between 2 and 3. Then we apply the floor function to this number, obtaining the value 2.

Definition of Composition

The *composition* of two functions f and g is the function denoted by $f \circ g$ and defined by

$$(f \circ g)(x) = f(g(x)).$$

Notice that composition makes sense only for values of x in the domain of g such that $g(x)$ is in the domain of f. So if $g : A \rightarrow B$ and $f : C \rightarrow D$ and $B \subset C$, then the composition $f \circ g$ makes sense. In other words, for every $x \in A$ it follows that $g(x) \in B$, and since $B \subset C$ it follows that $f(g(x)) \in D$. It also follows that $f \circ g : A \rightarrow D$.

For example, we'll consider the floor and $\log_2$ functions. These functions have types

$$\log_2 : R^+ \rightarrow R \text{ and floor} : R \rightarrow Z,$$

where R^+ denotes the set of positive real numbers. So for any positve real number x, the expression $\log_2(x)$ is a real number and thus floor($\log_2(x)$) is an integer. So the composition floor $\circ \log_2$ is defined and

$$\text{floor} \circ \log_2 : R^+ \rightarrow Z.$$

Composition of functions is associative. In other words, if f, g, and h are functions of the right type such that $(f \circ g) \circ h$ and $f \circ (g \circ h)$ make sense, then

$$(f \circ g) \circ h = f \circ (g \circ h).$$

This is easy to establish by noticing that the two expressions $((f \circ g) \circ h)(x)$ and $(f \circ (g \circ h))(x)$ are equal:

$$((f \circ g) \circ h)(x) = (f \circ g)(h(x)) = f(g(h(x))).$$
$$(f \circ (g \circ h))(x) = f((g \circ h)(x)) = f(g(h(x))).$$

So we can feel free to write the composition of three or more functions without the use of parentheses.

But we should observe that composition is usually not a commutative operation. For example, suppose that f and g are defined by $f(x) = x + 1$ and $g(x) = x^2$. To show that $f \circ g \neq g \circ f$, we only need to find one number x such that $(f \circ g)(x) \neq (g \circ f)(x)$. We'll try $x = 5$ and observe that

$$(f \circ g)(5) = f(g(5)) = f(5_2) = 5^2 + 1 = 26.$$
$$(g \circ f)(5) = g(f(5)) = g(5 + 1) = (5 + 1^2 = 36.$$

A function that always returns its argument is called an *identity* function. For a set A we sometimes write “id_A” to denote the identity function defined by $\text{id}_A(a) = a$ for all $a \in A$. If $f : A \to B$, then we always have the following equation.

$$f \circ \text{id}_A = f = \text{id}_B \circ f.$$

The Sequence, Distribute, and Pairs Functions

We'll describe here three functions that are quite useful as basic tools to construct more complicated functions that involve lists.

The *sequence* function “seq” has type $N \to \text{lists}(N)$ and is defined as follows for any natural number n:

$$\text{seq}(n) = \langle 0, 1, \ldots, n \rangle.$$

For example, $\text{seq}(0) = \langle 0 \rangle$ and $\text{seq}(4) = \langle 0, 1, 2, 3, 4 \rangle$.

The *distribute* function “dist” has type $A \times \text{lists}(B) \to \text{lists}(A \times B)$. It takes an element x from A and a list y from $\text{lists}(B)$ and returns the list of pairs made up by pairing x with each element of y. For example,

$$\text{dist}(x, \langle r, s, t \rangle) = \langle (x, r), (x, s), (x, t) \rangle.$$

The *pairs* function takes two lists of equal length and returns the list of pairs of corresponding elements. For example,

$$\text{pairs}(\langle a, b, c \rangle, \langle d, e, f \rangle) = \langle (a, d), (b, e), (c, f) \rangle.$$

Since the domain of pairs is a proper subset of $\text{lists}(A) \times \text{lists}(B)$, it is a partial function of type $\text{lists}(A) \times \text{lists}(B) \to \text{lists}(A \times B)$.

Composing Functions with Different Arities

Composition can also occur between functions with different arities. For example, suppose we define the following function.

$$f(x,\ y) = \text{dist}(x,\ \text{seq}(y)).$$

In this case dist has two arguments and seq has one argument. For example, we'll evaluate the expression $f(5,\ 3)$.

$$\begin{aligned} f(5,3) &= \text{dist}(5, \text{seq}(3)) \\ &= \text{dist}(5, \langle 0,1,2,3\rangle) \\ &= \langle (5,0),(5,1),(5,2),(5,3)\rangle \end{aligned}$$

In the next example we'll show that the definition $f(x,\ y) = \text{dist}(x,\ \text{seq}(y))$ is a special case of the following more general form of *composition*, where X can be replaced by any number of arguments.

$$f(X) = h(g_1(X),\ \ldots,\ g_n(X)).$$

example **2.6 Distribute a Sequence**

We'll show that the definition $f(x,\ y) = \text{dist}(x,\ \text{seq}(y))$ fits the general form of composition. To make it fit the form, we'll define the functions $\text{one}(x,\ y) = x$ and $\text{two}(x,\ y) = y$. Then we have the following representation of f.

$$\begin{aligned} f(x,y) &= \text{dist}(x, \text{seq}(Y)) \\ &= \text{dist}(\text{one}(x,y), \text{seq}(\text{two}(x,y))) \\ &= \text{dist}(\text{one}(x,y), (\text{seq} \circ \text{two})(x,y))). \end{aligned}$$

The last expression has the general form of composition

$$f(X) = h(g_1(X),\ g_2(X)),$$

where $X = (x,\ y)$, $h = \text{dist}$, $g_1 = \text{one}$, and $g_2 = \text{seq} \circ \text{two}$.

end example

example **2.7 The Max Function**

Suppose we define the function “max,” to return the maximum of two numbers as follows.

$$\max(x,\ y) = \text{if } x < y \text{ then } y \text{ else } x.$$

Then we can use max to define the function “max3,” which returns the maximum of three numbers, by the following composition.

$$\text{max3}(x,\ y,\ z) = \max(\max(x,\ y),\ z).$$

end example

We can often construct a function by first writing down an informal definition and then proceeding by stages to transform the definition into a formal one that suits our needs. For example, we might start with an informal definition of some function f such as

$$f(x) = \text{expression involving } x.$$

Now we try to transform the right side of the equality into an expression that has the degree of formality that we need. For example, we might try to reach a composition of known functions as follows:

$$\begin{aligned} f(x) &= \text{expression involving } x \\ &= \text{another expression involving } x \\ &= \ldots \\ &= g(h(x)). \end{aligned}$$

From a programming point of view, our goal would be to find an expression that involves known functions that already exist in the programming language being used. Let's do some examples to demonstrate how composition can be useful in solving problems.

example 2.8 Minimum Depth of a Binary Tree

Suppose we want to find the minimum depth of a binary tree in terms of the numbers of nodes. Figure 2.8 lists a few sample cases in which the trees are as compact as possible, which means that they have the least depth for the number of nodes. Let n denote the number of nodes. Notice that when $4 \leq n < 8$, the depth is 2. Similarly, the depth is 3 whenever $8 \leq n < 16$.

At the same time we know that $\log_2(4) = 2$, $\log_2(8) = 3$, and for $4 \leq n < 8$ we have $2 \leq \log_2(n) < 3$. So $\log_2(n)$ almost works as the depth function. The problem is that the depth must be exactly 2 whenever $4 \leq n < 8$. Can we make this happen? Sure—just apply the floor function to $\log_2(n)$ to get $\text{floor}(\log_2(n)) = 2$ if $4 \leq n < 8$. This idea extends to the other intervals that make up N. For example, if $8 \leq n < 16$, then $\text{floor}(\log_2(n)) = 3$.

So it makes sense to define our minimum depth function as the composition of the floor function and the $\log_2$ function:

$$\text{minDepth}(n) = \text{floor}(\log_2(n)).$$

end example

example 2.9 A List of Pairs

Suppose we want to construct a definition for the following function in terms of known functions.

$$f(n) = \langle(0, 0), (1, 1), \ldots, (n, n)\rangle \qquad \text{for any } n \in N.$$

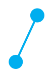
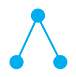
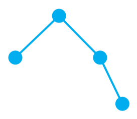
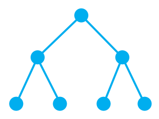
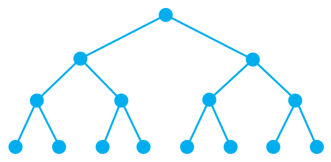
Binary tree	Nodes	Depth
	1	0
	2	1
	3	1
	4	2
	7	2
	15	3

Figure 2.8 Compact binary trees.

Starting with this informal definition, we'll transform it into a composition of known functions.

$$\begin{aligned} f(n) &= \langle (0,0), (1,1), \ldots, (n,n) \rangle \\ &= \text{pairs}(\langle 0, 1, \ldots, n \rangle, \langle 0, 1, \ldots, n \rangle) \\ &= \text{pairs}(\text{seq}(n), \text{seq}(n)). \end{aligned}$$

Can you figure out the type of f?

end example

example **2.10 Another List of Pairs**

Suppose we want to construct a definition for the following function in terms of known functions.

$$g(k) = \langle (k, 0), (k, 1), \ldots, (k, k) \rangle \qquad \text{for any } k \in N.$$

Starting with this informal definition, we'll transform it into a composition of known functions.

$$\begin{aligned} g(k) &= \langle (k,0),(k,1),\ldots,(k,k)\rangle \\ &= \mathrm{dist}(k,\langle 0,1,\ldots,k\rangle) \\ &= \mathrm{dist}(k,\mathrm{seq}(k)). \end{aligned}$$

Can you figure out the type of g?

end example

2.2.2 The Map Function

We sometimes need to compute a list of values of a function. A useful tool to accomplish this is the *map* function. It takes a function $f : A \to B$ and a list of elements from A and it returns the list of elements from B constructed by applying f to each element of the given list from A. Here is the definition.

