chapter 5

Analysis Techniques

Remember that time is money.

—Benjamin Franklin (1706–1790)

Time and space are important words in computer science because we want fast algorithms and we want algorithms that don't use a lot of memory. The purpose of this chapter is to study some fundamental techniques and tools that can be used to analyze algorithms for the time and space that they require. Although the study of algorithm analysis is beyond our scope, we'll give some examples to show how the process works.

The cornerstone of algorithm analysis is counting. So after an introduction to the ideas algorithm analysis, we'll concentrate on techniques to aid the counting process. We'll discuss techniques for finding closed forms for summations. Then we'll discuss permutations, combinations, and discrete probability. Next we'll introduce techniques for solving recurrences. Lastly, with an eye toward comparing algorithms, we'll discuss how to compare the growth rates of functions.

chapter guide

Section 5.1 introduces some ideas about analyzing algorithms. We'll define the worst-case performance of an algorithm and the idea of an optimal algorithm. Then we'll analyze a few example algorithms.

Section 5.2 introduces techniques for finding closed forms for sums that crop up in the analysis of algorithms.

Section 5.3 introduces basic counting techniques for permutations and combinations. We'll also introduce discrete probability so that we can discuss the average-case performance of algorithms.

Section 5.4 introduces techniques for solving recurrences that crop up in the analysis of algorithms.

Section 5.5 introduces techniques for comparing the rates of growth of functions. We'll apply the results to those functions that describe the approximate running time of algorithms.

5.1 Analyzing Algorithms

An important question of computer science is: Can you convince another person that your algorithm is efficient? This takes some discussion. Let's start by stating the following problem.

The Optimal Algorithm Problem

Suppose algorithm A solves problem P. Is A the best solution to P?

What does "best" mean? Two typical meanings are *least time* and *least space*. In either case, we still need to clarify what it means for an algorithm to solve a problem in the least time or the least space. For example, an algorithm running on two different machines may take different amounts of time. Do we have to compare A to every possible solution of P on every type of machine? This is impossible. So we need to make a few assumptions in order to discuss the optimal algorithm problem. We'll concentrate on "least time" as the meaning of "best" because time is the most important factor in most computations.

5.1.1 Worst-Case Running Time

Instead of executing an algorithm on a real machine to find its running time, we'll analyze the algorithm by counting the number of certain operations that it will perform when executed on a real machine. In this way we can compare two algorithms by simply comparing the number of operations of the same type that each performs. If we make a good choice of the type of operations to count, we should get a good measure of an algorithm's performance. For example, we might count addition operations and multiplication operations for a numerical problem. On the other hand, we might choose to count comparison operations for a sorting problem.

The number of operations performed by an algorithm usually depends on the size or structure of the input. The size of the input again depends on the problem. For example, for a sorting problem, "size" usually means the number of items to be sorted. Sometimes inputs of the same size can have different structures that affect the number of operations performed. For example, some sorting algorithms perform very well on an input data set that is all mixed up but perform badly on an input set that is already sorted!

Because of these observations we need to define the idea of a worst-case input for an algorithm A. An input of size n is a *worst-case input* if, when compared to all other inputs of size n, it causes A to execute the largest number of operations.

Now let's get down to business. For any input I we'll denote its size by size(I), and we'll let time(I) denote the number of operations executed by A on I. Then the *worst-case function* for A is defined as follows:

$$W_A(n) = \max\{\text{time}(I) \mid I \text{ is an input and size}(I) = n\}.$$

Now let's discuss comparing different algorithms that solve a problem P. We'll always assume that the algorithms we compare use certain specified operations that we intend to count. If A and B are algorithms that solve P and if $W_A(n) \leq W_B(n)$ for all $n > 0$, then we know algorithm A has worst-case performance that is better than or equal to that of algorithm B. This gives us the proper tool to describe the idea of an optimal algorithm.

Definition of Optimal in the Worst Case
An algorithm A is *optimal in the worst case* for problem P, if for any algorithm B that exists, or ever will exist the following relationship holds:

$$W_A(n) \leq W_B(n) \text{ for all } n > 0.$$

How in the world can we ever find an algorithm that is optimal in the worst case for a problem P? The answer involves the following three steps:

1. (Find an algorithm) Find or design an algorithm A to solve P. Then do an analysis of A to find the worst-case function W_A.
2. (Find a lower bound) Find a function F such that $F(n) \leq W_B(n)$ for all $n > 0$ and for all algorithms B that solve P.
3. Compare F and W_A. If $F = W_A$, then A is optimal in the worst case.

Suppose we know that $F \neq W_A$ in Step 3. This means that $F(n) < W_A(n)$ for some n. In this case there are two possible courses of action to consider:

1. Put on your construction hat and try to build a new algorithm C such that $W_C(n) \leq W_A(n)$ for all $n > 0$.
2. Put on your analysis hat and try to find a new function G such that $F(n) \leq G(n) \leq W_B(n)$ for all $n > 0$ and for all algorithms B that solve P.

We should note that zero is always a lower bound, but it's not very interesting because most algorithms take more than zero time. A few problems have optimal algorithms. For the vast majority of problems that have solutions, optimal algorithms have not yet been found. The examples contain both kinds of problems.

example

5.1 Matrix Multiplication

We can "multiply" two n by n matrices A and B to obtain the product AB, which is the n by n matrix defined by letting the element in the ith row and jth column of AB be the value of the expression . For example, let A and B be the following 2 by 2 matrices.

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \qquad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}.$$

The product AB is is the following 2 by 2 matrix:

$$AB = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}.$$

Notice that the computation of AB takes eight multiplications and four additions. The definition of matrix multiplication of two n by n matrices uses n^3 multiplication operations and $n^2(n - 1)$ addition operations.

A known lower bound for the number of multiplication operations needed to multiply two n by n matrices is n^2. Strassen [1969] showed how to multiply two matrices with about $n^{2.81}$ multiplication operations. The number 2.81 is an approximation to the value of $\log_2 (7)$. It stems from the fact that a pair of 2 by 2 matrices can be multiplied by using seven multiplication operations.

Multiplication of larger-size matrices is broken down into multiplying many 2 by 2 matrices. Therefore the number of multiplication operations becomes less than n^3. This revelation got research going in two camps. One camp is trying to find a better algorithm. The other camp is trying to raise the lower bound above n^2. In recent years, algorithms have been found with still lower numbers. Pan [1978] gave an algorithm to multiply two 70×70 matrices using 143,640 multiplications, which is less than $70^{2.81}$ multiplication operations. Coppersmith and Winograd [1987] gave an algorithm that, for large values of n, uses $n^{2.376}$ multiplication operations. So it goes.

end example

example

5.2 Finding the Minimum

Let's examine an optimal algorithm to find the minimum number in an unsorted list of n numbers. We'll count the number of comparison operations that an algorithm makes between elements of the list. To find the minimum number in a list of n numbers, the minimum number must be compared with the other $n - 1$ numbers. Therefore, $n - 1$ is a lower bound on the number of comparisons needed to find the minimum number in a list of n numbers.

If we represent the list as an array a indexed from 1 to n, then the following algorithm is optimal because the operation $\leq$ is executed exactly $n - 1$ times.

```
m := a[1];
for i := 2 to n do
        m := if m = a[i] then m else a[i]
od
```

end example

5.1.2 Decision Trees

We can often use a tree to represent the decision processes that take place in an algorithm. A *decision tree* for an algorithm is a tree whose nodes represent decision points in the algorithm and whose leaves represent possible outcomes. Decision trees can be useful in trying to construct an algorithm or trying to find properties of an algorithm. For example, lower bounds may equate to the depth of a decision tree.

If an algorithm makes decisions based on the comparison of two objects, then it can be represented by a *binary decision tree*. Each nonleaf node in the tree represents a pair of objects to be compared by the algorithm, and each branch from that node represents a path taken by the algorithm based on the comparison. Each leaf can represent an outcome of the algorithm. A *ternary decision tree* is similar except that each nonleaf node represents a comparison that has three possible outcomes.

Lower Bounds for Decision Tree Algorithms

Let's see whether we can compute lower bounds for decision tree algorithms. If a decision tree has depth d, then some path from the root to a leaf contains $d + 1$ nodes. Since the leaf is a possible outcome, it follows that there are d decisions made on the path. Since no other path from the root to a leaf can have more than $d + 1$ nodes, it follows that d is the worst-case number of decisions made by the algorithm.

Now, suppose that a problem has n possible outcomes and it can be solved by a binary decision tree algorithm. What is the best binary decision tree algorithm? We may not know the answer, but we can find a lower bound for the depth of any binary decision tree to solve the problem. Since the problem has n possible outcomes, it follows that any binary decision tree algorithm to solve the problem must have at least n leaves, one for each of the n possible outcomes. Recall that the number of leaves in a binary tree of depth d is at most 2^d.

So if d is the depth of a binary decision tree to solve a problem with n possible outcomes, then we must have $n \leq 2^d$. We can solve this inequality for d by taking $\log_2$ of both sides to obtain $\log_2 n \leq d$. Since d is a natural number, it follows that

$$\lceil \log_2 n \rceil \leq d.$$

In other words, any binary decision tree algorithm to solve a problem with n possible outcomes must have a depth of at least $\lceil \log_2 n \rceil$.

We can do the same analysis for ternary decision trees. The number of leaves in a ternary tree of depth d is at most 3^d. If d is the depth of a ternary decision tree to solve a problem with n possible outcomes, then we must have $n \leq 3^d$. Solve the inequality for d to obtain

$$\lceil \log_3 n \rceil \leq d.$$

In other words, any ternary decision tree algorithm to solve a problem with n possible outcomes must have a depth of at least $\lceil \log_3 n \rceil$.

Many sorting and searching algorithms can be analyzed with decision trees because they perform comparisons. Let's look at some examples to illustrate the idea.

example

5.3 Binary Search

Suppose we search a sorted list in a binary fashion. That is, we check the middle element of the list to see whether it's the key we are looking for. If not, then we perform the same operation on either the left half or the right half of the list, depending on the value of the key. This algorithm has a nice representation as a decision tree. For example, suppose we have the following sorted list of 15 numbers:

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}.$$

Suppose we're given a number key K, and we must find whether it is in the list. The decision tree for a binary search of the list has the number x_8 at its root. This represents the comparison of K with x_8. If $K = x_8$, then we are successful in one comparison. If $K < x_8$, then we go to the left child of x_8; otherwise we go to the right child of x_8. The result is a ternary decision tree in which the leaves are labeled with either S, for successful search, or U, for unsuccessful search. The decision tree is pictured in Figure 5.1.

Since the depth of the tree is 4, it follows that there will be four comparisons in the worst to find whether K is in the list. Is this an optimal algorithm? To see that the answer is yes, we can observe that there are 31 possible outcomes for the given problem: 15 leaves labeled with S to represent successful searchs; and 16 leaves labeled with U to represent the gaps where $K < x_1$, $x_i < K < x_{i+1}$ for $1 \leq i < 15$, and $x_{15} < K$. Therefore, a worst case lower bound for the number of comparisons is $\lceil \log_3 31 \rceil = 4$. Therefore the algorithm is optimal.

end example

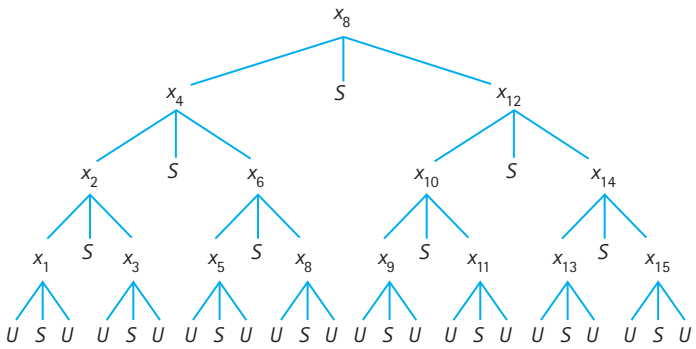

Figure 5.1 Decision tree for binary search.

example

5.4 Finding a Bad Coin

Suppose we are asked to use a pan balance to find the heavy coin among eight coins with the assumption that they all look alike and the other seven all have the same weight. One way to proceed is always place coins in the two pans that include the bad coin, so the pan will always go down.

This gives a binary decision tree, where each internal node of the tree represents the pan balance. If the left side goes down, then the heavy coin is on the left side of the balance. Otherwise, the heavy coin is on the right side of the balance. Each leaf represents one coin that is the heavy coin. Suppose we label the coins with the numbers 1, 2,. . . , 8.

The decision tree for one algorithm is shown in Figure 5.2, where the numbers on either side of a nonleaf node represent the coins on either side of the pan balance. This algorithm finds the heavy coin in three weighings. Can we do any better? Look at the next example.

end example

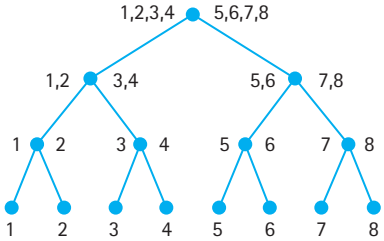

Figure 5.2 A binary decision tree.

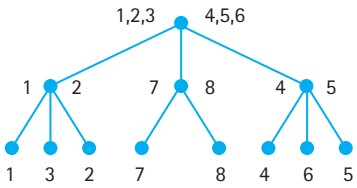

Figure 5.3 An optimal decision tree.

example

5.5 An Optimal Solution

The problem is the same as in Example 4. We are asked to use a pan balance to find the heavy coin among eight coins with the assumption that they all look alike and the other seven all have the same weight. In this case we'll weigh coins with the possibility that the two pans are balanced. So a decision tree can have nodes with three children.

So we don't have to use all eight coins on the first weighing. For example, Figure 5.3 shows the decision tree for one algorithm. Notice that there is no middle branch on the middle subtree because, at this point, one of the coins 7 or 8 must be the heavy one. This algorithm finds the heavy coin in two weighings.

This algorithm is an optimal pan-balance algorithm for the problem, where we are counting the number of weighings to find the heavy coin. To see this, notice that any one of the eight coins could be the heavy one. Therefore, there must be at least eight leaves on any algorithm's decision tree. But a binary tree of depth d can have 2^d possible leaves. So to get eight leaves, we must have $2^d \geq 8$. This implies that $d \geq 3$. But a ternary tree of depth d can have 3^d possible leaves. So to get eight leaves, we must have $3^d \geq 8$, or $d \geq 2$. Therefore, 2 is a lower bound for the number of weighings. Since the algorithm solves the problem in two weighings, it is optimal.

end example

example

5.6 A Lower Bound Computation

Suppose we have a set of 13 coins in which at most one coin is bad and a bad coin may be heavier or lighter than the other coins. The problem is to use a pan balance to find the bad coin if it exists and say whether it is heavy or light. We'll find a lower bound on the heights of decision trees for pan-balance algorithms to solve the problem.

Any solution must tell whether a bad coin is heavy or light. Thus there are 27 possible conditions: no bad coin and the 13 pairs of conditions (ith coin light, ith coin heavy). Therefore, any decision tree for the problem must have at least 27 leaves. So a ternary decision tree of depth d must satisfy $3^d \geq 27$, or $d \geq 3$. This gives us a lower bound of 3.

Now the big question: Is there an algorithm to solve the problem, where the decision tree of the algorithm has depth 3? The answer is no. Just look at the cases of different initial weighings, and note in each case that the remaining possible conditions cannot be distinguished with just two more weighings. Thus any decision tree for this problem must have depth 4 or more.

end example

Exercises

1. Draw a picture of the decision tree for an optimal algorithm to find the maximum number in the list x_1, x_2, x_3, x_4.

2. Suppose there are 95 possible answers to some problem. For each of the following types of decision tree, find a reasonable lower bound for the number of decisions necessary to solve the problem.

a. Binary tree. b. Ternary tree. **c.** Four-way tree.

3. Find a nonzero lower bound on the number of weighings necessary for any ternary pan-balance algorithm to solve the following problem: A set of 30 coins contains at most one bad coin, which may be heavy or light. Is there a bad coin? If so, state whether it's heavy or light.

4. Find an optimal pan-balance algorithm to find a bad coin, if it exists, from 12 coins, where at most one coin is bad (i.e., heavier or lighter than the others). *Hint:* Once you've decided on the coins to weigh for the root of the tree, then the coins that you choose at the second level should be the same coins for all three branches of the tree.

5.2 Finding Closed Forms

In trying to count things we often come up with expressions or relationships that need to be simplified to a form that can be easily computed with familiar operations.

Definition of Closed Form

A *closed form* is an an expression that can be computed by applying a fixed number of familiar operations to the arguments. A closed form can't have an ellipsis because the number of operations to evaluate the form would not be fixed. For example, the expression $n(n+1)/2$ is a closed form, but the expression $1 + 2 + \ldots + n$ is not a closed form. In this section we'll introduce some closed forms for sums.

5.2.1 Closed Forms for Sums

Let's start by reviewing a few important facts about summation notation and the indexes used for summing. We can use summation notation to represent a sum like $a_1 + a_2 + \cdots + a_n$ by writing

$$\sum_{i=1}^{n} a_i = a_1 + a_2 + \cdots + a_n.$$

Many problems can be solved by the simple manipulation of sums. So we'll begin by listing some useful facts about sums, which are easily verified.

Summation Fact (5.1)

a. $\displaystyle\sum_{i=m}^{n} c = (n - m + 1)c.$

b. $\displaystyle\sum_{i=m}^{n} (a_i + b_i) = \sum_{i=m}^{n} a_i + \sum_{i=m}^{n} b_i.$

c. $\displaystyle\sum_{i=m}^{n} c\, a_i = c \sum_{i=m}^{n} a_i.$

d. $\displaystyle\sum_{i=m}^{n} a_{i+k} = \sum_{i=m+k}^{n+k}.$ (k is any integer)

e. $\displaystyle\sum_{i=m}^{n} a_i x^{i+k} = x^k \sum_{i=m}^{n} a_i x^i.$ (k is any integer)

These facts are very useful in manipulating sums into simpler forms from which we might be able to find closed forms. So we better look at a few closed forms for some elementary sums. We already know some of them.

Closed Forms of Elementary Finite Sums (5.2)

a. $\displaystyle\sum_{i=1}^{n} = \frac{n(n+1)}{2}.$

b. $\displaystyle\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}.$

c. $\displaystyle\sum_{i=1}^{n} a^i = \frac{a^{n+1} - 1}{a - 1} \quad (a \neq 1).$

d. $\displaystyle\sum_{i=1}^{n} ia^i = \frac{a - (n+1)a^{n+1} + nan + 2}{(a-1)^2} \quad (a \neq 1).$

These closed forms are quite useful because they pop up in many situations when we are trying to count the number of operations performed by an algorithm. Are closed forms easy to find? Sometimes yes, sometimes no. Many techniques to find closed forms use properties (5.1) to manipulate sums into sums that have known closed forms such as those in (5.2). The following examples show a variety of ways to find closed forms.

example

5.7 A Sum of Odd Numbers

Suppose we need a closed form for the sum of the odd natural numbers up to a certain point:

$$1 + 3 + 5 + \cdots + (2n + 1).$$

We can write this expression using summation notation as follows:

$$\sum_{i=0}^{n} (2i + 1) = 1 + 3 + 5 + \cdots + (2n + 1).$$

Now, let's manipulate the sum to find a closed form. Make sure you can supply the reason for each step:

$$\begin{aligned}\sum_{i=0}^{n} (2i + 1) &= \sum_{i=0}^{n} 2i + \sum_{i=0}^{n} 1 \\ &= 2\sum_{i=0}^{n} i + \sum_{i=0}^{n} 1 \\ &= 2\frac{n\,(n+1)}{2} + (n + 1) = (n + 1)^2\,.\end{aligned}$$

end example

example

5.8 Finding a Geometric Progression

Suppose we forgot the closed form for a geometric progression (5.2c). Can we find it again? Here's one way. Let S_n be the following geometric progression, where $a \neq 1$.

$$S_n = 1 + a + a^2 + \cdots + a^n.$$

Notice that we can write the sum S_{n+1} in two different ways in terms of S_n:

$$\begin{aligned}S_{n+1} &= 1 + a + a^2 + \cdots + a^{n+1} \\ &= \left(1 + a + a^2 + \cdots + a^n\right) + a^{n+1} \\ &= S_n + a^{n+1},\end{aligned}$$

and

$$\begin{aligned} S_{n+1} &= 1 + a + a^2 + \cdots + a^{n+1} \\ &= 1 + a\left(1 + a + a^2 + \cdots + a^n\right) \\ &= 1 + aS_n. \end{aligned}$$

So we can equate the two expressions for S_{n+1} to obtain the equation

$$S_n + a^{n+1} = 1 + aS_n.$$

Since $a \neq 1$, we can solve the equation for S_n to obtain the well-known formula (5.2c) for a geometric progression.

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}.$$

We can verify the answer by mathematical induction on n or by multiplying both sides by $a - 1$ to get an equality.

end example

example

5.9 Finding a Closed Formula

Let's try to derive the closed formula given in (5.2d) for the sum $\sum_{i=1}^{n} ia^i$. Suppose we let $S_n = \sum_{i=1}^{n} ia^i$. As in Example 2, we'll find two expressions for S_{n+1} in terms of S_n. One expresion is easy:

$$S_{n+1} = \sum_{i=1}^{n+1} ia^i = \sum_{i=1}^{n} ia^i + (n+1)\,a^{n+1} = S_n + (n+1)\,a^{n+1}.$$

For the other expression we can proceed as follows:

$$\begin{aligned} S_{n+1} &= \sum_{i=1}^{n+1} ia^i \\ &= \sum_{i=0}^{n} (i+1)\,a^{i+1} && \text{(5.1d)} \\ &= \sum_{i=0}^{n} ia^{i+1} + \sum_{i=0}^{n} a^{i+1} && \text{(algebra and 5.1b)} \\ &= a\sum_{i=0}^{n} ia^i + a\sum_{i=0}^{n} a^i && \text{(5.1e)} \\ &= aS_n + a\left(\frac{a^{n+1} - 1}{a - 1}\right) && \text{(5.2c).} \end{aligned}$$

So we can equate the two expressions for S_{n+1} to obtain the equation

$$S_n + (n+1)\,a^{n+1} = aS_n + a\left(\frac{a^{n+1}-1}{a-1}\right).$$

Now solve the equation for S_n to obtain the closed form (5.2d):

$$\sum_{i=1}^{n} ia^i = \frac{a-(n+1)\,a^{n+1}+na^{n+2}}{(a-1)^2}.$$

end example

example

5.10 A Sum of Powers

A sum like $\sum_{i=1}^{n} i^3$ can be solved in terms of the two sums

$$\sum_{i=1}^{n} i \quad \text{and} \quad \sum_{i=1}^{n} i^2.$$

We'll start by adding the term $(n+1)^4$ to $\sum_{i=1}^{n} i^4$. Then we obtain the following equations.

$$\begin{aligned}\sum_{i=1}^{n} i^4 + (n+1)^4 &= \sum_{i=0}^{n} (i+1)^4 \\ &= \sum_{i=0}^{n} \left(i^4 + 4i^3 + 6i^2 + 4i + 1\right) \\ &= \sum_{i=1}^{n} i^4 + 4\sum_{i=1}^{n} i^3 + 6\sum_{i=1}^{n} i^2 + 4\sum_{i=1}^{n} i + \sum_{i=0}^{n} 1.\end{aligned}$$

Now subtract the term $\sum_{i=1}^{n} i^4$ from both sides of the above equation to obtain the following equation:

$$(n+1)^4 = 4\sum_{i=1}^{n} i^3 + 6\sum_{i=1}^{n} i^2 + 4\sum_{i=1}^{n} i + \sum_{i=0}^{n} 1. \tag{5.3}$$

Since we know the closed forms for the latter three sums on the right side of the equation, we can solve for $\sum_{i=1}^{n} i^3$ to find its closed form. We'll leave this as an exercise. We can use the same method to find a closed form for the expression $\sum_{i=1}^{n} i^k$ for any natural number k.

end example

example **5.11 The Polynomial Problem**

EXAMPLE 5 The Polynomial Problem

Suppose we're interested in the number of arithmetic operations needed to evaluate the following polynomial at some number x.

$$c_0 + c_1x + c_2x^2 + \cdots + c_nx^n.$$

The number of operations performed will depend on how we evaluate it. for example, suppose that we compute each term in isolation and then add up all the terms. There are no operatons if $n = 0$. If $n > 0$, then there are $n - 1$ addition operations and each term of the form c_ix^i takes i multiplication operations. So the total number of arithmetic operations for $n > 0$ is given by the following sum:

$$\begin{aligned}(n-1) + (1 + 2 + \cdots + n) &= (n-1) + \sum_{i=1}^{n} i \\ &= (n-1) + \frac{n(n+1)}{2} \\ &= \frac{n^2 + 3n - 2}{2}.\end{aligned}$$

So for even small values of n there are many operations to perform. For example, if $n = 30$, then there are 494 arithmetic operations to perform. Can we do better? Sure, we can group terms so that we don't have to repeatedly compute the same powers of x. We'll continue the discussion after we've introduced recurrences.

end example

example **5.12 Simple Sort**

In this example we'll construct a simple sorting algorithm and analyze it to find the number of comparison operations. We'll sort an array a of numbers indexed from 1 to n as follows: Find the smallest element in a, and exchange it with the first element. Then find the smallest element in positions 2 through n, and exchange it with the element in position 2. Continue in this manner to obtain a sorted array. To write the algorithm, we'll use a function "min" and a procedure "exchange," which are defined as follows:

> min(a, i, n) is the index of the minimum number among the elements $a[i]$, $a[i + 1]$,..., $a[n]$. We can easily modify the algorithm in Example 2 to accomplish this task with with $n - i$ comparisons.
>
> exchange($a[i]$, $a[j]$) is the usual operation of swapping elements and does not use any comparisons.

Now we can write the sorting algorithm as follows:

for $i := 1$ **to** $n - 1$ **do**
  $j := \min(a, i, n)$;
  exchange($a[i]$, $a[j]$)
od

Now let's compute the number of comparison operations. The algorithm for $\min(a, i, n)$ makes $n - i$ comparisons. So as i moves from 1 to $n - 1$, the number of comparison operations moves from $n - 1$ to $n - (n - 1)$. Adding these comparisons gives the sum of an arithmetic progression,

$$(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}.$$

The algorithm makes the same number of comparisons no matter what the form of the input array, even if it is sorted to begin with. So any arrangement of numbers is a worst-case input. For example, to sort 1,000 items it would take 499,500 comparisons, no matter how the items are arranged at the start.

end example

There are many faster sorting algorithms. For example, an algorithm called "heapsort" takes no more than $2n \log_2 n$ comparisons for its worst-case performance. So for 1000 items, heapsort would take a maximum of 20,000 comparisons—quite an improvement over our simple sort algorithm. In Section 5.3 we'll discover a good lower bound for the worst-case performance of comparison sorting algorithms.

Exercises

Closed Forms for Sums

1. Expand each expression into a sum of terms. Don't evaluate.

a. $\sum_{i=1}^{5} (2i + 3)$. b. $\sum_{i=1}^{5} i3^i$. **c.** $\sum_{i=0}^{4} (5 - i)\, 3^i$.

2. (*Changing Limits of Summation*). Given the following summation expression:

$$\sum_{i=1}^{n} g(i-1)\, a_i x^{i+1}.$$

For each of the following lower limits of summation, find an equivalent summation expression that starts with that lower limit.

a. $i = 0$. b. $i = 2$. **c.** $i = -1$. d. $i = 3$. e. $i = -2$.

3. Find a closed form for each of the following sums.

a. $3 + 6 + 9 + 12 + \cdots + 3n$.
b. $3 + 9 + 15 + 27 + \cdots + (6n + 3)$.
c. $3 + 6 + 12 + 24 + \cdots + 3(2^n)$.
d. $3 + (2)\,3^2 + (3)\,3^3 + (4)\,3^4 + \cdots + n3^n$.

4. For each of the following summations, use summation facts and known closed forms to transform each summation into a closed form.

a. $\sum_{i=1}^{n} (2i + 2)$.
b. $\sum_{i=1}^{n} (2i - 1)$.
c. $\sum_{i=1}^{n} (2i + 3)$.
d. $\sum_{i=1}^{n} (4i - 1)$.
e. $\sum_{i=1}^{n} (4i - 2)$.
f. $\sum_{i=1}^{n} i2^i$.
g. $\sum_{i=1}^{n} i\,(i + 1)$.
h. $\sum_{i=2}^{n} \left(i^2 - i\right)$.

5. Solve equation (5.3) to find a closed form for the expression $\sum_{i=1}^{n} i^3$.

Analyzing Algorithms

6. For the following algorithm, answer each question by giving a formula in terms of n:

```
for i := 1 to n do
    for j := 1 to i do x := x + f(x) od;
    x := x + g(x)
od
```

a. Find the number of times the assignment statement (:=) is executed during the running of the program.
b. Find the number of times the addition operation (+) is executed during the running of the program.

7. For the following algorithm, answer each question by giving a formula in terms of n:

```
i := 1;
while i < n + 1 do
    i := i + 1;
    for j := 1 to i do S od
od
```

a. Find the number of times the statement S is executed during the running of the program.

b. Find the number of times the assignment statement (:=) is executed during the running of the program.

Proofs and Challenges

8. For the following algorithm, answer each question by giving a formula in terms of n:

```
i := 1;
while i < n + 1 do
    i := i + 2;
    for j := 1 to i do S od
od
```

a. Find the number of times the statement S is executed during the running of the program.

b. Find the number of times the assignment statement (:=) is executed during the running of the program.

5.3 Counting and Discrete Probability

Whenever we need to count the number of ways that some things can be arranged or the number of subsets of things, there are some nice techniques that can help out. That's what our discussion of permutations and combinations will be about.

Whenever we need to count the number of operations performed by an algorithm, we need to consider whether the algorithm yields different results with each execution because of factors such as the size and/or structure of the input data. More generally, many experiments don't always yield the same results each time they are performed. For example, if we flip a coin we can't be sure of the outcome. This brings our discussion to probability. After introducing the basics, we'll see how probability is used when we need to count the number of operations performed by an algorithm in the average case.

5.3.1 Permutations (Order is Important)

In how many different ways can we arrange the elements of a set S? If S has n elements, then there are n choices for the first element. For each of these choices there are $n - 1$ choices for the second element. Continuing in this way, we obtain

$n! = n \cdot (n-1) \cdots 2 \cdot 1$ different arrangements of n elements. Any arrangement of n distinct objects is called a *permutation* of the objects. We'll write down the rule for future use:

Permutations (5.4)
There are $n!$ permutations of an n-element set.

For example, if $S = \{a, b, c\}$, then the six possible permutations of S, written as strings, are listed as follows:

$$abc,\ acb,\ bac,\ bca,\ cab,\ cba.$$

Now suppose we want to count the number of permutations of r elements chosen from an n-element set, where $1 \leq r \leq n$. There are n choices for the first element. For each of these choices there are $n-1$ choices for the second element. We continue this process r times to obtain the answer,

$$n\ (n-1) \cdots (n-r+1).$$

This number is denoted by the symbol $P(n, r)$ and is read "the number of permutations of n objects taken r at a time." We should emphasize here that we are counting r distinct objects. So we have the formulas shown in (5.5) and (5.6):

Permutations
The number of permutations of n objects taken r at a time is given by

$$P(n,r) = n(n-1)\cdots(n-r+1), \tag{5.5}$$

which can also be written,

$$P(n,r) = \frac{n!}{(n-4)!}. \tag{5.6}$$

Notice that $P(n, 1) = n$ and $P(n, n) = n!$. If $S = \{a, b, c, d\}$, then there are 12 permutations of two elements from S, given by the formula $P(4, 2) = 4!/2! = 12$. The permutations are listed as follows:

$$ab,\ ba,\ ac,\ ca,\ ad,\ da,\ bc,\ cb,\ bd,\ db,\ cd,\ dc.$$

Permutations with Repeated Elements

Permutations can be thought of as arrangements of objects selected from a set *without replacement*. In other words, we can't pick an element from the set more than once. If we can pick an element more than once, then the objects are said to be selected *with replacement*. In this case the number of arrangements of r objects from an n-element set is just n^r. We can state this idea in terms of bags as follows: The number of distinct permutations of r objects taken from a bag containing n distinct objects, each occurring r times, is n^r. For example, consider the bag $B = [a, a, b, b, c, c]$. Then the number of distinct permutations of two objects chosen from B is 3^2, and they can be listed as follows:

$$aa,\ ab,\ ac,\ ba,\ bb,\ bc,\ ca,\ cb,\ cc.$$

Let's look now at permutations of all the elements in a bag. For example, suppose we have the bag $B = [a, a, b, b, b]$. We can write down the distinct permutations of B as follows:

$$aabbb,\ ababb,\ abbab,\ abbba,\ baabb,\ babab,\ babba,\ bbaab,\ bbaba,\ bbbaa.$$

There are 10 strings. Let's see how to compute the number 10 from the information we have about the bag B. One way to proceed is to place subscripts on the elements in the bag, obtaining the five distinct elements a_1, a_2, b_1, b_2, b_3. Then we get $5! = 120$ permutations of the five distinct elements. Now we remove all the subscripts on the elements, and we find that there are many repeated strings among the original 120 strings.

For example, suppose we remove the subscripts from the two strings,

$$a_1b_1b_2a_2b_3 \quad \text{and} \quad a_2b_1b_3a_1b_2\ .$$

Then we obtain two occurrences of the string *abbab*. If we wrote all occurrences down, we would find 12 strings, all of which reduce to the string *abbab* when subscripts are removed. This is because there are 2! permutations of the letters a_1 and a_2, and there are 3! permutations of the letters b_1, b_2, and b_3. So there are $2!3! = 12$ distinct ways to write the string *abbab* when we use subscripts. Of course, the number is the same for any string of two a's and three b's. Therefore, the number of distinct strings of two a's and three b's is found by dividing the total number of subscripted strings by 2!3! to obtain $5!/2!3! = 10$. This argument generalizes to obtain the following result about permutations that can contain repeated elements.