Definition of the Map Function

Let f be a function with domain A and let $\langle x_1, \ldots, x_n\rangle$ be a list of elements from A. Then

$$\mathrm{map}(f, \langle x_1, \ldots, x_n\rangle) = \langle f(x_1), \ldots, f(x_n)\rangle.$$

So the type of the map function can be written as

$$\mathrm{map}\colon\ (A \to B) \times \mathrm{lists}(A) \to \mathrm{lists}(B).$$

Here are some example calculations.

$$\begin{aligned} &\mathrm{map}(\mathrm{floor}, \langle -1,5,-0.5,0.5,1.5,2.5\rangle) \\ &\qquad = \langle \mathrm{floor}(-1.5), \mathrm{floor}(-0.5), \mathrm{floor}(0.5), \mathrm{floor}(1.5), \mathrm{floor}(2.5)\rangle \\ &\qquad = \langle -2,-1,0,1,2\rangle. \\ &\mathrm{map}(\mathrm{floor} \circ \log_2, \langle 2,3,4,5\rangle) \\ &\qquad = \langle \mathrm{floor}(\log_2(2)), \mathrm{floor}(\log_2(3)), \mathrm{floor}(\log_2(4)), \mathrm{floor}(\log_2(5))\rangle \\ &\qquad = \langle 1,1,2,2\rangle. \\ &\mathrm{map}{+}, \langle (1,2),(3,4),(5,6),(7,8),(9,10)\rangle \\ &\qquad = \langle +(1,2), +(3,4), +(5,6), +(7,8), +(9,10)\rangle. \\ &\qquad = \langle 3,7,11,15,19\rangle. \end{aligned}$$

The map function is an example of a *higher-order* function, which is any function that either has a function as an argument has a function as a value. This is an important property that most good programming languages possess.

The composition and tupling operations are examples of functions that take other functions as arguments and return functions as results.

example 2.11 A List of Squares

Suppose we want to compute sequences of squares of natural numbers, such as 0, 1, 4, 9, 16. In other words, we want to compute $f : N \rightarrow \text{lists}(N)$ defined by $f(n) = \langle 0, 1, 4, \ldots, n^2\rangle$. We'll present two solutions. For the first solution we'll define $s(x) = x * x$ and then construct a definition for f in terms of map, s, and seq as follows.

$$\begin{aligned} f(n) &= \langle 0, 1, 4, \ldots, n^2\rangle \\ &= \langle (0), s(1), s(2), \ldots, s(n)\rangle \\ &= \text{map}(s, \langle 0, 1, 4, \ldots, n\rangle) \\ &= \text{map}(s, \text{seq}(n)). \end{aligned}$$

For the second solution we'll construct a definition for f without using the function s that we defined for the first solution.

$$\begin{aligned} f(n) &= \langle 0, 1, 4, \ldots, n^2\rangle \\ &= \langle 0 * 0, 1 * 1, 2 * 2, \ldots, n * n \\ &= \text{map}(\text{*}, \langle (0,0), (1,1), 2,2), \ldots, (n,n)\rangle) \\ &= \text{map}(*, \text{pairs}(\langle 0, 1, 2, \ldots, n\rangle, \langle 0, 1, 2, \ldots, n\rangle)) \\ &= \text{map}(*, \text{pairs}(\text{seq}(n), \text{seq}(n))). \end{aligned}$$

end example

example 2.12 Graphing with Map

Suppose we have a function f defined on the closed interval $[a, b]$ and we have a list of numbers $\langle x_0, \ldots, x_n\rangle$ that form a regular partition of $[a, b]$. We want to find the following sequence of $n + 1$ points:

$$\langle (x_0, f(x_0)), \ldots, (x_n, f(x_n))\rangle.$$

The partition is defined by $x_i = a + dk$ for $0 \leq k \leq n$, where $d = (b - a)/n$. So the sequence is a function of a, d, and n. If we can somehow create the two lists $\langle x_0, \ldots, x_n\rangle$ and $\langle f(x_0), \ldots, f(x_n)\rangle$, then the desired sequence of points can be obtained by applying the pairs function to these two sequences.

Let “makeSeq” be the function that returns the list $\langle x_0, \ldots, x_n\rangle$. We'll start by trying to define makeSeq in terms of functions that are already at hand. First we write down the desired value of the expression, makeSeq(a, d, n) and then try to gradually transform the value into an expression involving known functions and the arguments a, d, and n.

$$\begin{aligned}
&\text{makeSeq}(a, d, n)\\
&\qquad = \langle x_0, x_1, \ldots , x_n\rangle\\
&\qquad = \langle a, a + d, a + 2d, \ldots , a + nd\rangle\\
&\qquad = \text{map}(+, \langle (a, 0), (a, d), (a, 2d), \ldots , (a, nd)\rangle)\\
&\qquad = \text{map}(+, \text{dist}(a, \langle 0, d, 2d, \ldots , nd\rangle))\\
&\qquad = \text{map}(+, \text{dist}(a, \text{map}(*, \langle (d, 0), (d, 1), (d, 2), \ldots , (d, n)\rangle)))\\
&\qquad = \text{map}(+, \text{dist}(a, \text{map}(*, \text{dist}(d, \langle 0, 1, 2, \ldots , n\rangle))))\\
&\qquad = \text{map}(+, \text{dist}(a, \text{map}(*, \text{dist}(d, \text{seq}(n)))))
\end{aligned}$$

The last expression contains only known functions and the arguments a, d, and n. So we have a definition for makeSeq. Now it's an easy matter to build the second list. Just notice that

$$\begin{aligned}
\langle f(x_1), \ldots , f(x_n)\rangle &= \text{map}(f, \langle x_0, x_1, \ldots , x_n\rangle)\\
&= \text{map}(f, \text{makeSeq}(a, d, n)).
\end{aligned}$$

Now let "makeGraph" be the name of the function that returns the desired sequence of points. Then makeGraph can be written as follows:

$$\begin{aligned}
\text{makeGraph}(f, a, d, n) &= \langle (x_0, f(x_0)), \ldots , (x_n, f(x_n))\rangle\\
&= \text{pairs}(\text{makeSeq}(a, d, n), \text{map}(f, \text{makeSeq}(a, d, n))).
\end{aligned}$$

This gives us a definition of makeGraph in terms of known functions and the variables f, a, d, and n.

end example

From the programming point of view there are many other interesting ways to combine functions. But they will take us too far afield. The primary purpose now is to get a feel for what a function is and to grasp the idea of building a function from other functions by composition.

Exercises

Composing Functions

1. Evaluate each of the following expressions.

a. floor($\log_2 17$).

b. ceiling($\log_2 25$).

c. gcd(14 mod 6, 18 mod 7).
d. gcd(12, 18) mod 5.
e. dist(4, seq(3)).
f. pairs(seq(3), seq(3)).
g. dist(+, pairs(seq(2), seq(2))).

2. In each case find the compositions $f \circ g$ and $g \circ f$, and find an integer x such that $f(g(x)) \neq g(f(x))$.

a. $f(x) = \text{ceiling}(x/2)$ and $g(x) = 2x$.
b. $f(x) = \text{floor}(x/2)$ and $g(x) = 2x + 1$.
c. $f(x) = \text{gcd}(x, 10)$ and $g(x) = x \bmod 5$.

3. Let $f(x) = \mathrm{x}^2$ and $g(x, y) = x + y$. Find compositions that use the functions f and g for each of the following expressions.

a. $(x + y)^2$. b. $x^2 + y^2$. **c.** $(x + y + z)^2$. d. $x^2 + y^2 + z^2$.

4. Describe the set of natural numbers x satisfying each equation.

a. $\text{floor}(\log_2(x)) = 7$.
b. $\text{ceiling}(\log_2(x)) = 7$.

5. Find a definition for the function max4 that calculates the maximum value of four numbers. Use only composition and the function max that returns the maximum value of two numbers.

6. Find a formula for the number of binary digits in the binary representation of a nonzero natural number x. *Hint:* Notice, for example, that the numbers from 4 through 7 require three binary digits, while the numbers 8 through 15 require five binary digits, and so on.

Composing with the Map Function

7. Evaluate each expression:

a. $\text{map}(\text{floor} \circ \log_2, \langle 1, 2, 3, \ldots, 16\rangle)$.
b. $\text{map}(\text{ceiling} \circ \log_2, \langle 1, 2, 3, \ldots, 16\rangle)$.

8. Suppose that $f : N \to \text{lists}(N)$ is defined by $f(n) = \langle 0, 2, 4, 6, \ldots, 2n\rangle$. For example, $f(5) = \langle 0, 2, 4, 6, 8, 10\rangle$. In each case find a definition for f as a composition of the listed functions.

a. map, +, pairs, seq.
b. map, $*$, dist, seq.