Permutations of a Bag (5.7)
Let B be an n-element bag with k distinct elements, where each of the numbers $m_1,\ldots,\ m_k$ denotes the number of occurrences of each element. Then the number of permutations of the n elements of B is

$$\frac{n!}{m_1!\cdots m_k!}.$$

Now let's look at a few examples to see how permutations (5.4)–(5.7) can be used to solve a variety of problems. We'll start with an important result about sorting.

example

5.13 Worst-Case Lower Bound for Comparison Sorting

Let's find a lower bound for the number of comparisons performed by any algorithm that sorts by comparing elements in the list to be sorted. Assume that we have a set of n distinct numbers. Since there are $n!$ possible arrangements of these numbers, it follows that any algorithm to sort a list of n numbers has $n!$ possible input arrangements. Therefore, any decision tree for a comparison sorting algorithm must contain at least $n!$ leaves, one leaf for each possible outcome of sorting one arrangement.

We know that a binary tree of depth d has at most 2^d leaves. So the depth d of the decision tree for any comparison sort of n items must satisfy the inequality

$$n! = 2^d.$$

We can solve this inequality for the natural number d as follows:

$$\log_2 n! \leq d$$
$$\lceil \log_2 n! \rceil \leq d.$$

In other words, $\lceil \log_2 n! \rceil$ is a worst-case lower bound for the number of comparisons to sort n items. The number $\lceil \log_2 n! \rceil$ is hard to calculate for large values of n. We'll see in Section 5.5 that it is approximately $n \log_2 n$.

end example

example

5.14 People in a Circle

EXAMPLE 2 People in a Circle

In how many ways can 20 people be arranged in a circle if we don't count a rotation of the circle as a different arrangement? There are 20! arrangements of 20 people in a line. We can form a circle by joining the two ends of a line. Since there are 20 distinct rotations of the same circle of people, it follows that there are

$$\frac{20!}{20} = 19!$$

distinct arrangements of 20 people in a circle. Another way to proceed is to put one person in a certain fixed position of the circle. Then fill in the remaining 19 people in all possible ways to get 19! arrangements.

end example

example **5.15 Rearranging a String**

How many distinct strings can be made by rearranging the letters of the word *banana*? One letter is repeated twice, one letter is repeated three times, and one letter stands by itself. So we can answer the question by finding the number of permutations of the bag of letters $[b, a, n, a, n, a]$. Therefore, (5.7) gives us the result

$$\frac{6!}{1!2!3!} = 60.$$

end example

example **5.16 Strings with Restrictions**

How many distinct strings of length 10 can be constructed from the two digits 0 and 1 with the restriction that five characters must be 0 and five must be 1? The answer is

$$\frac{10!}{5!5!} = 252$$

because we are looking for the number of permutations from a 10-element bag with five 1's and five 0's.

end example

example **5.17 Constructing a Code**

Suppose we want to build a code to represent each of 29 distinct objects with a binary string having the same minimal length n, where each string has the same number of 0's and 1's. Somehow we need to solve an inequality like

$$\frac{n!}{k!k!} \geq 29,$$

where $k = n/2$. We find by trial and error that $n = 8$. Try it.

end example

5.3.2 Combinations (Order Is Not Important)

Suppose we want to count the number of r-element subsets in an n-element set. For example, if $S = \{a, b, c, d\}$, then there are four 3-element subsets of S: $\{a, b, c\}$, $\{a, b, d\}$, $\{a, c, d\}$, and $\{b, c, d\}$. Is there a formula for the general case? The answer is yes. An easy way to see this is to first count the number of r-element permutations of the n elements, which is given by the formula

$$P(n, r) = \frac{n!}{(n-4)!}.$$

Now each r-element subset has $r!$ distinct r-element permutations, which we have included in our count $P(n, r)$. How do we remove the repeated permutations from the count? Let $C(n, r)$ denote the number of r-element subsets of an n-element set. Since each of the r-element subsets has $r!$ distinct permutations, it follows that $r! \cdot C(n, r) = P(n, r)$. Now divide both sides by $r!$ to obtain the desired formula $C(n, r) = P(n, r)/r!$.

The expression $C(n, r)$ is usually said to represent the number of *combinations* of n things taken r at a time. With combinations, the order in which the objects appear is not important. We count only the different sets of objects. $C(n, r)$ is often read "n choose r." Here's the formula for the record.

Combinations (5.8)
The number of *combinations* of n things taken r at a time is given by

$$C(n,r) = \frac{P(n,r)}{r!} = \frac{n!}{r!\,(n-4!)}.$$

example

5.18 Subsets of the Same Size

Let $S = \{a, b, c, d, e\}$. We'll list all the three-element subsets of S:

$$\{a, b, c\}, \{a, b, d\}, \{a, b, e\}, \{a, c, d\}, \{a, c, e\},$$
$$\{a, d, e\}, \{b, c, d\}, \{b, c, e\}, \{b, d, e\}, \{c, d, e\}.$$

There are 10 such subsets, which we can verify by the formula

$$C(5,3) = \frac{5!}{3!2!} = 10.$$

end example

Binomial Coefficients

Notice how $C(n, r)$ crops up in the following binomial expansion of the expression $(x + y)^4$:

$$\begin{aligned}(x+y)^4 &= x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4 \\ &= C(4,0)\,x^4 + C(4,1)\,x^3y + C(4,2)\,x^2y^2 + C(4,3)\,xy^3 + C(4,4)\,y^4.\end{aligned}$$

A useful way to represent $C(n, r)$ is with the *binomial coefficient symbol:*

$$\binom{n}{r} = C(n,r).$$

Using this symbol, we can write the expansion for $(x + y)^4$ as follows:

$$(x+y)^4 = x^4 + 4x^3y + 6x^2y^2 + xy^3 + y^4$$
$$= \binom{4}{0}x^4 + \binom{4}{1}z^3y + \binom{4}{2}x^2y^2 + \binom{4}{3}xy^3 + \binom{4}{4}y^4.$$

This is an instance of a well-known formula called the *binomial theorem*, which can be written as follows, where n is a natural number:

Binomial Theorem (5.9)

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^{n-i}y^i.$$

Pascal's Triangle

The binomial coefficients for the expansion of $(x + y)^n$ can be read from the nth row of the table in Figure 5.4. The table is called *Pascal's triangle*—after the philosopher and mathematician Blaise Pascal (1623–1662). However, prior to the time of Pascal the triangle was known in China, India, the Middle East, and Europe. Notice that any interior element is the sum of the two elements above and to its left.

But how do we really know that the following statement is correct?

n	0	1	2	3	4	5	6	7	8	9	10
0	1										
1	1	1									
2	1	2	1								
3	1	3	3	1							
4	1	4	6	4	1						
5	1	5	10	10	5	1					
6	1	6	15	20	15	6	1				
7	1	7	21	35	35	21	7	1			
8	1	8	28	56	70	56	28	8	1		
9	1	9	36	84	126	126	84	36	9	1	
10	1	10	45	120	210	252	210	120	45	10	1

Figure 5.4 Pascal's triangle.

Elements in Pascal's Triangle (5.10)

The nth row kth column entry of Pascal's triangle is $\binom{n}{k}$.

Proof: For convenience we will designate a position in the triangle by an ordered pair of the form (row, column). Notice that the edge elements of the triangle are all 1, and they occur at positions $(n, 0)$ or (n, n). Notice also that

$$\binom{n}{0} = 1 = \binom{n}{n}.$$

So (5.10) is true when $k = 0$ or $k = n$. Next, we need to consider the interior elements of the triangle. So let $n > 1$ and $0 < k < n$. We want to show that the element in position (n, k) is $\binom{n}{k}$. To do this, we need the following useful result about binomial coefficients:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}. \tag{5.11}$$

To prove (5.11), just expand each of the three terms and simplify. Continuing with the proof of (5.10), we'll use well-founded induction. To do this, we need to define a well-founded order on something. For our purposes we will let the something be the set of positions in the triangle. We agree that any position in row $n - 1$ precedes any position in row n. In other words, if $n' < n$, then (n', k') precedes (n, k) for any values of k' and k. Now we can use well-founded induction. We pick position (n, k) and assume that (5.10) is true for all pairs in row $n - 1$. In particular, we can assume that the elements in positions $(n - 1, k)$ and $(n - 1, k - 1)$ have values

$$\binom{n-1}{k} \quad \text{and} \quad \binom{n-1}{k-1}.$$

Now we use this assumption along with (5.11) to tell us that the value of the element in position (n, k) is $\binom{n}{k}$. QED.

Can you find some other interesting patterns in Pascal's triangle? There are lots of them. For example, look down the column labeled 2 and notice that, for each $n \geq 2$, the element in position $(n, 2)$ is the value of the arithmetic sum $1 + 2 + \cdots + (n - 1)$. In other words, we have the formula

$$\binom{n}{2} = \frac{n(n-1)}{2}.$$

Combinations with Repeated Elements

Let's continue our discussion about combinations by counting bags of things rather than sets of things. Suppose we have the set $A = \{a, b, c\}$. How many 3-element bags can we construct from the elements of A? We can list them as follows:

$$[a, a, a], [a, a, b], [a, a, c], [a, b, c], [a, b, b],$$
$$[a, c, c], [b, b, b], [b, b, c], [b, c, c], [c, c, c].$$

So there are ten 3-element bags constructed from the elements of $\{a, b, c\}$.

Let's see if we can find a general formula for the number of k-element bags that can be constructed from an n-element set. For convenience, we'll assume that the n-element set is $A = \{1, 2, \ldots, n\}$. Suppose that $b = [x_1, x_2, x_3, \ldots, x_k]$ is some k-element bag with elements chosen from A, where the elements of b are written so that $x_1 \leq x_2 \leq \cdots \leq x_k$. This allows us to construct the following k-element set:

$$B = \{x_1, x_2 + 1, x_3 + 2, \ldots, x_k + (k - 1)\}.$$

The numbers $x_i + (i - 1)$ are used to ensure that the elements of B are distinct elements in the set $C = \{1, 2, \ldots, n + (k - 1)\}$. So we've associated each k-element bag b over A with a k-element subset B of C. Conversely, suppose that $\{y_1, y_2, y_3, \ldots, y_k\}$ is some k-element subset of C, where the elements are written so that $y_1 \leq y_2 \leq \cdots \leq y_k$. This allows us to construct the k-element bag

$$[y_1, y_2 - 1, y_3 - 2, \ldots, y_k - (k - 1)],$$

whose elements come from the set A. So we've associated each k-element subset of C with a k-element bag over A.

Therefore, the number of k-element bags over an n-element set is exactly the same as the number of k-element subsets of a set with $n + (k - 1)$ elements. This gives us the following result.

Bag Combinations (5.12)
The number of k-element bags whose distinct elements are chosen from an n-element set, where k and n are positive, is given by

$$\binom{n + k - 1}{k}.$$

example **5.19 Selecting Coins**

In how many ways can four coins be selected from a collection of pennies, nickels, and dimes? Let $S = \{\text{penny, nickel, dime}\}$. Then we need the number of 4-element bags chosen from S. The answer is

$$\binom{3+4-1}{4} = \binom{6}{4} = 15.$$

end example

example **5.20 Selecting a Committee**

In how many ways can five people be selected from a collection of Democrats, Republicans, and Independents? Here we are choosing five-element bags from a set of three characteristics {Democrat, Republican, Independent}. The answer is

$$\binom{3+5-1}{5} = \binom{7}{5} = 21.$$

end example

5.3.3 Discrete Probability

The founders of probability theory were Blaise Pascal (1623–1662) and Pierre Fermat (1601–1665). They developed the principles of the subject in 1654 during a correspondence about games of chance. It started when Pascal was asked about a gambling problem. The problem asked how the stakes of a "points" game should be divided up between two players if they quit before either had enough points to win.

Probability comes up whenever we ask about the chance of something happening. To answer such a question requires one to make some kind of assumption. For example, we might ask about the average behavior of an algorithm. That is, instead of the worst case performance, we might be interested in the average case performance. This can be bit tricky because it usually forces us to make one or two assumptions. Some people hate to make assumptions. But it's not so bad. Let's do an example.

Suppose we have a sorted list of the first 15 prime numbers, and we want to know the average number of comparisons needed to find a number in the list, using a binary search. The decision tree for a binary search of the list is pictured in Figure 5.5.

After some thought, you might think it reasonable to add up all the path lengths from the root to a leaf marked with an S (for successful search) and

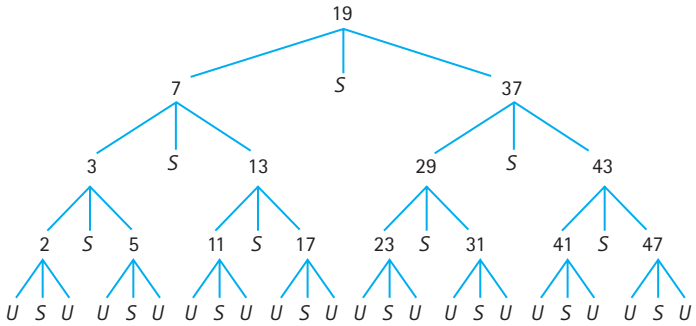

Figure 5.5 Binary search decision tree.

divide by the number of S leaves, which is 15. In this case there are eight paths of length 4, four paths of length 3, two paths of length 2, and one path of length 1. So we get

$$\text{Average path length} = \frac{32 + 12 + 4 + 1}{15} = \frac{49}{15} \approx 3.27.$$

This gives us the average number of comparisons needed to find a number in the list. Or does it? Have we made any assumptions here? Yes, we assumed that each path in the tree has the same chance of being traversed as any other path. Of course, this might not be the case. For example, suppose that we always wanted to look up the number 37. Then the average number of comparisons would be two. So our calculation was made under the assumption that each of the 15 numbers had the same chance of being picked.

Probability Terminology

Let's pause here and introduce some notions and notations for discrete probability, which gives us methods to calculate the likelihood of events that have a finite number of outcomes. If some operation or experiment has n possible outcomes and each outcome has the same chance of occurring, then we say that each outcome has *probability* $1/n$. In the preceding example we assumed that each number had probability $1/15$ of being picked. As another example, let's consider the coin-flipping problem. If we flip a fair coin, then there are two possible outcomes, assuming that the coin does not land on its edge. Thus the probability of a head is $1/2$, and the probability of a tail is $1/2$. If we flip the coin 1,000 times, we should expect about 500 heads and 500 tails. So probability has something to do with expectation.

Now for some terminology. The set of all possible outcomes of an experiment is called a *sample space*. The elements in a sample space are called *sample points* or simply *points*. Further, any subset of a sample space is called an *event*. For example, suppose we flip two coins and are interested in the set of possible outcomes. Let H and T mean head and tail, respectively, and let the string HT

mean that the first coin lands H and the second coin lands T. Then the sample space for this experiment is the set

$$\{HH,\ HT,\ TH,\ TT\}.$$

For example, the event that one coin lands as a head and the other coin lands as a tail can be represented by the subset $\{HT,\ TH\}$.

To discuss probability we need to make assumptions or observations about the probabilities of sample points. Here is the terminology.

Probability Distribution
A *probability distribution* on a sample space S is an assignment of probabilities to the points of S such that the sum of all the probabilities is 1.

Let's describe a probability distribution from a more formal point of view. Let $S = \{x_1,\ x_2,\ldots,\ x_n\}$ be a sample space. A probability distribution P on S is a function

$$P : S \to [0,\ 1]$$

such that

$$P(x_1) + P(x_2) + \cdots + P(x_n) = 1.$$

For example, in the two-coin-flip experiment it makes sense to define the following probability distribution on the sample space $S = \{HH,\ HT,\ TH,\ TT\}$:

$$P\,(HH) = P\,(HT) = P\,(TH) = P\,(TT) = \frac{1}{4}.$$

Probability of an Event

Once we have a probability distribution P defined on the points of a sample space S, we can use P to define the probability of any event E in S.

Probability of an Event
The *probability* of an event E is denoted by $P(E)$ and is defined by

$$P\,(E) = \sum_{x \in E} P\,(x)\,.$$

In particular, we have $P(S) = 1$ and $P(\varnothing) = 0$. If A and B are two events, then the following formula follows directly from the definition and the inclusion-exclusion principle.

$$P(A \cup B) = P(A) + P(B) - P(A \cap B).$$

This formula has a very useful consequence. If E' is the complement of E in S, then $S = E \cup E'$ and $E \cap E' = \varnothing$. So it follows from the formula that

$$P(E') = 1 - P(E).$$

example

5.21 Complement of an Event

In our two-coin-flip example, let E be the event that at least one coin is a tail. Then $E = \{HT, TH, TT\}$. We can calculate $P(E)$ as follows:

$$P(E) = P(\{HT, TH, TT\}) = P(HT) + P(TH) + P(TT) = \tfrac{1}{4} + \tfrac{1}{4} + \tfrac{1}{4} = \tfrac{3}{4}.$$

But we also could observe the that the complement E' is the event that both coins are heads. So we could calulate

$$P(E) = 1 - P(E') = 1 - P(HH) = 1 - \tfrac{1}{4} = \tfrac{3}{4}.$$

end example

Classic Example: The Birthday Problem

Suppose we ask 25 people, chosen at random, their birthday (month and day). Would you bet that they all have different birthdays? It seems a likely bet that no two have the same birthday since there are 365 birthdays in the year. But, in fact, the probability that two out of 25 people have the same birthday is greater than 1/2. Again, we're assuming some things here, which we'll get to shortly. Let's see why this is the case. The question we want to ask is:

> Given n people in a room, what is the probability that at least two of the people have the same birthday (month and day)?

We'll neglect leap year and assume that there are 365 days in the year. So there are 365^n possible n-tuples of birthdays for n people. This set of n-tuples is our sample space S. We'll also assume that birthdays are equally distributed throughout the year. So for any n-tuple $(b_1, \ldots, b_n)$ of birthdays, we have $P(b_1, \ldots, b_n) = 1/365^n$. The event E that we are concerned with is the subset of S consisting of all n-tuples that contain two or more equal entries. So our question can be written as follows:

What is $P(E)$?

To answer the question, let's use the negation technique. That is, we'll compute the probability of the event $E' = S - E$, consisting of all n-tuples that have distinct entries. In other words, no two of the n people have the same birthday. Then the probability that we want is $P(E) = 1 - P(E')$. So let's concentrate on E'.

n	$P(E)$
10	0.117
20	0.411
23	0.507
30	0.706
40	0.891

Figure 5.6 Birthday table.

An n-tuple is in E' exactly when all its components are distinct. The cardinality of E' can be found in several ways. For example, there are 365 possible values for the first element of an n-tuple in E'. For each of these 365 values there are 364 values for the second element of an n-tuple in E'. Thus we obtain

$$365 \cdot 364 \cdot 363 \cdot \cdots (365 - n + 1)$$

n-tuples in E'. Of course, this is also the number $P(365, n)$ of permutations of 365 things taken n at a time. Since each n-tuple of E' is equally likely with probability $1/365^n$, it follows that

$$P(E') = \frac{365 \cdot 364 \cdot 363 \cdots (365 - n + 1)}{365^n}.$$

Thus the probability that we desire is

$$P(E) = 1 - P(E') = 1 - \frac{365 \cdot 364 \cdot 363 \cdots (365 - n + 1)}{365^n}.$$

The table in Figure 5.6 gives a few calculations for different values of n.

Notice the case when $n = 23$. The probability is better than 0.5 that two people have the same birthday. Try this out next time you're in a room full of people. It always seems like magic when two people have the same birthday.

example 5.22 Switching Pays

Suppose there is a set of three numbers. One of the three numbers will be chosen as the winner of a three-number lottery. We pick one of the three numbers. Later, we are told that one of the two remaining numbers is not a winner, and we are given the chance to keep the number that we picked or to switch and choose the remaining number. What should we do? We should switch.

To see this, notice that once we pick a number, the probability that we did not pick the winner is 2/3. In other words, it is more likely that one of the other two numbers is a winner. So when we are told that one of the other numbers is not the winner, it follows that the remaining other number has probability 2/3

of being the winner. So go ahead and switch. Try this experiment a few times with a friend to see that in the long run it's better to switch.

Another way to see that switching is the best policy is to modify the problem to a set of 50 numbers and a 50-number lottery. If we pick a number, then the probability that we did not pick a winner is 49/50. Later we are told that 48 of the remaining numbers are not winners, but we are given the chance to keep the number we picked or switch and choose the remaining number. What should we do? We should switch because the chance that the remaining number is the winner is 49/50.

end example

Conditional Probability

If we ask a question about the chance of something happening given that something else has happened, we are using conditional probability.

Conditional Probability
If A and B are events and $P(B) \neq 0$, then *the conditional probability of* A *given* B is denoted by $P(A|B)$ and defined by

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

We can think of $P(A|B)$ as the probability of the event $A \cap B$ when the sample space is restricted to B once we make an appropriate adjustment to the probability distribution.

example **5.23 Conditional Probability**

In a university it is known that 1% of students major in mathematics and 2% major in computer science. Further, it is known that 0.1% of students major in both mathematics and computer science. If a student is chosen at random and happens to be a computer science major, what is the probability that the student is also majoring in mathematics?

To solve the problem we can let A and B be the sets of mathematics majors and computer science majors, respectively. Then the question asks for the value of $P(A|B)$. This is easily calculated because $P(A) = .01$, $P(B) = .02$, and $P(A \cap B) = .001$. Therefore $P(A|B) = .001/.02 = .05$.

end example

Suppose a sample space S is partitioned into disjoint events $E_1, \ldots, E_n$ and B is another event such that $P(B) \neq 0$. Then we can answer some interesting questions about the chance that "B was caused by E_i" for each i. In other

words, we can calculate $P(E_i|B)$ for each i. Although this is just a conditional probability, there is an interesting way to compute it in terms of the probabilities $P(B|E_i)$ and $P(E_i)$. The following formula, which is known as *Bayes' theorem*, follows from the assumption that the events $E_1,\ldots, E_n$ form a partition the sample space.

$$P(E_i|B) = \frac{P(E_i)\,P(B|E_i)}{P(E_1)\,P(B|E_1) + \cdots + P(E_n)\,P(B|E_n)}.$$

When using Bayes' theorem we can think of $P(E_i|B)$ as the probability that B *is caused by* E_i.

example

5.24 Probable Cause

Suppose that the input data set for a program is partitioned into two types, one makes up 60% of the data and the other makes up 40%. Suppose further that inputs from the two types cause warning messages 30% of the time and 20% of the time, respectively. If a random warning message is received, what are the chances that it was caused by an input of each type?

To solve the problem we can use Bayes' theorem. Let E_1 and E_2 be the two sets of data and let B be the set of data that causes warning messages. Then we want to find $P(E_1|B)$ and $P(E_2|B)$. Now we are given the following probabilities:

$$P(E_1) = .6,\ P(E_2) = 0.4,\ P(B|E_1) = .3,\ P(B|E_2) = .2$$

So we can calculate P() as follows:

$$\begin{aligned} P(E_1|B) &= \frac{P(E_1)\,P(B|E_1)}{P(E_1)\,P(B|E_1) + P(E_2)\,P(B|E_2)} \\ &= \frac{(.6)\,(.15)}{(.6)\,(.15) + (.4)\,(.2)} = \frac{.09}{.17} \approx .53. \end{aligned}$$

A similar calculation gives $P(E_2|B) \approx .47$.

end example

Independent Events

Informally, two events A and B are independent if they don't influence each other. If A and B don't influence each other and their probabilities are nonzero, we would like to say that $P(A|B) = P(A)$ and $P(B|A) = P(B)$. This condition follows from the definition of independence. Two events A and B are *independent* if the following equation holds:

$$P(A \cap B) = P(A) \cdot P(B).$$

It's interesting to note that if A and B are independent events, then so are the three pairs of events A and B', A' and B, and A' and B'. We'll discuss this in the exercises.

The nice thing about independent events is that it simplifies the task of assigning probabilities and computing probabilities.

example

5.25 Independence of Events

In the two-coin-flip example, let A be the event that the first coin is heads and let B be the event that the two coins come up different. Then $A = \{HT, HH\}$, $B = \{HT, TH\}$, and $A \cap B = \{HT\}$. If we each coin is fair, then A and B are independent because $P(A) = P(B) = 1/4$ and $P(A \cap B) = 1/2$.

Of course many events are not independent. For example, if C is the event that the at least one coin is tails, then $C = \{HT, TH, TT\}$. It follows that $A \cap C = \{HT\}$ and $B \cap C = \{HT, TH\}$. If the coins are fair, then it follows that A and C are dependent events and also that B and C are dependent events.

end example

Repeated Independent Trials

Independence is often used to assign probabilities for repeated trials of the same experiment. We'll be content here to discuss repeated trials of an experiment with two outcomes, where the trials are independent. For example, if we flip a coin n times, it's reasonable to assume that each flip is independent of the other flips. To make things a bit more general, we'll assume that a coin comes up either heads with probability p or tails with probability $1 - p$. Here is the question that we want to answer.

What is the probability that the coin comes up heads exactly k times?

To answer this question we need to consider the independence of the flips. For example, if we let A_i be the event that the ith coin comes up heads, then $P(A_i) = p$ and $P(A_i') = 1 - p$. Suppose now that we ask the probability that the first k flips come up heads and the last $n - k$ flips come up tails, then we are asking about the probability of the event

$$A_1 \cap \cdots \cap A_k \cap A_{k+1}^{'} \cap \cdots \cap A_n^{'}.$$

Since each event in the intersection is independent of the others events, the probability of the intersection is the product of probabilities

$$p^k(1 - \mathrm{p})^{n - k}.$$

We get the same answer for each arrangement of k heads and $n - k$ tails. So we'll have an answer the question if we can find the number of different arrangements

of k heads and $n - k$ tails. By (5.7) there are

$$\frac{n!}{k!\,(n-k)!}$$

such arrangements. This is also $C(n, k)$, which we can represent by the binomial coefficient symbol. So if a coin flip is repeated n times, then the probability of k successes is given by the expression

$$\binom{n}{k} p^k (1-p)^{n-k}.$$

This set of probabilities is called the *binomial distribution*. The name fits because by the binomial theorem, the sum of the probabilities as k goes from 0 to n is 1. We should note that although we used coin flipping to introduce the ideas, the binomial distribution applies to any experiment with two outcomes that has repeated trials.

Expectation = Average Behavior

Let's get back to talking about averages and expectations. We all know that the average of a bunch of numbers is the sum of the numbers divided by the number of numbers. So what's the big deal? The deal is that we often assign numbers to each outcome in a sample space. For example, in our beginning discussion we assigned a path length to each of the first 15 prime numbers. We added up the 15 path lengths and divided by 15 to get the average. Makes sense, doesn't it? But remember, we assumed that each number was equally likely to occur. This is not always the case. So we also have to consider the probabilities assigned to the points in the sample space.

Let's look at another example to set the stage for a definition of expectation. Suppose we agree to flip a coin. If the coin comes up heads, we agree to pay 4 dollars; if it comes up tails, we agree to accept 5 dollars. Notice here that we have assigned a number to each of the two possible outcomes of this experiment. What is our expected take from this experiment? It depends on the coin. Suppose the coin is fair. After one flip we are either 4 dollars poorer or 5 dollars richer. Suppose we play the game 10 times. What then? Well, since the coin is fair, it seems likely that we can expect to win five times and lose five times. So we can expect to pay 20 dollars and receive 25 dollars. Thus our expectation from 10 flips is 5 dollars.

Suppose we knew that the coin was biased with $P(\text{head}) = 2/5$ and $P(\text{tail}) = 3/5$. What would our expectation be? Again, we can't say much for just one flip. But for 10 flips we can expect about four heads and six tails. Thus we can expect to pay out 16 dollars and receive 30 dollars, for a net profit of 14 dollars. An equation to represent our reasoning follows:

$$10P\,(\text{head})\,(-4) + 10P\,(\text{tail})\,(5) = 10\left(\frac{2}{5}\right)(-4) + 10\left(\frac{3}{5}\right)(5) = \frac{70}{5} = 14.$$

Can we learn anything from this equation? Yes, we can. The 14 dollars represents our take over 10 flips. What's the average profit? Just divide by 10 to get \$1.40. This can be expressed by the following equation:

$$P\,(\text{head})\,(-4) + P\,(\text{tail})\,(5) = \left(\frac{2}{5}\right)(-4) + \left(\frac{3}{5}\right)(5) = \frac{70}{5} = 1.4.$$

So we can compute the average profit per flip without using the number of coin flips. The average profit per flip is \$1.40 no matter how many flips there are. That's what probability gives us. It's called *expectation*, and we'll generalize from this example to define expectation for any sample space having an assignment of numbers to the sample points.

Definition of Expectation

Let S be a sample space, P a probability distribution on S, and $V : S \to \mathbb{R}$ an assignment of numbers to the points of S. Suppose $S = \{x_1, x_2, \ldots, x_n\}$. Then the *expected value* (or *expectation*) of V is defined by the following formula.

$$E(V) = V(x_1)P(x_1) + V(x_2)P(x_2) + \cdots + V(x_n)P(x_n).$$

So when we want the average behavior, we're really asking for the expectation. For example, in our little coin-flip example we have $S = \{\text{head, tail}\}$, $P(\text{head}) = 2/5$, $P(\text{tail}) = 3/5$, $V(\text{head}) = -4$, and $V(\text{tail}) = 5$. So the expectation of V is calculated by $E(V) = (-4)(2/5) + 5(3/5) = 1.4$.

We should note here that in probability theory the function V is called a *random variable*.

Average Performance of an Algorithm

To compute the average performance of an algorithm A, we must do several things: First, we must decide on a sample space to represent the possible inputs of size n. Suppose our sample space is $S = \{I_1, I_2, \ldots, I_k\}$. Second, we must define a probability distribution P on S that represents our idea of how likely it is that the inputs will occur. Third, we must count the number of operations required by A to process each sample point. We'll denote this count by the function $V : S \to \mathbb{N}$. Lastly, we'll let $\text{Avg}_A(n)$ denote the average number of operations to execute A as a function of input size n. Then $\text{Avg}_A(n)$ is just the expectation of V:

$$\text{Avg}_A(n) = E(V) = V(I_1)P(I_1) + V(I_2)P(I_2) + \cdots + V(I_k)P(I_k).$$

To show that an algorithm A is *optimal in the average case* for some problem, we need to specify a particular sample space and probability distribution. Then we need to show that $\text{Avg}_A(n) \leq \text{Avg}_B(n)$ for all $n > 0$ and for all algorithms B that solve the problem. The problem of finding lower bounds for the average case is just as difficult as finding lower bounds for the worst case. So we're often content to just compare known algorithms to find the best of the bunch.

We'll finish the section with an example showing an average-case analysis of a simple algorithm for sequential search.

Analysis of Sequential Search

Suppose we have the following algorithm to search for an element X in an array L, indexed from 1 to n. If X is in L, the algorithm returns the index of the rightmost occurrence of X. The index 0 is returned if X is not in L:

```
i := n;
while i ≥ 1 and X ≠ L[i] do
    i := i – 1
od
```

We'll count the average number of comparisons $X \neq L[i]$ performed by the algorithm. Frst we need a sample space. Suppose we let I_i denote the input case where the rightmost occurrence of X is at the ith position of L. Let I_{n+1} denote the case in which X is not in L. So the sample space is the set

$$\{I_1, I_2, \ldots, I_{n+1}\}.$$

Let $V(I)$ denote the number of comparisons made by the algorithm when the input has the form I. Looking at the algorithm, we obtain

$$\begin{aligned} V(I_i) &= n - i + 1 \quad \text{for } 1 \leq i \leq n, \\ V(I_{n+1}) &= n. \end{aligned}$$

Suppose we let q be the probability that X is in L. Thus $1 - q$ is the probability that X is not in L. Let's also assume that whenever X is in L, its position is random. This gives us the following probability distribution P over the sample space:

$$\begin{aligned} P(I_i) &= \frac{q}{n} \quad \text{for } 1 \leq i \leq n, \\ P(I_{n+1}) &= 1 - q. \end{aligned}$$

Therefore, the expected number of comparisons made by the algorithm for this probability distribution is given by the expected value of V:

$$\begin{aligned} \text{Avg}_A(n) = E(V) &= V(I_1)P(I_1) + \cdots + V(I_{n+1})P(I_{n+1}) \\ &= \frac{q}{n}(n + (n-1) + \cdots + 1) + (1-q)n \\ &= q\left(\frac{n+1}{2}\right) + (1-q)n. \end{aligned}$$

Let's observe a few things about the expected number of comparisons. If we know that X is in L, then $q = 1$. So the expectation is $(n + 1)/2$ comparisons. If we know that X is not in L, then $q = 0$, and the expectation is n comparisons. If X is in L and it occurs at the first position, then the algorithm takes n comparisons. So the worst case occurs for the two input cases I_{n+1} and I_1, and we have $W_A(n) = n$.

Approximations (Monte Carlo Method)

Sometimes it is not so easy to find a formula to solve a problem. In some of these cases we can find reasonable approximations by repeating some experiment many times and then observing the results. For example, suppose we have an irregular shape drawn on a piece of paper and we would like to know the area of the shape. The *Monte Carlo method* would have us randomly choose a large number of points on the paper. Then the area of the shape would be pretty close to the percentage of points that lie within the shape multiplied by the area of the paper.

The Monte Carlo method is useful in probability not only to check a calculated answer for a problem, but to find reasonable answers to problems for which we have no other answer. For example, a computer simulating thousands of repetitions of an experiment can give a pretty good approximation to the average outcome of the experiment.

Exercises

Permutations and Combinations

1. Evaluate each of the following expressions.

a. $P(6, 6)$. b. $P(6, 0)$. **c.** $P(6, 2)$.
d. $P(10, 4)$. **e.** $C(5, 2)$. f. $C(10, 4)$.