9. For each of the following functions, construct a definition of the function as a composition of known functions. Assume that all of the variables are natural numbers.

a. $f(n, k) = \langle n, n + 1, n + 2, \ldots, n + \mathrm{k}\rangle$.
b. $f(n, k) = \langle 0, k, 2k, 3k, \ldots, nk\rangle$.
c. $f(n, m) = \langle n, n + 1, n + 2, \ldots, m - 1, m\rangle$, where $n \leq m$.
d. $f(n) = \langle n, n - 1, n - 2, \ldots, 1, 0\rangle$.
e. $f(n) = \langle (0, n), (1, n - 1), \ldots, (n - 1, 1), (n, 0)\rangle$.
f. $f(n) = \langle 1, 3, 5, \ldots, 2n + 1\rangle$.
g. $f(g, n) = \langle (0, g(0)), (1, g(1)), \ldots, (n , g(n))\rangle$.
h. $f(g, \langle x_1, x_2, \ldots, x_n\rangle) = \langle (x_1, g(x_1)), (x_2, g(x_2)), \ldots, (x_n, g(x_n))\rangle$.
i. $f(g, h, \langle x_1, \ldots, x_n\rangle) = \langle (g(x_1), h(x_1)), \ldots, (g(x_n), h(x_n))\rangle$.

10. We defined $\text{seq}(n) = \langle 0, 1, 2, 3, \ldots, n\rangle$. Suppose we want the sequence to start with the number one. In other words, we want to define a function $f(n) = \langle 1, 2, 3, \ldots, n\rangle$. Find a definition for f as a composition of the functions map, +, dist, and seq.

Proofs

11. Prove each of the following statements.

a. $\text{floor}(\text{ceiling}(x)) = \text{ceiling}(x)$.
b. $\text{ceiling}(\text{floor}(x)) = \text{floor}(x)$.
c. $\text{floor}(\log_2(x)) = \text{floor}(\log_2(\text{floor}(x)))$ for $x \geq 1$.

2.3 Properties Of Functions

Functions that satisfy one or both of two special properties can be very useful in solving a variety of computational problems. One property is that distinct elements map to distinct elements. The other property is that the range is equal to the codomain. We'll discuss these properties in more detail and give some examples where they are useful.

Injective Functions

A function $f : A \to B$ is called *injective* (also *one-to-one*, or an *embedding*) if it maps distinct elements of A to distinct elements of B. Another way to say this is that f is injective if $x \neq y$ implies $f(x) \neq f(y)$. Yet another way to say this is that f is injective if $f(x) = f(y)$ implies $x = y$. An injective function is called an *injection*.

For example, Figure 2.9 illustrates an injection from a set A to a B.

Surjective Functions

A function $f : A \to B$ is called *surjective* (also *onto*) if the range of f is the codomain B. Another way to say this is that f is surjective if each element

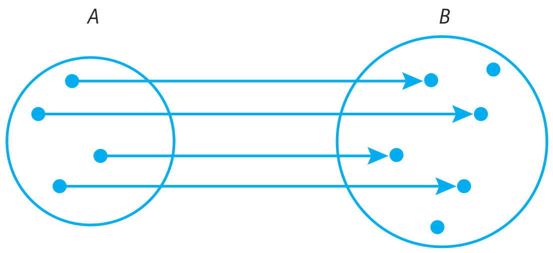


Figure 2.9 An injection.

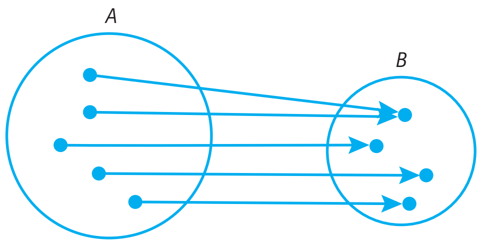


Figure 2.10 A surjection.

$b \in B$ can be written as $b = f(x)$ for some element $x \in A$. A surjective function is called a *surjection*.

For example, Figure 2.10 pictures a surjection from A to B.

example 2.13 Injective or Surjective

We'll give a few examples of functions that have one or the other of the injective and surjective properties.

1. The function $f : R \to Z$ defined by $f(x) = \lceil x + 1 \rceil$ is surjective because for any $y \in Z$ there is a number in R, namely $y - 1$, such that $f(y - 1) = y$. But f is not injective because, for example, $f(3.5) = f(3)$.

2. The function $f : N_8 \to N_8$ defined by $f(x) = 2x \bmod 8$ is not injective because, for example, $f(0) = f(4)$. f is not surjective because the range of f is only the set $\{0, 2, 4, 6\}$.

3. Let $g : N \to N \times N$ be defined by $g(x) = (x, x)$. Then g is injective because if $x, y \in N$ and $x \neq y$, then $g(x) = (x, x) \neq (y, y) = g(y)$. But g is not surjective because, for example, nothing maps to $(0, 1)$.

4. The function $f : N \times N \to N$ defined by $f(x, y) = 2x + y$ is surjective. To see this, notice that any $z \in N$ is either even or odd. If z is even, then $z = 2k$ for some $k \in N$, so $f(k, 0) = z$. If z is odd, then $z = 2k + 1$ for some

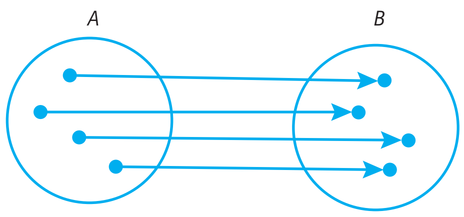


Figure 2.11 A bijection.

$k \in N$, so $f(k, 1) = z$. Thus f is surjective. But f is not injective because, for example, $f(0, 2) = f(1, 0)$.

end example

Bijections

A function is called *bijective* if it is both injective and surjective. Another term for bijective is "one-to-one and onto." A bijective function is called a *bijection* or a "one-to-one correspondence."

For example, Figure 2.11 pictures a bijection from A to B.

example 2.14 A Bijection

Let $(0, 1) = \{x \in R \mid 0 < x < 1\}$ and let R^+ denote the set of positive real numbers. We'll show that the function $f : (0, 1) \to R^+$ defined by

$$f(x) = \frac{x}{1-x}$$

is a bijection. To show that f is an injection, let $f(x) = f(y)$. Then

$$\frac{x}{1-x} = \frac{y}{1-y},$$

which can be cross multiplied to get $x - xy = y - xy$. Subtract $-xy$ from both sides to get $x = y$. Thus f is injective. To show that f is surjective, let $y > 0$ and try to find $x \in (0, 1)$ such that $f(x) = y$. We'll solve the equation

$$\frac{x}{1-x} = y.$$

Cross multiply and solve for x to obtain

$$x = \frac{y}{y+1}.$$

It follows that $f(y/(y+1)) = y$, and since $y > 0$, it follows that $0 < y/(y+1) < 1$. Thus f is surjective. Therefore, f is a bijection.

end example

Inverse Functions

Bijections always come in pairs. If $f : A \to B$ is a bijection, then there is a function $g : B \to A$, called the *inverse* of f, defined by $g(b) = a$ if $f(a) = b$. Of course, the inverse of f is also a bijection and we have $g(f(a)) = a$ for all $a \in A$ and $f(g(b)) = b$ for all $b \in B$. In other words, $g \circ f = \mathrm{id}_A$ and $f \circ g = \mathrm{id}_B$.

We should note that there is exactly one inverse of any bijection f. For suppose that g and h are two inverses of f. Then for any $x \in B$ we have

$$\begin{aligned} g(x) &= g(\mathrm{id}_B(x)) \\ &= g(f(h(x))) && (\text{since } f \circ h = \mathrm{id}_B) \\ &= \mathrm{id}_A\,(h(x)) && (\text{since } g \circ f = \mathrm{id}_A) \\ &= h(x). \end{aligned}$$

This tells us that $g = h$. The inverse of f is often denoted by the symbol f^{-1}. So if f is a bijection and $f(a) = b$, then $f^{-1}(b) = a$. Notice the close relationship between the equation $f^{-1}(b) = a$ and the pre-image equation $f^{-1}(\{b\}) = \{a\}$.

example **2.15 Inverses**

We'll look at two bijective functions together with their inverses.

1. Let *Odd* and *Even* be the sets of odd and even natural numbers, respectively. The function $f : \mathit{Odd} \to \mathit{Even}$ defined by $f(x) = x - 1$ is a bijection. Check it out. The inverse of f can be defined by $f^{-1}(x) = x + 1$. Notice that $f^{-1}(f(x)) = f^{-1}(x - 1) = (x - 1) + 1 = x$.
2. The function $f : N_5 \to N_5$ defined by $f(x) = 2x \bmod 5$ is bijective because, $f(0) = 0$, $f(1) = 2$, $f(2) = 4$, $f(3) = 1$, and $f(4) = 3$. The inverse of f can be defined by $f^{-1}(x) = 3x \bmod 5$. For example, $f^{-1}(f(4)) = 3f(4) \bmod 5 = 9 \bmod 5 = 4$. Check the other values too.

end example

The fact that the function $f : N_5 \to N_5$ defined by $f(x) = 2x \bmod 5$ (in the preceding example) is a bijection is an instance of an interesting and useful fact about the mod function and inverses. Here is the result.

The Mod Function and Inverses (2.6)

Let $n > 1$ and let $f : N_n \to N_n$ be defined as follows, where a and b are integers.

$$f(x) = (ax + b) \bmod n.$$

Continued ➡

Then f is a bijection if and only if $\gcd(a, n) = 1$. When this is the case, the inverse function f^{-1} is defined by

$$f^{-1}(x) = (kx + c) \bmod n,$$

where c is an integer such that $f(c) = 0$, and k is an integer such that $1 = ak + nm$ for some integer m.