2. Let $S = \{a, b, c\}$. Write down the objects satisfying each of the following descriptions.

a. All permutations of the three letters in S.
b. All permutations consisting of two letters from S.
c. All combinations of the three letters in S.
d. All combinations consisting of two letters from S.
e. All bag combinations consisting of two letters from S.

3. For each part of Exercise 2, write down the formula, in terms of P or C, for the number of objects requested.

4. Given the bag $B = [a, a, b, b]$, write down all the bag permutations of B, and verify with a formula that you wrote down the correct number.

5. Find the number of ways to arrange the letters in each of the following words. Assume all letters are lowercase.

a. Computer. b. Radar. **c.** States.
d. Mississippi. **e.** Tennessee.

6. A *derangement* of a string is a permutation of the letters such that each letter changes its position. For example, a derangement of the string ABC

is BCA. But ACB is not a derangement of ABC, since A does not change position. Write down all derangements for each of the following strings.

a. A. b. AB. **c.** ABC. d. $ABCD$.

7. Suppose we want to build a code to represent 29 objects in which each object is represented as a binary string of length n, which consists of k 0's and m 1's, and $n = k + m$. Find n, k, and m, where n has the smallest possible value.

8. We wish to form a committee of seven people chosen from five Democrats, four Republicans, and six Independents. The committee will contain two Democrats, two Republicans, and three Independents. In how many ways can we choose the committee?

9. Each row of Pascal's triangle (Figure 5.4) has a largest number. Find a formula to describe which column contains the largest number in row n.

Discrete Probability

10. Suppose three fair coins are flipped. Find the probability for each of the following events.

a. Exactly one coin is a head. b. Exactly two coins are tails.
c. At least one coin is a head. d. At most two coins are tails.

11. Suppose a pair of dice are flipped. Find the probability for each of the following events.

a. The sum of the dots is 7.
b. The sum of the dots is even.
c. The sum of the dots is either 7 or 11.
d. The sum of the dots is at least 5.

12. A team has probability 2/3 of winning whenever it plays. Find each of the following probabilities that the team will win.

a. Exactly 4 out of 5 games.
b. At most 4 out of 5 games.
c. Exactly 4 out of 5 games given that it has already won the first 2 games of a 5-game series.

13. A baseball player's batting average is.250. Find each of the following probabilities that he will get hits.

a. Exactly 2 hits in 4 times at bat.
b. At least one hit in 4 times at bat.

14. A computer program uses one of three procedures for each piece of input. The procedures are used with probabilities 1/3, 1/2, and 1/6. Negative results are detected at rates of 10%, 20%, and 30% by the three procedures,

respectively. Suppose a negative result is detected. Find the probabilities that each of the procedures was used.

15. A commuter crosses one of three bridges, A, B, or C, to go home from work, crossing A with probability 1/3, B with probability 1/6, and C with probability 1/2. The commuter arrives home by 6 p.m. 75%, 60%, and 80% of the time by crossing bridges A, B, and C, respectively. If the commuter arrives home after 6 p.m., find the probability that bridge A was used. Also find the probabilities for bridges B and C.

16. A student is chosen at random from a class of 80 students that has 20 honor students, 30 athletes, and 40 that are neither honor students nor athletes.

a. What is the probability that the student selected is an athlete given that he or she is an honors student?

b. What is the probability that the student selected is an honors student given that he or she is an athlete?

c. Are the events "honors student" and "athlete" independent?

17. Suppose we have an algorithm that must perform 2,000 operations as follows: The first 1,000 operations are performed by a processor with a capacity of 100,000 operations per second. Then the second 1,000 operations are performed by a processor with a capacity of 200,000 operations per second. Find the average number of operations per second performed by the two processors to execute the 2,000 operations.

18. Consider each of the following lottery problems.

a. Find the chances of winning a lottery that allows you to pick six numbers from the set $\{1, 2, \ldots, 49\}$.

b. Suppose that a lottery consists of choosing a set of five numbers from the set $\{1, 2, \ldots, 49\}$. Suppose further that smaller prizes are given to people with four of the five winning numbers. What is the probability of winning a smaller prize?

c. Suppose that a lottery consists of choosing a set of six numbers from the set $\{1, 2, \ldots, 49\}$. Suppose further that smaller prizes are given to people with four of the six winning numbers. What is the probability of winning a smaller prize?

d. Find a formula for the probability of winning a smaller prize that goes with choosing k of the winning m numbers from the set $\{1, \ldots, n\}$, where $k < m < n$.

19. For each of the following problems, compute the expected value.

a. The expected number of dots that show when a die is tossed.

b. The expected score obtained by guessing all 100 questions of a true-false exam in which a correct answer is worth 1 point and an incorrect answer is worth –1/2 point.

Challenges

20. Test the birthday problem on a group of people.

21. Show that if S is a sample space and A is an event, then S and A are independent events. What about the independence of two events A and B that are disjoint?

22. Prove that if A and B are independent events, then so are the three pairs of events A and B', A' and B, and A' and B'.

23. Suppose an operating system must schedule the execution of n processes, where each process consists of k separate actions that must be done in order. Assume that any action of one process may run before or after any action of another process. How many execution schedules are possible?

24. Count the number of strings consisting of n 0's and n 1's such that each string is subject to the following restriction: As we scan a string from left to right, the number of 0's is never greater than the number of 1's. For example, the string 110010 is OK, but the string 100110 is not. *Hint:* Count the total number of strings of length $2n$ with n 0's and n 1's. Then try to count the number that are not OK, and subtract this number from the total number.

25. Given a nonempty finite set S with n elements, prove that there are $n!$ bijections from S to S.

26. (*Average-Case Analysis of Binary Search*).

a. Assume that we have a sorted list of 15 elements, x_1, x_2,..., x_{15}. Calculate the average number of comparisons made by a binary search algorithm to look for a key that may or may not be in the list. Assume that the key has probability 1/2 of being in the list and that each of the events "key = x_i" is equally likely for $1 \leq i \leq 15$.

b. Generalize the problem to find a formula for the average number of comparisons used to look for a key in a sorted list of size $n = 2^k - 1$, where k is a natural number. Assume that the key has probability p of being in the list and that each of the events "key = x_i" is equally likely for $1 \leq i \leq n$. Test your formula with $n = 15$ and $p = 1/2$ to see that you get the same answer as part (a).

5.4 Solving Recurrences

Many counting problems result in answers that are expressed in terms of recursively defined functions. For example, any program that contains recursively defined procedures or functions will give rise to such expressions. Many of these

expressions have closed forms that can simplify the counting process. So let's discuss how to find closed forms for such expressions.

Definition of Recurrence Relation

Any recursively defined function f with domain N that computes numbers is called a *recurrence* or a *recurrence relation*. When working with recurrences we often write f_n in place of $f(n)$. For example, the following definition is a recurrence:

$$\begin{aligned} r(0) &= 1 \\ r(n) &= 2r(n-1) + n. \end{aligned}$$

We can also write this recurrence in the following useful form.

$$\begin{aligned} r_0 &= 1 \\ r_n &= 2r_{n+1} + n. \end{aligned}$$

To *solve a recurrence* r we must find an expression for the general term r_n that is not recursive.

5.4.1 Solving Simple Recurrences

Let's start by with some simple recurrences that can be solved without much fanfare. The recurrences we'll be considering have the following general form, where a_n and b_n denote either constants or expressions involving n but not involving r.

$$\begin{aligned} r_0 &= b_0, \\ r_n &= a_n r_{n-1} + b_n. \end{aligned} \tag{5.13}$$

We'll look at two similar techniques for solving these recurrences.

Solving by Substitution

One way to solve ecurrences of the form (5.13) is by *substitution*, where we start with the definition for r_n and keep substituting for occurrences of r on the right side of the equation until we discover a pattern that allows us to skip ahead and eventually replace the basis r_0.

We'll demonstrate the substitution technique in general terms by solving (5.13). Note the patterns that emerge with each substitution for r.

$$\begin{aligned}
r_n &= a_n r_{n-1} + b_n \\
&= a_n \left(a_{n-1} r_{n-2} + b_{n-1}\right) + b_n && \text{(replace } r_{n-1} = a_{n-1} r_{n-2} + b_{n-1}\text{)} \\
&= a_n a_{n-1} r_{n-2} + a_n b_{n-1} + b_n && \text{(regroup)} \\
&= a_n a_{n-1} \left(a_{n-2} r_{n-3} + b_{n-3}\right) + a_n b_{n-1} + b_n \\
& && \text{(replace } r_{n-2} = a_{n-2} r_{n-3} + b_{n-2}\text{)} \\
&= a_n a_{n-1} a_{n-2} r_{n-3} + a_n a_{n-1} b_{n-3} + a_n b_{n-1} + b_n && \text{(regroup)} \\
& \quad \vdots \\
&= a_n \cdots a_2 r_1 + a_n \cdots a_3 b_2 + \cdots + a_n b_{n-1} + b_n && \text{(regroup)} \\
&= a_n \cdots a_2 \left(a_1 r_0 + b_1\right) + a_n \cdots a_3 b_2 + \cdots + a_n b_{n-1} + b_n \\
& && \text{(replace } r_1 = a_1 r_0 + b_1\text{)} \\
&= a_n \cdots a_2 a_1 r_0 + a_n \cdots a_2 b_1 + a_n \cdots a_3 b_2 + \cdots + a_n b_{n-1} + b_n && \text{(regroup)} \\
&= a_n \cdots a_2 a_1 b_0 + a_n \cdots a_2 b_1 + a_n \cdots a_3 b_2 + \cdots + a_n b_{n-1} + b_n \\
& && \text{(replace } r_0 = b_0\text{)}
\end{aligned}$$

example

5.26 Solving by Substitution

We'll solve the following recurrence by substitution.

$$\begin{aligned}
r_0 &= 1, \\
r_n &= 2r_{n-1} + n.
\end{aligned}$$

Notice in the following solutions that we never multply numbers. Instead we keep track of products to help us discover general patterns. Once we find a pattern we emphasize it with parentheses and exponents. Each line represents a substitution and regrouping of terms.

$$\begin{aligned}
r_n &= 2r_{n-1} + n \\
&= 2^2 r_{n-2} + 2\left(n-1\right) + n \\
&= 2^3 r_{n-3} + 2^2\left(n-2\right) + 2\left(n-1\right) + n \\
& \quad \vdots \\
&= 2^{n-1} r_1 + 2^{n-2}\left(2\right) + 2^{n-2}\left(2\right) + \cdots + 2^2\left(n-2\right) + 2^1\left(n-1\right) + 2^0\left(n\right) \\
&= 2^n r_0 + 2^{n-1}\left(1\right) + 2^{n-2}\left(2\right) + \cdots + 2^2\left(n-2\right) + 2^1\left(n-1\right) + 2^0\left(n\right) \\
&= 2^n\left(1\right) + 2^{n-1}\left(1\right) + 2^{n-2}\left(2\right) + \cdots + 2^2\left(n-2\right) + 2^1\left(n-1\right) + 2^0\left(n\right).
\end{aligned}$$

Now we'll put it into closed form using (5.1), (5.2c), and (5.2d). Be sure you can see the reason for each step. We'll start by keeping the first term where it is and reversing the rest of the sum to get it in a nicer form.

$$\begin{aligned}
r_n &= 2^n\,(1) + n + 2\,(n-1) + 2^2\,(n-2) + \cdots + 2^{n-2}\,(2) + 2^{n-1}\,(1) \\
&= 2^n + \left[2^0\,(n) + 2^1\,(n-1) + 2^2\,(n-2) + \cdots + 2^{n-2}\,(2) + 2^{n-1}\,(1)\right] \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(group terms)} \\
&= 2^n + \sum_{i=0}^{n-1} 2^i\,(n-i) \\
&= 2^n + n\sum_{i=0}^{n-1} 2^i - \sum_{i=0}^{n-1} i2^i \\
&= 2^n + n\,(2^n - 1) - \left(2 - n2^n + (n-1)\,2^{n+1}\right) \\
&= 2^n\,(1 + n + n - 2n + 2) - n - 2 \\
&= 3\,(2^n) - n - 2.
\end{aligned}$$

Now check a few values of r_n to make sure that the sequence of numbers for the closed form and the recurrence are the same: $1, 3, 8, 19, 42, \ldots$.

end example

Solving by Cancellation

An alternative technique to solve recurences of the form (5.13) is by *cancellation*, where we start with the general equation for r_n. The term on the left side of each succeeding equation is the same as the term that contains r on the right side of the preceding equation. We normally write a few terms until a pattern emerges. The last equation always contains the basis element r_0 on the right side. Here is a sample.

$$\begin{aligned}
r_n &= a_n r_{n-1} + b_n \\
a_n r_{n-1} &= a_n a_{n-1} r_{n-2} + a_n b_{n-1} \\
a_n a_{n-1} r_{n-2} &= a_n a_{n-1} a_{n-2} r_{n-3} + a_n a_{n-1} b_{n-2} \\
&\vdots \\
a_n \cdots a_3 r_2 &= a_n \cdots a_2 r_1 + a_n \cdots a_3 b_2 \\
a_n \cdots a_2 r_1 &= a_n \cdots a_1 r_0 + a_n \cdots a_2 b_1
\end{aligned}$$

Now we add up the equations and observe that, except for r_n in the first equation, all terms on the left side of the remaning equations cancel with like terms on the right side of preceding equations. So that the sum of the equations gives us the following formula for r_n, where we have replaced r_0 by its basis value b_0.

$$r_n = a_n \cdots a_1 b_0 + (b_n + a_n b_{n-1} + a_n a_{n-1} b_{n-2} + \cdots + a_n \cdots a_3 b_2 + a_n \cdots a_2 b_1)$$

So we get to the same place by either substitution or cancellation. Since mistakes are easy to make it is nice to know that you can always check your solution against the original recurrence by testing. You also give an induction proof that your solution is correct.

example **5.27 Solving by Cancellation**

We'll solve the recurrence in Example 1 by cancellation:

$$\begin{aligned} r_0 &= 1, \\ r_n &= 2r_{n-1} + n. \end{aligned}$$

Starting with the general term, we obtain the following sequence of equations, where the term on the left side of a new equation is always the term that contains r from the right side of the preceding equation.

$$\begin{aligned} r_n &= 2r_{n-1} + n \\ 2r_{n-1} &= 2^2 r_{n-2} + 2\,(n-1) \\ 2^2 r_{n-2} &= 2^3 r_{n-3} + 2^2\,(n-2) \\ &\vdots \\ 2^{n-2} r_2 &= 2^{n-1} r_1 + 2^{n-2}\,(2) \\ 2^{n-1} r_1 &= 2^n r_0 + 2^{n-1}\,(1) \end{aligned}$$

Now add up all the equations, cancel the like terms, and replace r_0 by its value, to get the following equation.

$$r_n = 2^n\,(1) + n + 2\,(n-1) + 2^2\,(n-2) + \cdots + 2^{n-2}\,(2) + 2^{n-1}\,(1)\,.$$

Notice that, except for the ordering of terms, the solution is the same as the one obtained by substitution in Example 1.

end example

The Polynomial Problem

In Example 5 of Section 5.2 we found the number of arithmetic operations in a polynomial of degree n. By grouping terms of the polynomial we can reduce repeated multiplications. For example, here is the grouping when $n = 3$:

$$c_0 + c_1 x + c_2 x^2 + c_3 x^3 = c_0 + x\,(c_1 + x\,(c_2 + x\,(c_3)))\,.$$

Notice that the expression on the left uses 9 operations while the expression on the right uses 6. The following function will evaluate a polynomial with terms grouped in this way, where C is the list of coefficients.

$$\text{poly}(C,\, x) = \text{if } C = \langle\ \rangle \text{ then } 0 \text{ else head}(C) + x\ {}_*\ \text{poly}(\text{tail}(C),\, x).$$

For example, we'll evaluate the expression poly($\langle\ a,\ b,\ c,\ d\rangle\ ,\ x$):

$$\begin{aligned}
\text{poly}\left(\langle a,b,c,d\rangle, x\right) &= a + x * \text{poly}\left(\langle b,c,d\rangle, x\right) \\
&= a + x * (b + x * \text{poly}\left(\langle c,d\rangle, x\right)) \\
&= a + x * (b + x * (c + x * \text{poly}\left(\langle d\rangle, x\right))) \\
&= a + x * (b + x * (c + x * (d + x * \text{poly}\left(\langle\rangle, x\right)))) \\
&= a + x * (b + x * (c + d * 0))
\end{aligned}$$

So there are 6 arithmetic operations performed by poly to evaluate a polynomial of degree 3. Let's figure out how many operations are performed to evalutate a polynomial of degree n. Let $a(n)$ denote the number of arithmetic operations performed by poly(C, x) when C has length n. If $n = 0$, then $C = \langle\ \rangle$ and

$$\text{poly}(C,\, x) = \text{poly}(\langle\ \rangle\ ,\ x) = 0.$$

Therefore, $a(0) = 0$. If $n > 0$, then $C \neq \langle\ \rangle$ and

$$\text{poly}(C,\, x) = \text{head}(C) + x\ {}_*\ \text{poly}(\text{tail}(C),\, x).$$

This expression has two arithmetic operations plus the number of operations performed by poly(tail(C), x). Since tail(C) has n – 1 elements, it follows that poly(tail(C), x) performs $a(n-1)$ operations. Therefore, for n > 0 we have a(n) = a(n – 1) + 2. So we have the following recursive definition:

$$\begin{aligned}
a\,(0) &= 0 \\
a\,(n) &= a\,(n-1) + 2.
\end{aligned}$$

Writing it in subscripted form we have

$$\begin{aligned}
a_0 &= 0 \\
a_n &= a_{n-1} + 2
\end{aligned}$$

It's easy to solve this recurrence by cancellation:

$$\begin{aligned}
a_n &= a_{n-1} + 2 \\
a_{n-1} &= a_{n-2} + 2 \\
a_{n-2} &= a_{n-3} + 2 \\
&\vdots \\
a_2 &= a_1 + 2 \\
a_1 &= a_0 + 2
\end{aligned}$$

Add up the equations and replace $a_0 = 0$ to obtain the solution

$$a_n = 2n.$$

This is quite a savings in the number of arithmetic operations to evaluate a polynomial of degree n. For example, if $n = 30$, then poly uses only 60 operations compared with 494 operations using the method discussed in Example 5 of Section 5.2.