Proof: We'll prove the iff part of the statement and leave the form of the inverse as an exercise.

So assume that f is a bijection and show that $\gcd(a, n) = 1$. Then f is surjective, so there are numbers $s, c \in N_n$ such that $f(s) = 1$ and $f(c) = 0$. Using the definition of f, these equations become

$$(as + b) \bmod n = 1 \text{ and } (ac + b) \bmod n = 0.$$

Therefore, there are intgers q_1 and q_2 such that the two equations become

$$as + b + nq_1 = 1 \text{ and } ac + b + nq_2 = 0.$$

Solve the second equation for b to get $b = -ac - nq_2$, and substitute for b in the first equation to get

$$1 = a(s - c) + n(q_1 - nq_2).$$

Since $\gcd(a, n)$ divides both a and n, it divides the right side of the equation (1.1b) and therefore must also divide 1. Therefore, $\gcd(a, n) = 1$.

Now assume that $\gcd(a, n) = 1$ and show that f is a bijection. Since N_n is finite, we need only show that f is an injection to conclude that it is a bijection. So let $x, y \in N_n$ and let $f(x) = f(y)$. Then

$$(ax + b) \bmod n = (ay + b) \bmod n,$$

which by (2.4a) implies that n divides $(ax + b) - (ay + b)$. Therefore, n divides $a(x - y)$, and since $\gcd(a, n) = 1$, we conclude from (2.2d) that n divides $x - y$. But the only way for n to divide $x - y$ is for $x - y = 0$ because both $x, y \in N_n$. Thus $x = y$ and it follows that f is injective, hence also surjective, and therefore bijective. QED.

Relationships

An interesting property that relates injections and surjections is that if there is an injection from A to B, then there is a surjection from B to A, and conversely. A less surprising fact is that if the composition $f \circ g$ makes sense and if both f and g have one of the properties injective, surjective, or bijective, then $f \circ g$ also has the property. We'll list these facts for the record.

Injective and Surjective Relationships (2.7)

a. If f and g are injective, then $g \circ f$ is injective.

b. If f and g are surjective, then $g \circ f$ is surjective.

c. If f and g are bijective, then $g \circ f$ is bijective.

d. There is an injection from A to B if and only if there is a surjection from B to A.

Proof: We'll prove (2.7c) and leave the others as exercises. Suppose that f is an injection from A to B. We'll define a function g from B to A. Since f is an injection, it follows that for each $b \in \text{range}(f)$ there is exactly one $a \in A$ such that $b = f(a)$. In this case, we define $g(b) = a$. For each $b \in B - \text{range}(f)$ we have the freedom to let g map b to any element of A that we like. So g is a function from B to A and we defined g so that $\text{range}(g) = A$. Thus g is surjective.

For the other direction, assume that f is a surjection from A to B. We'll define a function g from B to A. Since g is a surjection, it follows that for each $b \in B$ the pre-image $f^{-1}(\{b\}) \neq \varnothing$. So we can pick an element $a \in f^{-1}(\{b\})$ and define $g(b) = a$. Thus g is a function from B to A. If $x, y \in B$, then $g(x) \in f^{-1}(\{x\})$ and $g(y) \in f^{-1}(\{y\})$. Thus if $x \neq y$, then $g(x) \neq g(y)$ because $f^{-1}(\{x\}) \cap f^{-1}(\{y\}) = \varnothing$. So g is injective. QED.

The Pigeonhole Principle

We're going to describe a useful rule that we often use without thinking. For example, suppose 21 pieces of mail are placed in 20 mail boxes. Then one mailbox receives at least two pieces of mail. This is an example of the *pigeonhole principle*, where we think of the pieces of mail as pigeons and the mail boxes as pigeonholes.

Pigeonhole Principle

If m pigeons fly into to n pigeonholes where $m > n$, then one pigeonhole will have two or more pigeons.

We can describe the pigeonhole principle in more formal terms as follows: If A and B are finite sets with $|A| > |B|$, then every function from A to B maps at least two elements of A to a single element of B. This is the same as saying that no function from A to B is an injection.

This simple idea is used often in many different settings. We'll be using it at several places in the book.

example

2.16 Pigeonhole Examples

Here are a few sample statements that can be justified by the pigeonhole principle.

1. The "musical chairs" game is played with n people and $n - 1$ chairs for them to sit on when the music stops.

2. In a group of eight people, two were born on the same day of the week.

3. If a six-sided die is tossed seven times, one side will come up twice.

4. If a directed graph with n vertices has a path of length n or longer, then the path must pass through some vertex at least twice. This implies that the graph contains a cycle.

5. In any set of $n + 1$ integers there are two numbers that have the same remainder on division by n. This follows because there are only n remainders possible on division by n.

6. The decimal expansion of any rational number contains a repeating sequence of digits (they might be all zeros). For example, $359/495 = 0.7252525\ldots$, $7/3 = 2.333\ldots$, and $2/5 = 0.4000\ldots$. To see this, let m/n be a rational number. Divide m by n until all the digits of m are used up. This gets us to the decimal point. Now continue the division by n for $n + 1$ more steps. This gives us $n + 1$ remainders. Since there are only n remainders possible on division by n, the pigeonhole principle tells us that one of remainders will be repeated. So the sequence of remainders between the repeated remainders will be repeated forever. This causes the corresponding sequence of digits in the decimal expansion to be repeated forever.

end example

Simple Ciphers

Bijections and inverse functions play an important role when working with systems (called ciphers) to encipher and decipher information. We'll give a few examples to illustrate the connections. For ease of discussion we'll denote the 26 letters of the lowercase alphabet by the set $N_{26} = \{0, 1, 2, \ldots, 25\}$, where we identify a with 0, b with 1, and so on.

To get things started we'll describe a cipher to transform a string of text by means of a simple translation of the characters. For example, the message 'abc' translated by 5 letters becomes 'fgh'. The cipher is easy to write once we figure out how wrap around the end of the alphabet. For example, to translate the letter z (i.e., 25) by 5 letters we need to come up with the letter e (i.e., 4). All we need to do is add the two numbers mod 26:

$$(25 + 5) \bmod 26 = 4.$$

So we can define a cipher f to translate any letter by 5 letters as

$$f(x) = (x + 5) \bmod 26.$$

Is f a bijection? Yes, because f has type $N_{26} \to N_{26}$. So we have a cipher for transforming letters. For example, the message ‘hello’ transforms to ‘mjqqt’. To decode the message we need to reverse the process, which is easy in this case because the inverse of f is easy to guess.

$$f^{-1}(x) = (x - 5) \bmod 26.$$

For example, to see that e maps back to z, we can observe that 4 maps to 25.

$$f^{-1}(4) = (4 - 5) \bmod 26 = (-1) \bmod 26 = 25.$$

The cipher we’ve been talking about is called an *additive* cipher. A cryptanalyst who intercepts the message ‘mjqqt’ can easily check whether it was created by an additive cipher. An additive cipher is an example of a *monoalphabetic* cipher, which is a cipher that always replaces any character of the alphabet by the same character from the cipher alphabet.

A *multiplicative* cipher is a monoalphabetic cipher that translates each letter by using a multiplier. For example, suppose we define the cipher

$$g(x) = 3x \bmod 26.$$

For example, this cipher maps a to a, c to g, and m to k. Is g a bijection? You can convince yourself that it is by exaustive checking. But it’s easier to use (2.6). Since gcd(3, 26) = 1 it follows that g is a bijection. What abut deciphering? Again, (2.6) comes to the rescue to tell us the form of g^{-1}. Since we can write gcd(3, 26) = 1 = 3(9) + 26(–1), and since g(0) = 0, it follows that we can define g^{-1} as

$$g^{-1}(x) = 9x \bmod 26.$$

There are some questions to ask about multiplicative ciphers. Which keys act as an identity (not changing the message)? Is there always one letter that never changes no matter what the key? Do fractions work as keys? What about decoding (i.e., deciphering) a message? Do you need a new deciphering algorithm?

An *affine* cipher is a monoalphabetic cipher that translates each letter by using two kinds of translation. For example, we can start with a pair of keys (M, A) and transform a letter first applying the additive cipher with key A to get an intermediate letter. Then apply the multiplicative cipher with key M to that letter to obtain the desired letter. For example, we might use the pair of keys (5, 3) and define f as

$$f(x) = 3((x + 5) \bmod 26) \bmod 26 = (3x + 5) \bmod 26.$$

We can use (2.6) to conclude that f is a bijection because gcd(3, 26) = 1. So we can also decipher messages with f^{-1}, which we can construct using (2.6) as

$$f^{-1}(x) = (9x + 7) \bmod 26.$$

Some ciphers leave one or more letters fixed. For example, an additive cipher that translates by a multiple of 26 will leave all letters fixed. A multiplicative cipher always sends 0 to 0, so one letter is fixed. But what about an affine cipher of the form $f(x) = (ax + b) \bmod 26$? When can we be sure that no letters are fixed? In other words, when can we be sure that $f(x) \neq x$ for all $x \in N$? The answer is, when $\gcd(a - 1, 26)$ does not divide b. Here is the general result that we've been discussing.