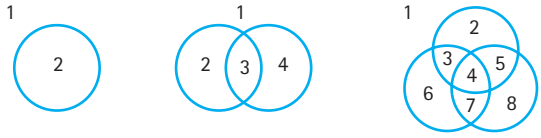

Figure 5.7 1-, 2-, and 3-ovals.

The n-Ovals Problem

Suppose we are given the following sequence of three numbers:

$$2,\ 4,\ 8.$$

What is the next number in the sequence? The problem below might make you think about your answer.

The n-Ovals Problem
Suppose that n ovals (an oval is a closed curve that does not cross over itself) are drawn on the plane such that no three ovals meet in a point and each pair of ovals intersects in exactly two points. How many distinct regions of the plane are created by n ovals?

For example, the diagrams in Figure 5.7 show the cases for one, two, and three ovals.

If we let r_n denote the number of distinct regions of the plane for n ovals, then it's clear that the first three values are

$$\begin{aligned} r_1 &= 2, \\ r_2 &= 4, \\ r_3 &= 8. \end{aligned}$$

What is the value of r_4? Is it 16? Check it out. To find r_n, consider the following description: $n-1$ ovals divide the region into r_{n-1} regions. The nth oval will meet each of the previous $n-1$ ovals in $2(n-1)$ points. So the nth oval will itself be divided into $2(n-1)$ arcs. Each of these $2(n-1)$ arcs splits some region in two. Therefore, we add $2(n-1)$ regions to r_{n-1} to obtain r_n. This gives us the following recurrence.

$$\begin{aligned} r_1 &= 2, \\ r_n &= r_{n-1} + 2\,(n-1)\,. \end{aligned}$$

We'll solve it by the substitution technique:

$$\begin{aligned} r_n &= r_{n-1} + 2(n-1) \\ &= r_{n-2} + 2(n-2) + 2(n-1) \\ &\vdots \\ &= r_1 + 2(1) + \cdots + 2(n-2) + 2(n-1) \\ &= 2 + 2(1) + \cdots + 2(n-2) + 2(n-1). \end{aligned}$$

Now we can find a closed from for r_n.

$$\begin{aligned} r_n &= 2 + 2(1) + \cdots + 2(n-2) + 2(n-1) \\ &= 2 + 2(1 + 2 + \cdots (n-2) + (n-1)) \\ &= 2 + 2\sum_{i=1}^{n-1} i \\ &= 2 + 2\frac{(n-1)(n)}{2} \\ &= n^2 - n + 2. \end{aligned}$$

For example, we can use this formula to calculate $r_4 = 14$. Therefore, the sequence of numbers 2, 4, 8 could very well be the first three numbers in the following sequence for the n-ovals problem.

2, 4, 8, 14, 22, 32, 44, 62, 74, 92,....

5.4.2 Generating Functions

For some recurrence problems we need to find new techniques. For example, suppose we wish to find a closed form for the nth Fibonacci number F_n, which is defined by the recurrence system

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2} \quad (n \geq 2). \end{aligned}$$

We can't use the cancellation technique with this system because F occurs twice on the right side of the general equation. This problem belongs to a large class of problems that need a more powerful technique.

The technique that we present comes from the simple idea of equating the coefficients of two polynomials. For example, suppose we have the following equation.

$$a + bx + cx^2 = 4 + 7x^2.$$

We can solve for a, b, and c by equating coefficients to yield $a = 4$, $b = 0$, and $c = 7$. We'll extend this idea to expressions that have infinitely many terms of the form $a_n\ x^n$ for each natural number n. Let's get to the definition.

Definition of Generating Function

The *generating function* for the infinite sequence $a_0, a_1, \ldots, a_n, \ldots$ is the following infinite expression, which is also called a formal power series or an infinite polynomial:

$$\begin{aligned} A(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n + \cdots \\ &= \sum_{n=0}^{\infty} a_n x^n. \end{aligned}$$

Two generating functions may be added by adding the corresponding coefficients. Similarly, two generating functions may be multiplied by extending the rule for multiplying regular polynomials. In other words, multiply each term of one generating function by every term of the other generating function, and then add up all the results. Two generating functions are equal if their corresponding coefficients are equal.

We'll be interested in those generating functions that have closed forms. For example, let's consider the following generating function for the infinite sequence 1, 1,. . . , 1, . . . :

$$\sum_{n=0}^{\infty} x^n.$$

This generating function is often called a *geometric series*, and its closed form is given by the following formula.

Geometric Series Generating Function (5.14)

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n.$$

To justify equation (5.14), multiply both sides of the equation by 1 – x.

Using a Generating Function Formula

But how can we use this formula to solve recurrences? The idea, as we shall see, is to create an equation in which $A(x)$ is the unknown, solve for $A(x)$, and hope that our solution has a nice closed form. For example, if we find that

$$A(x) = \frac{1}{1-2x},$$

then we can rewrite it using (5.14) in the following way.

$$A(x) = \frac{1}{1-2x} = \frac{1}{1-(2x)} = \sum_{n=0}^{\infty} (2x)^n = \sum_{n=0}^{\infty} 2^n x^n.$$

Now we can equate coefficients to obtain the solution $a_n = 2^n$. In other words, the solution sequence is 1, 2, 4,..., 2^n, ...

Finding a Generating Function Formula

How do we obtain the closed form for $A(x)$? It's a four-step process, and we'll present it with an example. Suppose we want to solve the following recurrence:

$$\begin{aligned} a_0 &= 0, \\ a_1 &= 1, \\ a_n &= 5a_{n-1} - 6a_{n-2} \quad (n \geq 2). \end{aligned} \tag{5.15}$$

Step 1

Use the general equation in the recurrence to write an infinite polynomial with coefficients a_n. We start the index of summation at 2 because the general equation in (5.15) holds for $n \geq 2$. Thus we obtain the following equation:

$$\begin{aligned} \sum_{n=2}^{\infty} a_n x^n &= \sum_{n=2}^{\infty} (5a_{n-1} - 6a_{n-2}) x^n \\ &= \sum_{n=2}^{\infty} 5a_{n-1}x^n - \sum_{n=2}^{\infty} 6a_{n-2}x^n \\ &= 5\sum_{n=2}^{\infty} a_{n-1}x^n - 6\sum_{n=2}^{\infty} a_{n-2}x^n \end{aligned} \tag{5.16}$$

We want to solve for $A(x)$ from this equation. Therefore, we need to transform each infinite polynomial in (5.16) into an expression containing $A(x)$. To do this, notice that the left-hand side of (5.16) can be written as

$$\begin{aligned} \sum_{n=2}^{\infty} a_n x^n &= A(x) - a_0 - a_1 x \\ &= A(x) - x \quad \text{(substitute for } a_0 \text{ and } a_1\text{)}. \end{aligned}$$

The first infinite polynomial on the right side of (5.16) can be written as

$$\begin{aligned}\sum_{n=2}^{\infty} a_{n-1}x^n &= \sum_{n=1}^{\infty} a_n x^{n-1} \quad \text{(by a change of indices)}\\ &= x\sum_{n=1}^{\infty} a_n x^n\\ &= x\left(A\left(x\right)-a_0\right)\\ &= xA\left(x\right).\end{aligned}$$

The second infinite polynomial on the right side of (5.16) can be written as

$$\begin{aligned}\sum_{n=2}^{\infty} a_{n-2}x^n &= \sum_{n=0}^{\infty} a_n x^{n+2} \quad \text{(by a change of indices)}\\ &= x^2\sum_{n=0}^{\infty} a_n x^n\\ &= x^2A\left(x\right).\end{aligned}$$

Thus Equation (5.16) can be rewritten in terms of $A(x)$ as

$$A(x) - x = 5xA(x) - 6x^2A(x). \tag{5.17}$$

Step 1 can often be done equationally by starting with the definition of $A(x)$ and continuing until an equation involving $A(x)$ is obtained. For this example the process goes as follows:

$$\begin{aligned}A\left(x\right) &= \sum_{n=0}^{\infty} a_n x^n\\ &= a_0 + a_1x + \sum_{n=2}^{\infty} a_n x^n\\ &= x + \sum_{n=2}^{\infty} a_n x^n\\ &= x + \sum_{n=2}^{\infty} \left(5a_{n-1} - 6a_{n-2}\right)x^n\\ &= x + 5\sum_{n=0}^{\infty} a_{n-1}x^n - 6\sum_{n=2}^{\infty} a_{n-2}x^n\\ &= x + 5x\left(A\left(x\right)-a_0\right) - 6x^2A\left(x\right)\\ &= x + 5xA\left(x\right) - 6x^2A\left(x\right).\end{aligned}$$

Step 2

Solve the equation for $A(x)$ and try to transform the result into an expression containing closed forms of known generating functions. We solve Equation (5.17) by isolating $A(x)$ as follows:

$$A(x)(1 - 5x + 6x^2) = x.$$

Therefore, we can solve for $A(x)$ and try to obtain known closed forms, which can then be replaced by generating functions:

$$\begin{aligned}
A(x) &= \frac{x}{1-5x+6x^2} \\
&= \frac{x}{(2x-1)(3x-1)} \\
&= \frac{1}{2x-1} - \frac{1}{3x-1} && \text{(partial fractions)} \\
&= -\frac{1}{1-2x} - \frac{1}{1-3x} && \text{(put into the form } \tfrac{1}{1-t}\text{)} \\
&= -\sum_{n=0}^{\infty} (2x)^n + \sum_{n=0}^{\infty} (3x)^n \\
&= -\sum_{n=0}^{\infty} 2^n x^n + \sum_{n=0}^{\infty} 3^n x^n \\
&= \sum_{n=0}^{\infty} (-2^n + 3^n) x^n.
\end{aligned}$$

Step 3

Equate coefficients, and obtain the result. In other words, we equate the original definition for $A(x)$ and the form of $A(x)$ obtained in Step 2:

$$\sum_{n=0}^{\infty} a_n x^n = \sum_{n=0}^{\infty} (-2^n + 3^n) x^n.$$

These two infinite polynomials are equal if and only if the corresponding coefficients are equal. Equating the coefficients, we obtain the following closed form for a_n:

$$a_n = 3^n - 2^n \quad \text{for } n \geq 0. \tag{5.18}$$

Step 4 (Check the answer)

To make sure that no mistakes were made in Steps 1 to 3, we should check to see whether (5.18) is the correct answer to (5.15). Since the recurrence has two

basis cases, we'll start by verifying the special cases for $n = 0$ and $n = 1$. These cases are verified below:

$$a_0 = 3^0 - 2^0 = 0,$$
$$a_1 = 3^1 - 2^1 = 1.$$

Now verify that (5.18) satisfies the general case of (5.15) for $n \geq 2$. We'll start on the right side of (5.15) and substitute (5.18) to obtain the left side of (5.15).

$$\begin{aligned} 5a_{n-1} - 6a_{n-2} &= 5\left(3^{n-1} - 2^{n-1}\right) - 6\left(3^{n-2} - 2^{n-2}\right) && \text{(substitution)} \\ &= 3^n - 2^n && \text{(simplification)} \\ &= a_n. \end{aligned}$$

An Aside on Partial Fractions

Let's recall a few facts about partial fractions. Suppose we are given the following quotient of two polynomials $p(x)$ and $q(x)$:

$$\frac{p(x)}{q(x)},$$

where the degree of $p(x)$ is less than the degree of $q(x)$. The first thing to do is factor $q(x)$ into a product of linear and/or quadratic polynomials that can't be factored further (say, over the real numbers). Therefore, each factor of $q(x)$ has one of the following forms:

$$ax + b \quad \text{or} \quad cx^2 + dx + e.$$

The *partial fraction* representation of

$$\frac{p(x)}{q(x)}$$

is a sum of terms, where each term in the sum is a quotient as follows:

Partial Fractions

1. If the linear polynomial $ax + b$ is repeated k times as a factor of $q(x)$, then add the following terms to the partial fraction representation, where $A_1, \ldots, A_k$ are constants to be determined:

$$\frac{A_1}{ax+b} + \frac{A_2}{(ax+b)^2} + \cdots + \frac{A_k}{(ax+b)^k}.$$

Continued ➧

2. If the quadratic polynomial $cx^2 + dx + e$ is repeated k times as a factor of $q(x)$, then add the following terms to the partial fraction representation, where A_i and B_i are constants to be determined.

$$\frac{A_1x + B_1}{cx^2 + dx + e} + \frac{A_2x + B_2}{(cx^2 + dx + e)^2} + \cdots + \frac{A_kx + B_k}{(cx^2 + dx + e)^k}.$$

example

5.28 Sample Partial Fractions

Here are a few samples of partial fractions that can be obtained from the two rules.

$$\frac{x-1}{x(x-2)(x+1)} = \frac{A}{x} + \frac{B}{x-2} + \frac{C}{x+1},$$

$$\frac{x^3-1}{x^2(x-2)^3} = \frac{A}{x} + \frac{B}{x^2} + \frac{C}{x-2} + \frac{D}{(x-2)^2} + \frac{E}{(x-2)^3},$$

$$\frac{x^2}{(x-1)(x^2+2x+1)} = \frac{A}{x-1} + \frac{Bx+C}{x^2+2x+1},$$

$$\frac{x}{(x-1)(x^2+1)^2} = \frac{A}{x-1} + \frac{Bx+C}{x^2+1} + \frac{Dx+C}{(x^2+1)^2}.$$

end example

To determine the constants in a partial fraction representation, we can solve simultaneous equations. Suppose there are n constants to be found. Then we need to create n equations. To create an equation, pick some value for x, with the restriction that the value for x does not make any denominator zero. Do this for n distinct values for x. Then solve the resulting n equations. For example, in Step 2 of the generating function example we wrote down the following equalities.

$$\begin{aligned} A(x) &= \frac{x}{1-5x+6x^2} \\ &= \frac{x}{(2x-1)(3x-1)} \\ &= \frac{1}{2x-1} - \frac{1}{3x-1}. \end{aligned}$$

The last equality is the result of partial fractions. Here's how we got it. First we write the partial fraction representation

$$\frac{x}{(2x-1)(3x-1)} = \frac{A}{2x-1} + \frac{B}{3x-1}.$$

Then we create two equations in A and B by letting $x = 0$ and $x = 1$.

$$0 = -A - B$$
$$1/2 = A + (1/2)\,B$$

Solving for A and B, we get $A = 1$ and $B = -1$. This yields the desired equality

$$\frac{x}{(2x-1)\,(3x-1)} = \frac{1}{2x-1} - \frac{1}{3x-1}.$$

A Final Note on Partial Fractions

If the degree of the numerator $p(x)$ is greater than or equal to the degree of $q(x)$, then a simple division of $p(x)$ by $q(x)$ will yield an equation of the form

$$\frac{p\,(x)}{q\,(x)} = s\,(x) + \frac{p'\,(x)}{q'\,(x)}.$$

where the degree of is less than the degree of $q'(x)$. Then we can apply partial fractions to the quotient

$$\frac{p'\,(x)}{q'\,(x)}.$$

More Generating Functions

There are many useful generating functions. Since our treatment is not intended to be exhaustive, we'll settle for listing two more generating functions that have many applications.