The Mod Function and Fixed Points (2.8)

Let $n > 1$ and let $f : N_n \to N_n$ be defined as follows, where a and b are integers.

$$f(x) = (ax + b) \bmod n.$$

Then f has no fixed points (i.e., f changes every letter of an alphabet) if and only if $\gcd(a - 1, n)$ does not divide b.

This result follows from an old and easy result from number theory, and we'll discuss it in the exercises. Let's see how the result helps our cipher problem.

example 2.17 Simple Ciphers

The function $f(x) = (3x + 5) \bmod 26$ does not have any fixed points because $\gcd(3 - 1, 26) = \gcd(2, 26) = 2$, and 2 does not divide 5. It's nice to know that we don't have to check all 26 values of f.

On the other hand, the function $f(x) = (3x + 4) \bmod 26$ has fixed points because $\gcd(3 - 1, 26) = 2$ and 2 divides 4. For this example, we can observe that $f(11) = 11$ and $f(24) = 24$. So in terms of our association of letters with numbers we would have $f(l) = l$ and $f(y) = y$.

end example

Whatever cipher we use, we always have some questions: Is it a bijection? What is the range of values for the keys? Is it hard to decipher an intercepted message?

Hash Functions

Suppose we wish to retrieve some information stored in a table of size n with indexes 0, 1, . . . , $n - 1$. The items in the table can be very general things. For example, the items might be strings of letters, or they might be large records with many fields of information. To look up a table item we need a *key* to the information we desire.

For example, if the table contains records of information for the 12 months of the year, the keys might be the three-letter abbreviations for the 12 months. To look up the record for January, we would present the key Jan to a lookup program. The program uses the key to find the table entry for the January record of information. Then the information would be available to us.

An easy way to look up the January record is to search the table until the key Jan is found. This might be OK for a small table with 12 entries. But it may be impossibly slow for large tables with thousands of entries. Here is the general problem that we want to solve.

Given a key, find the table entry containing the key without searching.

This may seem impossible at first glance. But let's consider a way to use a function to map each key directly to its table location.

Definition of Hash Function

A *hash function* is a function that maps a set S of keys to a finite set of table indexes, which we'll assume are 0, 1, . . . , $n - 1$. A table whose information found by a hash function called a *hash table*.

For example, let S be the set of three-letter abreviations for the months of the year. We might define a hash function $f : S \to \{0, 1, \ldots, 11\}$ in the following way.

$$f(XYZ) = (\text{ord}(X) + \text{ord}(Y) + \text{ord}(Z)) \bmod 12.$$

where ord(X) denotes the integer value of the ASCII code for X. (The ASCII values for A to Z and a to z are 65 to 90 and 97 to 122, respectively.) For example, we'll compute the value for the key Jan.

$$\begin{aligned} f(\text{JAN}) &= (\text{ord}(X) + \text{ord}(Y) + \text{ord}(n)) \quad \text{mod } 12 \\ &= (74 + 97 + 110) \quad \text{mod } 12 \\ &= 5. \end{aligned}$$

Most programming languages have efficient implementations of the ord and mod functions, so hash functions constructed from them are quite fast. Here is the listing of all the values of f.

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
5	5	0	3	7	1	11	9	8	6	7	4

Notice the function f is not injective. For example, $f(\text{Jan}) = f(\text{Feb}) = 5$. So if we use f to construct a hash table, we can't put the information for January and February at the same address. Let's discuss this problem.

Collisions

If a hash function is injective, then it maps every key to the index of the hash table where the information is stored and no searching is involved. Often this is not possible. When two keys map to the same table index, the result is called a *collision*. So if a hash function is not injective, it has collisions. Our example hash function has collisions $f(\text{Jan}) = f(\text{Feb})$ and $f(\text{May}) = f(\text{Nov})$.

When collisions occur, we store the information for one of the keys in the common table location and must find some other location for the other keys. There are many ways to find the location for a key that has collided with another key. One technique is called *linear probing*. With this technique the program searches the remaining locations in a "linear" manner.

For example, if location k is the collision index, then the following sequence of table locations is searched

$$(k + 1) \bmod n,\ (k + 2) \bmod n,\ \ldots,\ (k + n) \bmod n.$$

In constructing the table in the first place, these locations would be searched to find the first open table entry. Then the key would be placed in that location.

example

2.18 A Hash Table

We'll use the sample hash function f to construct a hash table for the months of the year by placing the three-letter abreviations in the table one by one, starting with Jan and continuing to Dec. We'll use linear probing to resolve collisions that occur in the process. For example, since $f(\text{Jan}) = 5$, we place Jan in position 5 of the table. Next, since $f(\text{Feb}) = 5$ and since postition 5 is full, we look for the next available position and place Feb in postition 6. Continuing in this way, we eventually construct the following hash table, where entries in parentheses need some searching to be found.

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
5	5	0	3	7	1	11	9	8	6	7	4

There are many questions. Can we find an injection so there are no collisions? If we increased the size of the table, would it give us a better chance of finding an injection? If the table size is increased, can we scatter the elements so that collisions can be searched for in less time?

end example

Probe Sequences

Linear probing that looks at locations one step at a time may not be the best way to resolve collisions for some kinds of keys. An alternative is to try linear probing with a "gap" between table locations in order to "scatter" or "hash" the information to different parts of the table. The idea is to keep the number of searches to a minimum. Let g be a gap, where $1 \leq g < n$. Then the following sequence of table locations is searched in case a collision occurs at location k:

$$(k + g) \bmod n,\ (k + 2g) \bmod n,\ \ldots,\ (k + ng) \bmod n.$$

Some problems can occur if we're not careful with our choice of g. For example, suppose $n = 12$ and $g = 4$. Then the probe sequence can skip some table entries. For example, if $k = 7$, the above sequence becomes

$$11, 3, 7, 11, 3, 7, 11, 3, 7, 11, 3, 7.$$

So we would miss table entries 0, 1, 2, 4, 5, 6, 8, 9, and 10. Let's try another value for g. Suppose we try $g = 5$. Then we obtain the following probe sequence starting at $k = 7$:

$$0, 5, 10, 3, 8, 1, 6, 11, 4, 9, 2, 7.$$

In this case we cover the entire set $\{0, 1, \ldots, 11\}$. In other words, we've defined a bijection $f : \mathrm{N}_{12} \rightarrow \mathrm{N}_{12}$ by $f(x) = 5x \bmod 12$. Can we always find a probe sequence that hits all the elements of $\{0, 1, \ldots, n - 1\}$? Happily, the answer is yes. Just pick g and n so that they are relatively prime, $\gcd(g, n) = 1$. For example, if we pick n to be a prime number, then $(g, n) = 1$ for any g in the interval $1 \leq g < n$. That's why table sizes are often prime numbers, even though the data set may have less entries than the table size.

There are many ways to define hash functions and to resolve collisions. The paper by Cichelli [1980] examines some bijective hash functions.

Exercises

Injections, Surjections, and Bijections

1. The fatherOf function from *People* to *People* is neither injective nor surjective. Why?

2. For each of the following cases, construct a function satisfying the given condition, where the domain and codomain are chosen from the sets

$$A = \{a, b, c\},\ B = \{x, y, z\},\ C = \{1, 2\}.$$

a. Injective but not surjective.

b. Surjective but not injective.

c. Bijective.

d. Neither injective nor surjective.

3. For each of the following types, compile some statistics: the number of functions of that type; the number that are injective; the number that are surjective; the number that are bijective; the number that are neither injective, surjective, nor bijective.

a. $\{a, b, c\} \to \{1, 2\}$.
b. $\{a, b\} \to \{1, 2, 3\}$.
c. $\{a, b, c\} \to \{1, 2, 3\}$.

4. Show that each function $f : N \to N$ has the listed properties.

a. $f(x) = 2x$.	(injective and not surjective)
b. $f(x) = x + 1$.	(injective and not surjective)
c. $f(x) = \text{floor}(x/2)$.	(surjective and not injective)
d. $f(x) = \text{ceiling}(\log_2 (x + 1))$.	(surjective and not injective)
e. $f(x) =$ if x is odd then $x - 1$ else $x + 1$.	(bijective)

5. For each of the following functions, state which of the properties injective and surjective holds.

a. $f : R \to Z$, where $f(x) = \text{floor}(x)$.
b. $f : N \to N$, where $f(x) = x \bmod 10$.
c. $f : Z \to N$ defined by $f(x) = |x + 1|$.
d. seq : $N \to \text{lists}(N)$.
e. dist : $A \times \text{lists}(B \to \text{lists}(A \times B\ )$.
f. $f : A \to \text{power}(A)$, A is any set, and $f(x) = \{x\}$.
g. $f : \text{lists}(A) \to \text{power}(A)$, A is finite, and $f(\langle x_1, \ldots, x_n\rangle) = \{x_1, \ldots, x_n\}$.
h. $f : \text{lists}(A) \to \text{bags}(A)$, A is finite, and $f(\langle x_1, \ldots, x_n\rangle) = [x_1, \ldots, x_n]$.
i. $f : \text{bags}(A) \to \text{power}(A)$, A is finite, and $f([x_1, \ldots, x_n]) = \{x_1, \ldots, x_n\}$.