Two More Useful Generating Functions

$$\frac{1}{(1-x)^{k+1}} = \sum_{n=0}^{\infty} \binom{k+n}{n} x^n \quad \text{for } k \in \mathbb{N}. \tag{5.19}$$

$$(1+x)^r = \sum_{n=0}^{\infty} \left(\frac{r\,(r-1)\cdots(r-n+1)}{n!} \right) x^n \quad \text{for } r \in \mathbb{R}. \tag{5.20}$$

The numerator of the coefficient expression for the nth term in (5.20) contains a product of n numbers. When $n = 0$, we use the convention that a vacuous product—of zero numbers—has the value 1. Therefore the 0th term of (5.20) is $\frac{1}{0!} = 1$. So the first few terms of (5.20) look like the following:

$$(1+x)^r = 1 + rx + \frac{r\,(r-1)}{2}x^2 + \frac{r\,(r-1)\,(r-2)}{6}x^3 + \cdots.$$

The Problem of Parentheses

Suppose we want to find the number of ways to parenthesize the expression

$$t_1 + t_2 + \ldots + t_{n-1} + t_n \tag{5.21}$$

so that a parenthesized form of the expression reflects the process of adding two terms. For example, the expression $t_1 + t_2 + t_3 + t_4$ has several different forms as shown in the following expressions:

$$\begin{gathered}((t_1 + t_2) + (t_3 + t_4)) \\ (t_1 + (t_2 + (t_3 + t_4))) \\ (t_1 + ((t_2 + t_3) + t_4)) \\ \vdots\end{gathered}$$

To solve the problem, we'll let b_n denote the total number of possible parenthesizations for an n-term expression. Notice that if $1 \leq k \leq n - 1$, then we can split the expression (5.21) into two subexpressions as follows:

$$t_1 + \cdots + t_{n-k} \quad \text{and} \quad t_{n-k+1} + \cdots + t_n. \tag{5.22}$$

So there are $b_{n-k}b_k$ ways to parenthesize the expression (5.21) if the final + is placed between the two subexpressions (5.22). If we let k range from 1 to $k - 1$, we obtain the following formula for b_n when $n \geq 2$:

$$b_n = b_{n-1}b_1 + b_{n-2}b_2 + \ldots + b_2b_{n-2} + b_1b_{n-1}. \tag{5.23}$$

But we need $b_1 = 1$ for (5.23) to make sense. It's OK to make this assumption because we're concerned only about expressions that contain at least two terms. Similarly, we can let $b_0 = 0$. So we can write down the recurrence to describe the solution as follows:

$$\begin{aligned} &b_0 = 0, \\ &b_1 = 1, \\ &b_n = b_nb_0 + b_{n-1}b_1 + \cdots + b_1b_{n-1} + b_0b_n \quad (n \geq 2)\,. \end{aligned} \tag{5.24}$$

Notice that this system cannot be solved by the cancellation method. Let's try generating functions. Let $B(x)$ be the generating function for the sequence

$$b_0,\ b_1,\ldots,\ b_n,\ \ldots.$$

So $B\,(x) = \sum_{n=0}^{\infty} b_n x^n$. Now let's try to apply the four-step procedure for generating functions. First we use the general equation in the recurrence to introduce the partial (since $n \geq 2$) generating function

$$\sum_{n=2}^{\infty} b_n x^n = \sum_{n=2}^{\infty} (b_nb_0 + b_{n-1}b_1 + \cdots + b_1b_{n-1} + b_0b_n)\, x^n. \tag{5.25}$$

Now the left-hand side of (5.25) can be written in terms of $B(x)$:

$$\begin{aligned}\sum_{n=2}^{\infty} b_n x^n &= B(x) - b_1 x - b_0 \\ &= B(x) - x \quad (\text{since } b_0 = 0 \text{ and } b_1 = 1).\end{aligned}$$

Before we discuss the right hand-side of equation (5.25), notice that we can write the product

$$\begin{aligned}B(x)\,B(x) &= \left(\sum_{n=0}^{\infty} b_n x^n\right)\left(\sum_{n=0}^{\infty} b_n x^n\right) \\ &= \sum_{n=0}^{\infty} c_n x^n,\end{aligned}$$

where $c_0 = b_0\ b_0$ and for $n > 0$,

$$c_n = b_n\ b_0 + b_{n-1}\ b_1 + \cdots + b_1\ b_{n-1} + b_0\ b_n.$$

So the right-hand side of Equation (5.25) can be written as

$$\begin{aligned}\sum_{n=2}^{\infty} (b_n b_0 + b_{n-1} b_1 + \cdots + b_1 b_{n-1} + b_0 b_n)\, x^n \\ &= B(x)\,B(x) - b_0 b_0 - (b_1 b_0 + b_0 b_1)\, x \\ &= B(x)\,B(x) \quad (\text{since } b_0 = 0).\end{aligned}$$

Now Equation (5.25) can be written in simplified form as

$$B(x) - x = B(x)\,B(x) \quad \text{or} \quad B(x)^2 - B(x) + x = 0.$$

Now, thinking of $B(x)$ as the unknown, the equation is a quadratic equation with two solutions:

$$B(x) = \frac{1 \pm \sqrt{1 - 4x}}{2}.$$

Notice that $\sqrt{1-4x}$ is the closed form for a binomial generating function obtained from (5.20), where $r = \frac{1}{2}$. Thus we can write

$$\begin{aligned}
\sqrt{1-4x} &= (1+(4x))^{\frac{1}{2}} \\
&= \sum_{n=0}^{\infty} \frac{\frac{1}{2}\left(\frac{1}{2}-1\right)\left(\frac{1}{2}-2\right)\cdots\left(\frac{1}{2}-n+1\right)}{n!}(-4x)^n \\
&= \sum_{n=0}^{\infty} \frac{\frac{1}{2}\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)\cdots\left(-\frac{2n-3}{2}\right)}{n!}(-2)^n\,2^n x^n \\
&= 1+\sum_{n=1}^{\infty} \frac{(-1)(1)(3)\cdots(2n-3)}{n!}2^n x^n \\
&= 1+\sum_{n=1}^{\infty} \left(-\frac{2}{n}\right)\binom{2n-2}{n-1}x^n.
\end{aligned}$$

Expansion of the last equality is left as an exercise. Notice that, for $n \geq 1$, the coefficient of x^n is negative in this generating function. In other words, the nth term ($n \geq 1$) of the generating function $\sqrt{1-4x}$ for always has a negative coefficient. Since we need positive values for b_n, we must choose the following solution of our quadratic equation:

$$B(x) = \frac{1}{2} - \frac{1}{2}\sqrt{1-4x}.$$

Putting things together, we can write our desired generating function as follows:

$$\begin{aligned}
\sum_{n=0}^{\infty} b_n x^n = B(x) &= \frac{1}{2} - \frac{1}{2}\sqrt{1-4x} \\
&= \frac{1}{2} - \frac{1}{2}\left\{1+\sum_{n=1}^{\infty}\left(-\frac{2}{n}\right)\binom{2n-2}{n-1}x^n\right\} \\
&= 0+\sum_{n=1}^{\infty}\frac{1}{n}\binom{2n-2}{n-1}x^n.
\end{aligned}$$

Now we can finish the job by equating coefficients to obtain the following solution:

$$b_n = \text{if } n = 0 \text{ then } 0 \text{ else } \frac{1}{n}\binom{2n-2}{n-1}.$$

The Problem of Binary Trees

Suppose we want to find, for any natural number n, the number of structurally distinct binary trees with n nodes. Let b_n denote this number. We can figure out a few values by experiment. For example, since there is one empty binary tree and one binary tree with a single node, we have $b_0 = 1$ and $b_1 = 1$. It's also easy to see that $b_2 = 2$, and for $n = 3$ we see after a few minutes that $b_3 = 5$.

Let's consider b_n for $n \geq 1$. A tree with n nodes has a root and two subtrees whose combined nodes total $n - 1$. For each k in the interval $0 \leq k \leq n - 1$ there are b_k left subtrees of size k and b_{n-1-k} distinct right subtrees of size $n - 1 - k$. So for each k there are $b_k b_{n-1-k}$ distinct binary trees with n nodes. Therefore, the number b_n of binary trees can be given by the sum of these products as follows:

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_k b_{n-k} + \cdots + b_{n-2}\, b_1 + b_{n-1} b_0.$$

Now we can write down the recurrence to describe the solution as follows:

$$\begin{aligned} b_0 &= 1, \\ b_n &= b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_k b_{n-k} + \cdots + b_{n-2} b_1 + b_{n-1} b_0 \quad (n \geq 1) \end{aligned}$$

Notice that this system cannot be solved by the cancellation method. Let's try generating functions. Let $B(x)$ be the generating function for the sequence

$$b_0,\ b_1, \ldots,\ b_n,\ \ldots.$$

So $B(x) = \sum_{n=0}^{\infty} b_n x^n$. Now let's try to apply the four-step procedure for generating functions. First we use the general equation in the recurrence to introduce the partial (since $n \geq 1$) generating function

$$\sum_{n=1}^{\infty} b_n x^n = \sum_{n=1}^{\infty} \left(b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_{n-2} b_{n-1} b_0\right) x^n. \tag{5.26}$$

Now the left-hand side of (5.26) can be written in terms of $B(x)$.

$$\begin{aligned} \sum_{n=1}^{\infty} b_n x^n &= B(x) - b_0 \\ &= B(x) - 1 \quad (\text{since } b_0 = 1). \end{aligned}$$

Before we discuss the right hand-side of equation (5.26), notice that we can write the product

$$\begin{aligned} B(x)\, B(x) &= \left(\sum_{n=0}^{\infty} b_n x^n\right)\left(\sum_{n=0}^{\infty} b_n x^n\right) \\ &= \sum_{n=0}^{\infty} c_n x^n, \end{aligned}$$

where $c_0 = b_0\ b_0$ and for $n > 0$,

$$c_n = b_0 b_n + b_1 b_{n-1} + \cdots + b_{n-1} b_1 + b_n b_0$$

So the right-hand side of equation (5.26) can be written as

$$\begin{aligned}
&\sum_{n=1}^{\infty}\left(b_0b_{n-1}+b_1b_{n-2}+\cdots+b_{n-2}b_1+b_{n-1}b_0\right)x^n\\
&\quad=\sum_{n=0}^{\infty}\left(b_0b_n+b_1b_{n-1}+\cdots+b_{n-1}b_1+b_nb_0\right)x^{n+1}\\
&\quad=x\sum_{n=0}^{\infty}\left(b_0b_n+b_1b_{n-1}+\cdots+b_{n-1}b_1+b_nb_0\right)x^n\\
&\quad=xB\left(x\right)B\left(x\right)
\end{aligned}$$

Now Equation (5.26) can be written in simplified form as

$$B\left(x\right)-1=xB\left(x\right)B\left(x\right)\quad\text{or}\quad xB\left(x\right)^2-B\left(x\right)+1=0.$$

Now, thinking of $B(x)$ as the unknown, the equation is a quadratic equation with two solutions:

$$B\left(x\right)=\frac{1\pm\sqrt{1-4x}}{2x}.$$

Notice that is $\sqrt{1-4x}$ the closed form for a binomial generating function obtained from (5.20), where $r=\frac{1}{2}$. Thus we can write

$$\begin{aligned}
\sqrt{1-4x}&=\left(1+\left(-4x\right)\right)^{\frac{1}{2}}\\
&=\sum_{n=0}^{\infty}\frac{\frac{1}{2}\left(\frac{1}{2}-1\right)\left(\frac{1}{2}-2\right)\cdots\left(\frac{1}{2}-n+1\right)}{n!}\left(-4x\right)^n\\
&=\sum_{n=0}^{\infty}\frac{\frac{1}{2}\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)\cdots\left(-\frac{2n-3}{2}\right)}{n!}\left(-2\right)^n2^nx^n\\
&=1+\sum_{n=1}^{\infty}\frac{\left(-1\right)\left(1\right)\left(3\right)\cdots\left(2n-3\right)}{n!}2^nx^n\\
&=1+\sum_{n=1}^{\infty}\left(-\frac{2}{n}\right)\binom{2n-2}{n-1}x^n.
\end{aligned}$$

Notice that for $n\geq 1$ the coefficient of x^n is negative in this generating function. In other words, the nth term $(n\geq 1)$ of the generating function for $\sqrt{1-4x}$ always has a negative coefficient. Since we need positive values for b_n, we must choose the following solution of our quadratic equation:

$$B\left(x\right)=\frac{1-\sqrt{1-4x}}{2x}.$$

Putting things together, we can write our desired generating function as follows:

$$\begin{aligned}\sum_{n=0}^{\infty} b_n x^n = B(x) &= \frac{1-\sqrt{1-4x}}{2x} = \frac{1}{2x}\left(1-\sqrt{1-4x}\right)\\ &= \frac{1}{2x}\sum_{n=1}^{\infty}\left(\frac{2}{n}\right)\binom{2n-2}{n-1}x^n\\ &= \sum_{n=1}^{\infty}\frac{1}{n}\binom{2n-2}{n-1}x^{n+1}\\ &= \sum_{n=0}^{\infty}\frac{1}{n+1}\binom{2n}{n}x^n.\end{aligned}$$

Now we can finish the job by equating coefficients to obtain

$$b_n = \frac{1}{n+1}\binom{2n}{n}.$$

Exercises

Simple Recurrences

1. Solve each of the following recurrences by the substitution technique and the cancellation technique. Put each answer in closed form (no ellipsis allowed).

a. $a_1 = 0$, $a_n = a_{n-1} + 4$.

b. $a_1 = 0$, $a_n = a_{n-1} + 2n$.

c. $a_0 = 1$, $a_n = 2a_{n-1} + 3$.

2. For each of the following functions, find a recurrence to describe the number of times the cons operation :: is called. Solve each recurrence.

a. $\text{cat}(L, M) =$ if $L = \langle\,\rangle$ then M else $\text{head}(L) :: \text{cat}(\text{tail}(L), M)$.

b. $\text{dist}(x, L) =$ if $L = \langle\,\rangle$ then $\langle\,\rangle$
else $(x :: \text{head}(L) :: \langle\,\rangle) :: \text{dist}(x, \text{tail}(L))$.

c.
```
power(L) = if L = ⟨ ⟩ then return ⟨ ⟩ :: ⟨ ⟩
           else
               A := power(tail(L));
               B := dist(head(L), A);
               C := map(::, B);
               return cat(A, C)
           fi
```

3. (*Towers of Hanoi*). The *Towers of Hanoi* puzzle was invented by Lucas in 1883. It consists of three stationary pegs with one peg containing a stack of

n disks that form a tower (each disk has a hole in the center for the peg) in which each disk has a smaller diameter than the disk below it. The problem is to move the tower to one of the other pegs by transferring one disk at a time from one peg to another peg, no disk ever being placed on a smaller disk. Find the minimum number of moves H_n to do the job.

Hint: It takes 0 moves to transfer a tower of 0 disks and 1 move to transfer a tower of 1 disk. So $H_0 = 0$ and $H_1 = 1$. Try it out for $n = 2$ and $n = 3$ to get the idea. Then try to find a recurrence relation for the general term H_n as follows: Move the tower consisting of the top $n - 1$ disks to the nonchosen peg; then move the bottom disk to the chosen peg; then move the tower of $n - 1$ disks onto the chosen peg.

4. (*Diagonals in a Polygon*). A diagonal in a polygon is a line from one vertex to another nonadjacent vertex. For example, a triangle doesn't have any diagonals because each vertex is adjacent to the other vertices. Find the number of diagonals in an n-sided polygon, where $n \geq 3$.

5. (*The n-Lines Problem*). Find the number of regions in a plane that are created by n lines, where no two lines are parallel and where no more than two lines intersect at any point.

Generating Functions

6. Given the generating function $A(x) = \sum_{n=0}^{\infty} a_n x^n$, find a closed form for the general term a_n for each of the following representations of $A(x)$.

a. $A(x) = \frac{1}{x-2} - \frac{2}{3x+1}$.

b. $A(x) = \frac{1}{2x+1} + \frac{3}{x+6}$.

c. $A(x) = \frac{1}{3x-2} - \frac{1}{(1-x)^2}$.

7. Use generating functions to solve each of the following recurrences.

a. $a_0 = 0$,
$a_1 = 4$,
$a_n = 2a_{n-1} + 3a_{n-2} \quad (n = 2)$.

b. $a_0 = 0$,
$a_1 = 1$,
$a_n = 7a_{n-1} - 12a_{n-2} \quad (n = 2)$.

c. $a_0 = 0$,
$a_1 = 1$,
$a_2 = 1$,
$a_n = 2a_{n-1} + a_{n-2} - 2a_{n-3} \quad (n = 3)$.

8. Use generating functions to solve each recurrence in Exercise 1. For those recurrences that do not have an a_0 term, assume that $a_0 = 0$.

Proofs and Challenges

9. Prove that the following equation holds for all positive integers n, in two different ways, as indicated:

$$\frac{(1)\,(1)\,(3)\cdots(2n-3)}{n!}2^n = \frac{2}{n}\binom{2n-2}{n-1}.$$

a. Use induction.

b. Transform the left side into the right side by "inserting" the missing even numbers in the numerator.

10. Find a closed form for the nth Fibonacci number defined by the following recurrence system.

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2} \quad (n \geq 2). \end{aligned}$$

5.5 Comparing Rates of Growth

Sometimes it makes sense to approximate the number of steps required to execute an algorithm because of the difficulty involved in finding a closed form for an expression or the difficulty in evaluating an expression. To approximate one function with another function, we need some way to compare them. That's where "rate of growth" comes in. We want to give some meaning to statements like "f has the same growth rate as g" and "f has a lower growth rate than g."