6. Let R^+ and R^- denote the sets of positive and negative real numbers, respectively. If $a, b \in R$ and $a < b$, let $(a, b) = \{x \in R \mid a < x < b\}$. Show that each of the following functions is a bijection.

a. $f : (0, 1) \to (a, b)$ defined by $f(x) = (b - a)x + a$.
b. $f : R^+ \to (0, 1)$ defined by $f(x) = 1/(x + 1)$.
c. $f : (1/2, 1) \to R^+$ defined by $f(x) = 1/(2x - 1) - 1$.
d. $f : (0, 1/2) \to R–$ defined by $f(x) = 1/(2x - 1) + 1$
e. $f : (0, 1) \to R$ defined by $f(x) = \begin{cases} 1/(2x-1)-1 & \text{if } 1/2 < x < 1 \\ 0 & \text{if } x = 1/2 \\ 1/(2x-1)+1 & \text{if } 0 < x < 1/2 \end{cases}$.

The Pigeonhole Principle

7. Use the pigeonhole principle for each of the following statements.

a. How many people are needed in a group to say that three were born on the same day of the week?

b. How many people are needed in a group to say that four were born in the same month?

c. Why does any set of 10 nonempty strings over $\{a, b, c\}$ contain two different strings whose starting letters agree and whose ending letters agree?

d. Find the size needed for a set of nonempty strings over $\{a, b, c, d\}$ to contain two strings whose starting letters agree and whose ending letters agree.

8. Use the pigeonhole principle to verify each of the following statements.

a. In any set of 11 natural numbers there are two numbers whose decimal representations contain a common digit.

b. In any set of four numbers picked from the set $\{1, 2, 3, 4, 5, 6\}$ there are two numbers whose sum is seven.

c. If five distinct numbers are chosen from the set $\{1, 2, 3, 4, 5, 6, 7, 8\}$, then two of the numbers chosen are consecutive (i.e., of the form n and $n + 1$). *Hint:* List the five numbers chosen as, x_1, x_2, x_3, x_4, x_5 and list the successors as $x_1 + 1$, $x_2 + 1$, $x_3 + 1$, $x_4 + 1$, $x_5 + 1$. Are there more than eight numbers listed?

Simple Ciphers and the Mod Function

9. Each of the following functions has the form $f(x) = (ax + b) \bmod n$. Assume that each function has type $\mathrm{N}_n \to \mathrm{N}_n$, so that we can think of f as a cipher for an alphabet represented by the numbers $0, 1, \ldots, n - 1$. Use (2.6) to determine whether each function is a bijection, and, if so, construct its inverse. Then use (2.8) to determine whether the function has fixed points (i.e., letters that don't change), and, if so, find them.

a. $f(x) = 2x \bmod 6$.

b. $f(x) = 2x \bmod 5$.

c. $f(x) = 5x \bmod 6$.

d. $f(x) = (3x + 2) \bmod 6$.

e. $f(x) = (2x + 3) \bmod 7$.

f. $f(x) = (5x + 3) \bmod 12$.

g. $f(x) = (25x + 7) \bmod 16$.

10. Think of the letters A to Z as the numbers numbers 0 to 26 and let f be a cipher of the form $f(x) = (ax + b) \bmod 26$.

a. Use (2.6) to find all values of a ($0 \leq a < 26$) that will make f bijective.

b. For the values of a in part (a) that make f bijective, use (2.8) to find a general statement about the values of b ($0 \leq b < 26$) that will ensure that f maps each letter to a different letter.

Hash Functions

11. Let $S = \{$one, two, three, four, five, six, seven, eight, nine$\}$ and let $f : S \rightarrow N_9$ be defined by $f(x) = (3|x|) \bmod 9$, where $|x|$ means the number of letters in x.. For each of the following gaps, construct a hash table that contains the strings of S by choosing a string for entry in the table by the order that it is listed in S. Resolve collisions by linear probing with the given gap and observe whether all strings can be placed in the table.

a. Gap = 1. b. Gap = 2 **c.** Gap = 3.

12. Repeat Exercise 11 for the set $S = \{$Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday$\}$ and the function f: $S \rightarrow N_7$ defined by $f(x) = (2|\mathrm{x}| + 3) \bmod 7$.

13. Repeat Exercise 11 for the set $S = \{$January, February, March, April, May, June, July, August$\}$ and f: $S \rightarrow N_7$ defined by $f(x) = (|x| + 3) \bmod 8$.

Proofs and Challenges

14. Find integers a and b such that the function $f : N_{12} \rightarrow N_{12}$ defined by $f(x) = (ax + b) \bmod 12$ is bijective and $f^{-1} = f$.

15. Let $f : A \rightarrow B$ and $g : B \rightarrow C$. Prove each of the following statements.

a. If f and g are injective, then $g \circ f$ is injective.

b. If f and g are surjective, then $g \circ f$ is surjective.

c. If f and g are bijective, then $g \circ f$ is bijective.

16. Let f and g be bijections of type $A \rightarrow A$ such that $g(f(x) = x$ for all $x \in A$. Prove that $f(g(x)) = x$ for all $x \in A$.

17. Assume that the functions f and g can be formed into a composition $g \circ f$.

a. If $g \circ f$ is surjective, what can you say about f or g ?

b. If $g \circ f$ is injective, what can you say about f or g ?

18. Let $g : A \rightarrow B$ and $h : A \rightarrow C$ and let f be defined by $f(x) = (g(x), h(x))$. Show that each of the following statements holds.

a. If f is surjective, then g and h are surjective. Find an example to show that the converse is false.

b. If g or h is injective, then f is injective. Find an example to show that the converse is false.

19. Prove that the equation $ax \bmod n = b \bmod n$ has a solution x if and only if $\gcd(a, n)$ divides b.

20. Use the result of Exercise 16 to prove (2.8): Let $n > 1$ and $f : N_n \to N_n$ be defined by $f(x) = (ax + b) \bmod n$. Then f has no fixed points if and only if $\gcd(a - 1, n)$ does not divide b.

21. Prove the second part of (2.6). In other words, assume the following facts.

$f : N_n \to N_n$ is defined by $f(x) = (ax + b) \bmod n$.
f is a bijection, which also means that $\gcd(a, n) = 1$.
c is an integer such that $f(c) = 0$.
k is an integer such that $1 = ak + nm$ for some integer m.
$g : N_n \to N_n$ is defined by $g(x) = (kx + c) \bmod n$.

Prove that $g = f^{-1}$.

2.4 Countability

Let's have a short discussion about counting sets that may not be finite. We'll have to examine what it means to count an infinite set and what it means to compare the size of infinite sets. In so doing we'll find some useful techniques that can be applied to questions in computer science. For example, we'll see as a consequence of our discussions that there are some limits on what can be computed. We'll start with some simplifying notation.

2.4.1 Comparing the Size of Sets

Let A and B be sets. If there is a bijection between A and B, we'll denote the fact by writing

$$|A| = |B|.$$

In this case we'll say that A and B have the *same size* or have the *same cardinality*, or are *equipotent*.

example **2.19 Cardinality of a Finite Set**

Let $A = \{(x + 1)^3 \mid 1 \le (x + 1)^3 \le 3000\}$. Let's find the cardinality of A. After a few calculations we can observe that

$$(0 + 1)^3 = 1,\ (1 + 1)^3 = 8,\ \ldots,\ (13 + 1)^3 = 2744 \text{ and } (14 + 1)^3 = 3375.$$

So we have a bijection $f : \{0, 1, \ldots, 13\} \to A$, where $f(x) = (x + 1)^3$. Therefore, $|A| = |\{0, 1, \ldots, 13\}| = 14$.

end example

example

2.20 Cardinality of an Infinite Set

Let Odd denote the set of odd natural numbers. Then the function $f : \mathrm{N} \rightarrow Odd$ defined by $f(x) = 2x + 1$ is a bijection. So Odd and N have the same size and we write $|Odd| = |N|$.

end example

If there is an injection from A to B, we'll denote the fact by writing

$$|A| \leq |B|.$$

In this case we'll say that the size, or cardinality, of A is *less that or the same as* that of B. Recall that there is an injection from A to B if and only if there is a surjection from B to A. So $|A| \leq |B|$ also means that there is a surjection from B to A.

If there is an injection from A to B but *no* bijection between them, we'll denote the fact by writing

$$|A| < |B|.$$

In this case we'll say that the size, or cardinality, of A *is less than* that of B. So $|A| < |B|$ means that $|A| \leq |B|$and $|A| \neq |B|$.

Sets that Are Countable

Informally, a set is countable if its elements can be counted in a step by step fashion (e.g., count one element each second), even if it takes as many seconds as there are natural numbers. Let's clarify the idea by relating sets that can be counted to subsets of the natural numbers.

If A is a finite set with n elements, then we can represent the elements of A by listing them in the following sequence:

$$x_0, x_1, x_2, \ldots, x_{n-1}.$$

If we associate each x_k with the subscript k, we get a bijection between A and the set $\{0, 1, \ldots, n - 1\}$.

Suppose A is an infinite set such that we can represent all the elements of A by listing them in the following infinite sequence:

$$x_0, x_1, x_2, \ldots, x_n, \ldots .$$

If we associate each x_k with the subscript k, we get a bijection between A and the set N of natural numbers.

Definition of Countable

The preceding descriptions give us the seeds for a definition of countable. A set is *countable* if it is finite or if there is a bijection between it and N. In the latter case, the set is said to be *countably infinite.* In terms of size we can say that a set S is countable if $|S| = |\{0, 1, \ldots, n-1\}|$ for some natural number n or $|S| = |N|$. If a set is not countable, it is said to be *uncountable*.

Countable Properties

a. Every subset of N is countable.

b. S is countable if and only if $|S| \leq |N|$.

c. Any subset of a countable set is countable.

d. Any image of a countable set is countable.

Proof. We'll prove (a) and (b) and leave (c) and (d) as exercises. Let S be a subset of N. If S is finite, then it is countable by definition. So assume that S is infinite. Now since S is a set of natural numbers, it has a smallest element that we'll represent by x_0. Next, we'll let x_1 be the smallest element of the set $S - \{x_0\}$. We'll continue in this manner, letting x_n be the smallest element of $S - \{x_0, \ldots, x_{n-1}\}$. In this way we obtain an infinite listing of the elements of S:

$$x_0,\ x_1,\ x_2,\ \ldots,\ x_n,\ \ldots.$$

Notice that each element $m \in S$ is in the listing because there are at most m elements of S that are less than m. So m must be represented as one of the elements x_0, x_1, x_2, ..., x_m in the sequence. The association x_k to k gives a bijection between S and N. So $|S| = |N|$ and thus S is countable.

(b) If S is countable then $|S| \leq |N|$ by definition. So assume that $|S| \leq |N|$. Then there is an injection $f : S \rightarrow N$. So $|S| = |f(S)|$. Since $f(S)$ is a subset of N, it is countable by (a). Therefore it is either finite or $|f(S)|{=}|N|$. So S is either finite or $|S| = |f(S)|{=}|N|$. QED

Techniques to Show Countability

An interesting and useful fact about countablity is that the set N × N is countable. We'll state it for the record.

Theorem (2.9)

$$N \times N \text{ is a countable set.}$$

Proof: We need to describe a bijection between $N \times N$ and N. We'll do this by arranging the elements of $N \times N$ in such a way that they can be easily counted. One way to do this is shown in the following listing, where each row lists a sequence of tuples in $N \times N$ followed by a corresponding sequence of natural numbers.

$$\begin{array}{lcl}
(0,0), & \longleftrightarrow & 0, \\
(0,1),(1,0), & \longleftrightarrow & 1,2 \\
(0,2),(1,1),(2,0), & \longleftrightarrow & 3,4,5, \\
\quad\vdots & \vdots & \quad\vdots \\
(0,n),\cdots & \longleftrightarrow & (n^2+n)/2,\cdots \\
\quad\vdots & \vdots & \quad\vdots
\end{array}$$

Notice that each row of the listing contains all the tuples whose components add up to the same number. For example, the sequence (0, 2), (1, 1), (2, 0) consists of all tuples whose components add up to 2. So we have a bijection between $N \times N$ and N. Therefore, $N \times N$ is countable. QED.

We should note that the bijection described in (2.9) is called *Cantor's pairing function*. It maps each pair of natural numbers (x, y) to the natural number

$$\frac{(x+y)^2+3x+y}{2}.$$

We can use (2.9) to prove the following result that the union of a countable collection of countable sets is countable.

Counting Unions of Countable Sets (2.10)

If $S_0, S_1, \ldots, S_n, \ldots$ is a sequence of countable sets, then the union

$$S_0 \cup S_1 \cup \cdots \cup S_n \cup \cdots$$

is a countable set.

Proof: Since each set is countable its elements can be indexed by natural numbers. So for each set S_n we'll list its elements as $x_{n0}, x_{n1}, x_{n2}, \ldots$. If S_n is a finite set then we'll list one of its elements repeatedly to make the listing infinite. In the same way, if there are only finitely many sets, then we'll list one of the sets repeatedly to make the sequence infinite. In this way we can associate each tuple (m, n) in $N \times N$ with an element x_{mn} in the union of the given sets. The mapping may not be a bijection since some elements of the union might be repeated in the listings. But the mapping is a surjection from $N \times N$ to the union of the sets. So, since $N \times N$ is countable, it follows that the union is countable. QED.

example **2.21 Countability of the Rationals**

We'll show that the set Q of rational numbers is countable by showing that $|Q| = |N|$. Let Q^+ denote the set of positive rational numbers. So we can represent Q^+ as the following set of fractions, where repetitions are included (e.g., 1/1 and 2/2 are both elements of the set).

$$Q^+ = \{m/n \mid m,\, n \in \mathrm{N} \text{ and } n \neq 0\}.$$

Now we'll associate each fraction m/n with the tuple $(m,\, n)$ in $N \times N$. This association is an injection, so we have $|Q^+| \leq |N \times N|$. Since $N \times N$ is countable, it follows that Q^+ is countable. In the same way, the set Q– of negative rational numbers is countable. Now we can write Q as the union of three countable sets:

$$Q = Q^+ \cup \{0\} \cup Q^-.$$

Since each set in the union is countable, it follows from (2.10) that the union of the sets is countable.

end example

Counting Strings

An important consequence of (2.10) is the following fact about the countability of the set of all strings over a finite alphabet.

Theorem (2.11)

The set A^* of all strings over a finite alphabet A is countably infinite.

Proof: For each $n \in \mathrm{N}$, let A_n be the set of strings over A having length n. It follows that A^* is the union of the sets $A_0,\, A_1,\, \ldots,\, A_n,\, \ldots$. Since each set A_n is finite, we can apply (2.10) to conclude that A^* is countable. QED.

2.4.2 Diagonalization

Let's discuss a classic construction technique, called *diagonalization*, which is quite useful in several different settings that deal with counting. The technique was introduced by Cantor when he showed that the real numbers are uncountable. Here is a general description of diagonalization.

Diagonalization (2.12)

Let A be an alphabet with two or more symbols and let $S_0,\, S_1,\, \ldots,\, S_n,\, \ldots,$ be a countable listing of sequences of the form $S_n = (a_{n0},\, a_{n1},\, \ldots,\, a_{nn},\, \ldots)$, where $a_{ni} \in A$. The sequences are listed as the rows of the following infinite matrix.

Continued ➡

$$\begin{array}{c|cccccc}
 & 0 & 1 & 2 & \cdots & n & \cdots \\
\hline
S_0 & a_{00} & a_{01} & a_{02} & \cdots & a_{0n} & \cdots \\
S_1 & a_{10} & a_{11} & a_{12} & \cdots & a_{1n} & \cdots \\
S_2 & a_{20} & a_{21} & a_{22} & \cdots & a_{2n} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
S_n & a_{n0} & a_{n1} & a_{n2} & \cdots & a_{nn} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

Then there is a sequence $S = (a_0, a_1, a_2, \ldots, a_n, \ldots)$ over A that is *not* in the original list. We can construct S from the list of diagonal elements $(a_{00}, a_{11}, a_{22}, \ldots, a_{nn}, \ldots)$ by changing each element so that $a_n \neq a_{nnv}$ for each n. Therefore S differs from each S_n at the nth element. For example, pick two elements x, $y \in A$ and define

$$a_n = \begin{cases} x & \text{if } a_{nn} = y \\ y & \text{if } a_{nn} \neq y. \end{cases}$$

Uncountable Sets

Now we're in position to give some examples of uncountable sets. We'll demonstrate the method of Cantor, which uses proof by contradiction together with the diagonalization technique.

example 2.22 Uncountability of the Reals

We'll show that the set R of real numbers is uncountable. It was shown in Exercise 7 of Section 2.3 that there is a bijection between R and the set U of real numbers between 0 and 1. So $|R| = |U|$ and we need only show that U is uncountable. Assume, by way of contradiction, that U is countable. Then we can list the numbers all the numbers between 0 and 1 as a countable sequence

$$r_0,\ r_1,\ r_2,\ \ldots,\ r_n,\ \ldots.$$

Each real number in between 0 and 1 can be represented as an infinite decimal. So for each n there is a representation $r_n = 0.d_{n0}d_{n1}\ldots d_{nn}\ldots$, where each d_{ni} is a decimal digit. Since we can also represent r_n by the sequence of decimal digits $(d_{n0}d_{n1}\ldots d_{nn}\ldots)$, it follows by diagonalization (2.12) that there is an infinite decimal that is not in the list. For example, we'll choose the digits 1 and

2 to construct the number $s = 0.s_0s_1s_2\ldots$ where

$$s_k = \begin{cases} 1 & \text{if } d_{kk} = 2 \\ 2 & \text{if } d_{kk} \neq 2. \end{cases}$$

So $0 < s < 1$ and s differs from each r_n at the nth decimal place. Thus s is not in the list, contrary to our assumption that we have listed all numbers in U. So the set U is uncountable, and hence also R is uncountable.

end example

example **2.23 Natural Number Functions**

How many different functions are there from N to N? We'll show that set of all such functions is uncountable. Assume, by way of contradiction, that the set is countable. Then we can list all the functions type $N \to N$ as $f_0, f_1, \ldots f_n, \ldots$. We'll represent each function f_n as the sequence of its values ($f_n(0)$, $f_n(1)$, ..., $f_n(n)$, ...). Now (2.12) tells us there is a function missing from the list, which contradicts our assumption that all functions are in the list. So the set of all functions of type $N \to N$ is uncountable.