For our purposes we will consider functions whose domains and codomains are subsets of the real numbers. We'll examine the asymptotic behavior of two functions f and g by comparing $f(n)$ and $g(n)$ for large positive values of n (i.e., as n approaches infinity).

5.5.1 Big Theta

Let's begin by discussing the meaning of the statement "f has the same growth rate as g."

A function f has the *same growth rate* as g (or f has the *same order* as g if we can find a number m and two positive constants c and d such that

$$c\,|g\,(n)| \leq |f\,(n)| \leq d\,|g\,(n)| \quad \text{for all } n \geq m.$$

In this case we write $f(n) = \Theta(g(n))$ and say that $f(n)$ is *big theta* of $g(n)$.

It's easy to verify that the relation "has the same growth rate as" is an equivalence relation. In other words, the following three properties hold for all functions.

$$f(n) = \Theta(f(n)).$$
$$\text{If } f(n) = \Theta(g(n)), \text{ then } g(n) = \Theta(f(n)).$$
$$\text{If } f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)), \text{ then } f(n) = \Theta(h(n)).$$

If $f(n) = \Theta(g(n))$ and we also know that $g(n) \neq 0$ for all $n \geq m$, then we can divide the inequality (5.27) by $g(n)$ to obtain

$$c \leq \left| \frac{f(n)}{g(n)} \right| \leq d \quad \text{for all } n \geq m.$$

This inequality gives us a better way to think about "having the same growth rate." It tells us that the ratio of the two functions is always within a fixed bound beyond some point. We can always take this point of view for functions that count the steps of algorithms because they are positive valued.

Now let's see whether we can find some functions that have the same growth rate. To start things off, suppose f and g are proportional. This means that there is a nonzero constant c such that $f(n) = cg(n)$ for all n. In this case, definition (5.27) is satisfied by letting $d = c$. Thus we have the following statement.

Proportionality (5.28)
If two functions f and g are proportional, then $f(n) = \Theta(g(n))$.

example

5.29 The Log Function

Recall that log functions with different bases are proportional. In other words, if we have two bases $a > 1$ and $b > 1$, then

$$\log_a n = (\log_a b)(\log_b n) \quad \text{for all } n > 0.$$

So we can disregard the base of the log function when considering rates of growth. In other words, we have

$$\log_a n = \theta(\log_b n). \tag{5.29}$$

end example

It's interesting to note that two functions can have the same growth rate without being proportional. Here's an example.

example

5.30 Polynomials of the Same Degree

Let's show that $n^2 + n$ and n^2 have the same growth rate. The following inequality is true for all $n = 1$:

$$1n^2 = n^2 + n = 2n^2.$$

Therefore, $n^2 + n = \Theta(n^2)$.

end example

The following theorem gives us a nice tool for showing that two functions have the same growth rate.

Theorem (5.30)

$$\text{If } \lim_{n\to\infty} \frac{f(n)}{g(n)} = c \quad \text{where } c \neq 0 \text{ and } c \neq \infty, \text{ then } f(n) = \Theta(g(n)).$$

For example, the quotient $(25n^2 + n)/n^2$ approaches 25 as n approaches infinity. Therefore we can say that $25n^2 + n = \Theta(n^2)$.

We should note that the limit in (5.30) is not a necessary condition for $f(n) = \Theta(g(n))$. For example, suppose we let f and g be the two functions

$$f(n) = \text{if } n \text{ is odd then 2 else 4,}$$
$$g(n) = 2.$$

We can write $1 \cdot g(n) = f(n) = 2 \cdot g(n)$ for all $n = 1$. Therefore $f(n) = \Theta(g(n))$. But the quotient $f(n)/g(n)$ alternates between the two values 1 and 2. Therefore the limit of the quotient does not exist. Still the limit test (5.30) will work for the majority of functions that occur in analyzing algorithms.

Approximations can be quite useful for those of us who can't remember formulas that we don't use all the time. For example, we can write the sums from (5.2) in terms of Θ as follows:

Some Approximations

$$\sum_{i=1}^{n} i = \Theta(n^2). \tag{5.31}$$

$$\sum_{i=1}^{n} i^2 = \Theta(n^3). \tag{5.32}$$

Continued ➧

If $a \neq 1$, then $\sum_{n=0}^{n} a^i = \Theta\left(a^{n+1}\right)$. (5.33)

If $a \neq 1$, then $\sum_{n=0}^{n} ia^i = \Theta\left(na^{n+1}\right)$. (5.34)

$\sum_{n=1}^{n} i^k = \Theta\left(n^{k+1}\right)$. (5.35)

Notice that (5.31) and (5.32) are special cases of (5.35).

example **5.31 A Worst-Case Lower Bound for Sorting**

Let's clarify a statement that we made in Example 1 of Section 5.3. We showed that $\lceil \log_2 n! \rceil$ is the worst case lower bound for comparison sorting algorithms. But $\log n!$ is hard to calculate for even modest values of n. We stated that $\lceil \log_2 n! \rceil$ is approximately equal to $n \log_2 n$. Now we can make the following statement:

$$\log n! = \Theta(n \log n). \tag{5.36}$$

To prove this statement, we'll find some bounds on $\log n!$ as follows:

$$\begin{aligned} \log n! &= \log n + \log(n-1) + \cdots + \log 1 \\ &\leq \log n + \log n + \cdots + \log n && (n \text{ terms}) \\ &= n \log n. \end{aligned}$$

$$\begin{aligned} \log n! &= \log n + \log(n-1) + \cdots + \log 1 \\ &\geq \log n + \log(n-1) + \cdots + \log(\lceil n/2 \rceil) && (\lceil n/2 \rceil \text{ terms}) \\ &\geq \log \lceil n/2 \rceil + \cdots + \log \lceil n/2 \rceil && (\lceil n/2 \rceil \text{ terms}) \\ &= \lceil n/2 \rceil \log \lceil n/2 \rceil \\ &\geq (n/2) \log(n/2). \end{aligned}$$

So we have the inequality:

$$(n/2)\log(n/2) \leq \log n! \leq n \log n.$$

It's easy to see (i.e., as an exercise) that if $n > 4$, then $(1/2) \log n < \log(n/2)$. Therefore, we have the following inequality for $n > 4$:

$$(1/2)\ (n \log n) \leq (n/2) \log(n/2) \leq \log n! = n \log n.$$

So there are nonzero constants 1/2 and 1 and the number 4 such that

$$(1/2)\ (n \log n) \leq \log n! \leq (1)(n \log n) \text{ for all } n > 4.$$

This tells us that $\log n! = \Theta(n \log n)$.

end example

An important approximation to $n!$ is *Stirling's formula*—named for the mathematician James Stirling (1692–1770)—which is written as

$$n! = \Theta\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right). \tag{5.37}$$

Let's see how we can use big theta to discuss the approximate performance of algorithms. For example, the worst-case performance of the binary search algorithm is $\Theta(\log\ n)$ because the actual value is $1 + \lfloor \log_2\ n \rfloor$. Both the average and worst-case performances of a linear sequential search are $\Theta(n)$ because the average number of comparisons is $(n+1)/2$ and the worst case number of comparisons is n.

For sorting algorithms that sort by comparison, the worst-case lower bound is $\lceil \log_2\ n! \rceil = \Theta(n \log\ n)$. Many sorting algorithms, like the simple sort algorithm in Example 6 of Section 5.2, have worst-case performance of $\Theta(n^2)$. The "dumbSort" algorithm, which constructs a permutation of the given list and then checks to see whether it is sorted, may have to construct all possible permutations before it gets the right one. Thus dumbSort has worst-case performance of $\Theta(n!)$. An algorithm called "heapsort" will sort any list of n items using at most $2n \log_2\ n$ comparisons. So heapsort is a $\Theta(n \log\ n)$ algorithm in the worst case.

5.5.2 Little Oh

Now let's discuss the meaning of the statement "f has a lower growth rate than g."

A function f has a *lower growth rate* than g (or f has *lower order* than g) if

$$\lim_{n\to\infty} \frac{f\,(n)}{g\,(n)} = 0. \tag{5.38}$$

In this case we write $f(n) = o\ (g(n))$ and say that f is *little oh* of g."

For example, the quotient approaches 0 as n goes to infinity. Therefore, $n = o(n^2)$, and we can say that n has lower order than n^2. For another example, if a and b are positive numbers such that $a < b$, then $a^n = o(b^n)$. To see this, notice that the quotient approaches 0 as n approaches infinity because $0 < a/b < 1$.

For those readers familiar with derivatives, the evaluation of limits can often be accomplished by using L'Hôpital's rule.

Theorem (5.39)
If $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$ or $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = 0$ and f and g are differentiable beyond some point, then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}.$$

example **5.32 Different Orders**

We'll show that $\log n = o(n)$. Since both n and $\log n$ approach infinity as n approaches infinity, we can apply (5.39) to $(\log n)/n$. Since we can write $\log n = (\log e)(\log_e n)$, it follows that the derivative of $\log n$ is $(\log e)(1/n)$. Therefore, we obtain the following equations:

$$\lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \frac{(\log e)\,(1/n)}{1} = 0.$$

So $\log n$ has lower order than n, and we can write $\log n = o(n)$.

end example

Let's list a hierarchy of some familiar functions according to their growth rates, where $f(n) \prec g(n)$ means that $f(n) = o(g(n))$:

$$1 \prec \log n \prec n \prec n \log n \prec n^2 \prec n^3 \prec 2^n \prec 3^3 \prec n! \prec n^n. \tag{5.40}$$

This hierarchy can help us compare different algorithms. For example, we would certainly choose an algorithm with running time $\Theta(\log n)$ over an algorithm with running time $\Theta(n)$.

5.5.3 Big Oh and Big Omega

Now let's look at a notation that gives meaning to the statement "the growth rate of f is bounded above by the growth rate of g." The standard notation to describe this situation is

$$f(n) = O(g(n)), \tag{5.41}$$

which we read as $f(n)$ is *big oh* of $g(n)$. The precise meaning of the notation $f(n) = O(g(n))$ is given by the following definition.

The Meaning of Big Oh (5.42)

The notation $f(n) = O(g(n))$ means that there are positive numbers c and m such that

$$|f(n)| \leq c\,|g(n)| \text{ for all } n \geq m.$$

example **5.33 Comparing Polynomials**

We'll show that $n^2 = O(n^3)$ and $5n^3 + 2n^2 = O(n^3)$. Since $n^2 \leq 1n^3$ for all $n \geq 1$, it follows that $n^2 = O(n^3)$. Since $5n^3 + 2n^2 \leq 7n^3$ for all $n \geq 1$, it follows that $5n^3 + 2n^2 = O(n^3)$.

end example

Now let's go the other way. We want a notation that gives meaning to the statement "the growth rate of f is bounded below by the growth rate of g." The standard notation to describe this situation is

$$f(n) = \Omega(g(n)), \tag{5.43}$$

which we can read as $f(n)$ is *big omega* of $g(n)$. The precise meaning of the notation $f(n) = \Omega\,(g(n))$ is given by the following definition.

The Meaning of Big Omega (5.44)

The notation $f(n) = \Omega\,(g(n))$ means that there are positive numbers c and m such that

$$|f(n)| \geq c\,|g(n)| \text{ for all } n \geq m.$$

example **5.34 Comparing Polynomials**

We'll show that $n^3 = \Omega\,(n^2)$ and $3n^2 + 2n = \Omega\,(n^2)$. Since $n^3 \geq 1n^2$ for all $n \geq 1$, it follows that $n^3 = \Omega\,(n^2)$. Since $3n^2 + 2n \geq 1n^2$ for all $n \geq 1$, it follows that $3n^2 + 2n = \Omega\,(n^2)$.

end example

Let's see how we can use the terms that we've defined so far to discuss algorithms. For example, suppose we have constructed an algorithm A to solve some problem P. Suppose further that we've analyzed A and found that it takes $5n^2$ operations in the worst case for an input of size n. This allows us to make a few general statements. First, we can say that the worst-case performance of A

is $\Theta(n^2)$. Second, we can say that an optimal algorithm for P, if one exists, must have a worst-case performance of $O(n^2)$. In other words, an optimal algorithm for P must do no worse than our algorithm A.

Continuing with our example, suppose some good soul has computed a worst-case theoretical lower bound of $\Theta(n \log n)$ operations for any algorithm that solves P. Then we can say that an optimal algorithm, if one exists, must have a worst-case performance of $\Omega(n \log n)$. In other words, an optimal algorithm for P can do no better than the given lower bound of $\Theta(n \log n)$.

Before we leave our discussion of approximate optimality, let's look at some other ways to use the symbols. The four symbols Θ, o, O, and Ω can also be used to represent terms within an expression. For example, the equation

$$h(n) = 4n^3 + O(n^2)$$

means that $h(n)$ equals $4n^3$ plus a term of order at most n^2. When used as part of an expression, big oh is the most popular of the four symbols because it gives a nice way to concentrate on those terms that contribute the most muscle.

We should also note that the four symbols Θ, o, O, and Ω can be formally defined to represent sets of functions. In other words, for a function g we define the following four sets:

$\Theta(g)$ is the set of functions with the same order as g;

$o(g)$ is the set of functions with lower order than g;

$O(g)$ is the set of functions of order bounded above by that of g;

$\Omega(g)$ is the set of functions of order bounded below by that of g.

When set representations are used, we can use an expression like $f(n) \in \Theta(g(n))$ to mean that f has the same order as g. The set representations also give some nice relationships. For example, we have the following relationships, where the subset relation is proper.

$$O(g(n)) \supset \Theta(g(n)) \cup o(g(n)),$$
$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n)).$$

Exercises

Calculations

1. Find a place to insert the function $\log \log n$ in the sequence (5.40).

2. For each each of the following functions f, find an appropriate place in the sequence (5.40).

a. $f(n) = \log 1 + \log 2 + \log 3 + \cdots + \log n$.

b. $f(n) = \log 1 + \log 2 + \log 4 + \cdots + \log 2^n$.

3. For each of the following values of n, calculate the following three numbers: the exact value of $n!$, Stirling's approximation (5.37) for the value of $n!$, and the difference between the two values.

a. $n = 5$. b. $n = 10$.

Proofs and Challenges

4. Find an example of an increasing function f such that $f(n) = \Theta(1)$.

5. Prove that the binary relation on functions defined by $f(n) = \Theta(g(n))$ is an equivalence relation.

6. For any constant $k > 0$, prove each of the following statements.

a. $\log(kn) = \Theta(\log n)$.

b. $\log(k + n) = \Theta(\log n)$.

7. Prove the following sequence of orders: $n \prec n \log n \prec n^2$.

8. For any constant k, show that n^k has lower order than 2^n.

9. Prove the following sequence of orders: $2^n \prec n! \prec n^n$.

10. Let $f(n) = O(h(n))$ and $g(n) = O(h(n))$. Prove each of the following statements.

a. $af(n) = O(h(n))$ for any real number a.

b. $f(n) + g(n) = O(h(n))$.

11. Show that each of the following subset relations is proper.

a. $O(g(n)) \supset \Theta(g(n)) \cup o(g(n))$.

b. $o(g(n)) \subset O(g(n)) - \Theta(g(n))$.

Hint: For example, let $g(n) = n$ and let $f(n) =$ if n is odd then 1 else n. Then show that $f(n) \in O(g(n)) - \Theta(g(n)) \cup o(g(n))$ for part (a) and show that $f(n) \in (O(g(n)) - \Theta(g(n))) - o(g(n))$ for part (b).

5.6 Chapter Summary

This chapter introduces some basic tools and techniques that are used to analyze algorithms. Analysis by worst-case running time is discussed. A lower bound is a value that can't be beat by any algorithm in a particular class. An algorithm is optimal if its performance matches the lower bound.

Counting problems often give rise to finite sums that need closed form solutions. Properties of sums together with summation notation provide us with techniques to find closed forms for many finite sums.

Two useful things to count are permutations, in which order is important, and combinations, in which order is not important. Pascal's triangle contains

formulas for combinations, which are the same as binomial coefficients. There are formulas to count permutations and combinations of bags; these allow repeated elements. Discrete probability—with finite sample spaces—gives us the tools to define the average-case performance of an algorithm.

Counting problems often give rise to recurrences. Some simple recurrences can be solved by either substitution or cancellation to obtain a finite sum, which can be then transformed into a closed form. The use of generating functions provides a powerful technique for solving recurrences.

Often it makes sense to find approximations for functions that describe the number of operations performed by an algorithm. The rates of growth of two functions can be compared in various ways—big theta, little oh, big oh, and big omega.

Notes

In this chapter we've just scratched the surface of techniques for manipulating expressions that crop up in counting things while analyzing algorithms. The book by Knuth [1968] contains the first account of a collection of techniques for the analysis of algorithms. The book by Graham, Knuth, and Patashnik [1989] contains a host of techniques, formulas, anecdotes, and further references to the literature. The book also introduces an alternative notation for working with sums, which often makes it easier to manipulate them without having to change the expressions for the upper and lower limits of summation. The notation is called Iverson's convention, and it is also described in the article by Knuth [1992].