For example, we might choose the numbers $1, 2 \in N$ and define a function $g : N \to N$ by

$$g(n) = \begin{cases} 1 & \text{if } f_n = 2 \\ 2 & \text{if } f_n \neq 2 \end{cases}$$

Then the sequence of values ($g(0)$, $g(1)$, ..., $g$listed functions because $g(n) \neq f_n(n)$ for each n. In this example there are many different ways to define $g : N \to N$ so that it is not in the list. For example, instead of picking 1 and 2, we could pick any two natural numbers to define g. We could also define g by

$$g(n) = f_n(n) + 1.$$

This definition gives us a function g from N to N such such that $g(n) \neq f_n(n)$ for each n. So g cannot be in the list f_0, f_1, ... f_n, ...

end example

2.4.3 Limits on Computability

Let's have a short discussion about whether there are limits on what can be computed. As another application of (2.11) we can answer the question: How many programs can be written in your favorite programming language? The answer is countably infinite. Here is the result.

Theorem (2.13)
The set of programs for a programming language is countably infinite.

Proof: One way to see this is to consider each program as a finite string of symbols over a fixed finite alphabet A. For example, A might consist of all characters that can be typed from a keyboard. Now we can proceed as in the proof of (2.11). For each natural number n, let P_n denote the set of all programs that are strings of length n over A. For example, the program

$$\{\text{print('help')}\}$$

is in P_{15} because it's a string of length 15. So the set of all programs is the union of the sets $P_0, P_1, \ldots, P_n, \ldots$. Since each P_n is finite, hence countable, it follows from (2.10) that the union is countable.

Not Everything Is Computable

Since there are “only” a countable number of computer programs, it follows that there are limits on what can be computed. For example, there are an uncountable number of functions of type $N \to N$. So there are programs to calculate only a countable set of these functions.

Can any real number be computed to any given number of decimal places? The answer is no. The reason is that there are “only” a countable number of computer programs (2.13) but the set of real numbers is uncountable. Therefore, there are only a countable number of computable numbers in R because each computable number needs a program to compute it. If we remove the computable numbers from R, the resulting set is still uncountable. Can you see why? So most real numbers cannot be computed.

The rational numbers can be computed, and there are also many irrational numbers that can be computed. Pi is the most famous example of a computable irrational number. In fact, there are countably infinitely many computable irrational numbers.

Higher Cardinalities

It's easy to find infinite sets having many different cardinalities because Cantor proved that there are more subsets of a set than there are elements of the set. In other words, for any set A, we have

Theorem (2.14)

$$|A| < |\text{power}(A)|\,.$$

We know this is true for finite sets. But it's also true for infinite sets. We'll discuss the proof in an exercise. Notice that if A is countably infinite, then we can conclude from (2.14) that power(A) is uncountable. So, for example, we can conclude that power(N) is uncountable.

For another example, we might wonder how many different languages there are over a finite alphabet such as $\{a, b\}$. Since a language over $\{a, b\}$ is a set of strings over $\{a, b\}$, it follows that such a language is a subset of $\{a, b\}^*$, the set of all strings over $\{a, b\}$. So the set of all languages over $\{a, b\}$ is power($\{a, b\}^*$). From (2.11) we can conclude that $\{a, b\}^*$ is countably infinite. In other words, we have $|\{a, b\}^*| = |N|$. So we can use (2.14) to obtain

$$|N| = |\{a, b\}^*| < |\text{power}(\{a, b\}^*)|.$$

Therefore, power($\{a, b\}^*$) is uncountable, which is the same as saying that there are uncountably many languages over the alphabet $\{a, b\}$. Of course, this generalizes to any finite alphabet. So we have the following statement.

Theorem (2.15)

There are uncountably many languages over a finite alphabet

We can use (2.14) to find infinite sequences of sets of higher and higher cardinality. For example, we have

$$|N| < |\text{power}(N)| < |\text{power}(\text{power}(N))| < \cdots$$

Can we associate these sets with more familiar sets? Sure, it can be shown that $|R| = |\text{power}(N)|$, which we'll discuss in an exercise. So we have

$$|N| < |R| < |\text{power}(\text{power}(N))| < \cdots$$

Is there any “well-known” set S such that $|S| = |\text{power}(\text{power}(N))|$? Since the real numbers are hard enough to imagine, how can we comprehend all the elements in power(power(N))? Luckily, in computer science we will seldom, if ever, have occasion to worry about sets having higher cardinality than the set of real numbers.

The Continuum Hypothesis

We'll close the discussion with a question: Is there a set S whose cardinality is between that of N and that of the real numbers R? In other words, does there exist a set S such that $|N| < |S| < |R|$? The answer is that no one knows. Interestingly, it has been shown that people who assume that the answer is yes won't run into any contradictions by using the assumption in their reasoning. Similarly, it has been shown that people who assume that the answer is no won't run into any contradictions by using the assumption in their arguments! The assumption that the answer is no is called the *continuum hypothesis*.

If we accept the continuum hypothesis, then we can use it as part of a proof technique. For example, suppose that for some set S we can show that $|N| \leq |S| < |R|$. Then we can conclude that $|N| = |S|$ by the continuum hypothesis.

Exercises

Finite Sets

1. Find the cardinality of each set by establishing a bijection between it and a set of the form $\{0, 1, \ldots, n\}$.

a. $\{2x + 5 \in N \mid 1 \leq 2x + 5 \leq 100\}$.
b. $\{x^2 \in N \mid 0 \leq x^2 \leq 500\}$.
c. $\{2, 5, 8, 11, 14, 17, \ldots, 44, 47\}$.

Countable Infinite Sets

2. Show that each of the following sets is countable by establishing a bijection between the set and N.

a. The set of even natural numbers.
b. The set of negative integers.
c. The set of strings over $\{a\}$.
d. The set of lists over $\{a\}$ that have even length.
e. The set Z of integers.
f. The set of odd integers.
g. The set of even integers.

3. Use (2.10) to show that each of the following sets is countable by describing the set as as a union of countable sets.

a. The set of strings over $\{a, b\}$ of that have odd length.
b. The set of all lists over {a, b}.
c. The set of all binary trees over $\{a, b\}$.
d. $N \times N \times N$.

Diagonalization

4. For each countable set of infinite sequences, use diagonalization (2.12) to construct an infinite sequence of the same type that is not in the set.

a. $\{(f_n(0), f_n(1), \ldots, f_n(n), \ldots) \mid f_n(k) \in \{\text{hello, world}\} \text{ for } n, k \in N\}$.
b. $\{(f(n, 0), f(n, 1), \ldots, f(n, n), \ldots) \mid f(n, k) \in \{a, b, c\} \text{ for } n, k \in N\}$.
c. $\{\{a_{n0}, a_{n1}, \ldots, a_{nn}, \ldots\}. \mid a_{nk} \in \{2, 4, 6, 8\} \text{ for } n, k \in N\}$.

5. To show that power(N) is uncountable, we can proceed by way of contradiction. Assume that it is countable, so that all the subsets of N can be listed $S_0, S_1, S_2, \ldots, S_n, \ldots$. Complete the proof by finding a way to represent each subset of N as an infinite sequence of 1's and 0's, where 1 means true and 0 means false. Then a contradiction arised by using diagonalization (2.12) to construct an infinite sequence of the same type that represents a subset of N that is not listed.

Proofs and Challenges

6. Show that if A is uncountable and B is a countable subset of A, then the set $A - B$ is uncountable.

7. Prove each statement about countable sets:

   a. Any subset of a countable set is countable.

   b. Any image of a countable set is countable.

8. Let A be a countably infinite alphabet $A = \{a_0, a_1, a_2, \ldots\}$. Let A^* denote the set of all strings over A. For each $n \in N$, let A_n denote the set of all strings in A^* having length n.

   a. Show that A_n is countable for $n \in \mathrm{N}$. *Hint:* Use (2.10).

   b. Show that A^* is countable. *Hint:* Use (2.10) and part (a).

9. Let finite(N) denote the set of all finite subsets of N. Use (2.10) to show that finite(N) is countable.

10. We'll start a proof that $|A| < |\text{power}(A)|$ for any set A. Proof: Since each element $x \in A$ can be associated with $\{x\} \in \text{power}(A)$, it follows that $|A| \leq |\text{power}(A)|$. To show that $|A| < |\text{power}(A)|$ we'll assume, by way of contradiction, that there is a bijection $A \to \text{power}(A)$. So each $x \in A$ is associated with a subset S_x of A. Now, define the following subset of A.

$$S = \{x \in A \mid x \notin S_x\}.$$

Since S is a subset of A, our assumed bijection tells us that there must be an element y in A that is associated with S. In other words, $S_y = S$. Find a contradiction by observing where y is and where it is not.

2.5 Chapter Summary

Functions allow us to associate different sets of objects. They are characterized by associating each domain element with a unique codomain element. For any function $f : A \to B$, subsets of the domain A have images in the codomain B

and subsets of B have pre-images in A. The image of A is the range of f. Partial functions need not be defined for all domain elements.

Some functions that are particularly useful in computer science are floor, ceiling, greatest common divisor (with the associated division algorithm), mod, and log.

Composition is a powerful tool for constructing new functions from known functions. Three functions that are useful in programming with lists are sequence, distribute, and pairs. The map function is a useful tool for computing lists of values of a function.

Three important properties of functions that allow us to compare sets are injective, surjective, and bijective. These properties are useful in describing the pigeonhole principle and in working with ciphers and hash functions. These properties are also useful in comparing the cardinality of sets.

A set is countable if it is finite or has the same cardinality as the set of natural numbers. Countable unions of countable sets are countable. The set of all computer programs is countable. The diagonalization technique can be used to show that a countable listing is not exhaustive. It can also be used to show that some sets, such as the real numbers, are uncountable. So we can’t compute all the real numbers. Any set has smaller cardinality than its power set, even when the set is infinite.