CHAPTER 4

Combinatorics and Probability

In computer science we frequently need to count things and measure the likelihood of events. The science of counting is captured by a branch of mathematics called *combinatorics*. The concepts that surround attempts to measure the likelihood of events are embodied in a field called *probability theory*. This chapter introduces the rudiments of these two fields. We shall learn how to answer questions such as how many execution paths are there in a program, or what is the likelihood of occurrence of a given path?

4.1 What This Chapter Is About

We shall study combinatorics, or "counting," by presenting a sequence of increasingly more complex situations, each of which is represented by a simple paradigm problem. For each problem, we derive a formula that lets us determine the number of possible outcomes. The problems we study are:

- Counting assignments (Section 4.2). The paradigm problem is how many ways can we paint a row of n houses, each in any of k colors.
- Counting permutations (Section 4.3). The paradigm problem here is to determine the number of different orderings for n distinct items.
- Counting ordered selections (Section 4.4), that is, the number of ways to pick k things out of n and arrange the k things in order. The paradigm problem is counting the number of ways different horses can win, place, and show in a horse race.
- Counting the combinations of m things out of n (Section 4.5), that is, the selection of m from n distinct objects, without regard to the order of the selected objects. The paradigm problem is counting the number of possible poker hands.

- Counting permutations with some identical items (Section 4.6). The paradigm problem is counting the number of anagrams of a word that may have some letters appearing more than once.
- Counting the number of ways objects, some of which may be identical, can be distributed among bins (Section 4.7). The paradigm problem is counting the number of ways of distributing fruits to children.

In the second half of this chapter we discuss probability theory, covering the following topics:

- Basic concepts: probability spaces, experiments, events, probabilities of events.
- Conditional probabilities and independence of events. These concepts help us think about how observation of the outcome of one experiment, e.g., the drawing of a card, influences the probability of future events.
- Probabilistic reasoning and ways that we can estimate probabilities of combinations of events from limited data about the probabilities and conditional probabilities of events.

We also discuss some applications of probability theory to computing, including systems for making likely inferences from data and a class of useful algorithms that work "with high probability" but are not guaranteed to work all the time.

4.2 Counting Assignments

One of the simplest but most important counting problems deals with a list of items, to each of which we must assign one of a fixed set of values. We need to determine how many different assignments of values to items are possible.

Example 4.1. A typical example is suggested by Fig. 4.1, where we have four houses in a row, and we may paint each in one of three colors: red, green, or blue. Here, the houses are the "items" mentioned above, and the colors are the "values." Figure 4.1 shows one possible assignment of colors, in which the first house is painted red, the second and fourth blue, and the third green.

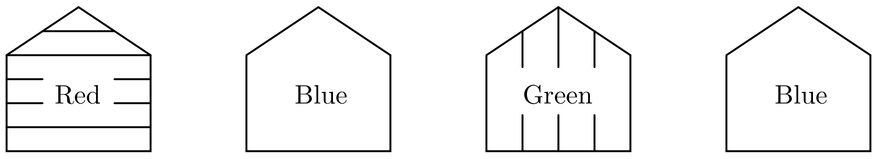


Fig. 4.1. One assignment of colors to houses.

To answer the question, "How many different assignments are there?" we first need to define what we mean by an "assignment." In this case, an assignment is a list of four values, in which each value is chosen from one of the three colors red, green, or blue. We shall represent these colors by the letters R, G, and B. Two such lists are *different* if and only if they differ in at least one position.

In the example of houses and colors, we can choose any of three colors for the first house. Whatever color we choose for the first house, there are three colors in which to paint the second house. There are thus nine different ways to paint the first two houses, corresponding to the nine different pairs of letters, each letter chosen from R, G, and B. Similarly, for each of the nine assignments of colors to the first two houses, we may select a color for the third house in three possible ways. Thus, there are $9 \times 3 = 27$ ways to paint the first three houses. Finally, each of these 27 assignments can be extended to the fourth house in 3 different ways, giving a total of $27 \times 3 = 81$ assignments of colors to the houses. ❖

The Rule for Counting Assignments

We can extend the above example. In the general setting, we have a list of n "items," such as the houses in Example 4.1. There is also a set of k "values," such as the colors in Example 4.1, any one of which can be assigned to an item. An *assignment* is a list of n values $(v_1, v_2, \ldots, v_n)$. Each of $v_1, v_2, \ldots, v_n$ is chosen to be one of the k values. This assignment assigns the value v_i to the ith item, for $i = 1, 2, \ldots, n$.

Assignment

There are k^n different assignments when there are n items and each item is to be assigned one of k values. For instance, in Example 4.1 we had $n = 4$ items, the houses, and $k = 3$ values, the colors. We calculated that there were 81 different assignments. Note that $3^4 = 81$. We can prove the general rule by an induction on n.

STATEMENT $S(n)$: The number of ways to assign any one of k values to each of n items is k^n.

BASIS. The basis is $n = 1$. If there is one item, we can choose any of the k values for it. Thus there are k different assignments. Since $k^1 = k$, the basis is proved.

INDUCTION. Suppose the statement $S(n)$ is true, and consider $S(n + 1)$, the statement that there are k^{n+1} ways to assign one of k values to each of $n+1$ items. We may break any such assignment into a choice of value for the first item and, for each choice of first value, an assignment of values to the remaining n items. There are k choices of value for the first item. For each such choice, by the inductive hypothesis there are k^n assignments of values to the remaining n items. The total number of assignments is thus $k \times k^n$, or k^{n+1}. We have thus proved $S(n+1)$ and completed the induction.

Figure 4.2 suggests this selection of first value and the associated choices of assignment for the remaining items in the case that $n + 1 = 4$ and $k = 3$, using as a concrete example the four houses and three colors of Example 4.1. There, we assume by the inductive hypothesis that there are 27 assignments of three colors to three houses.

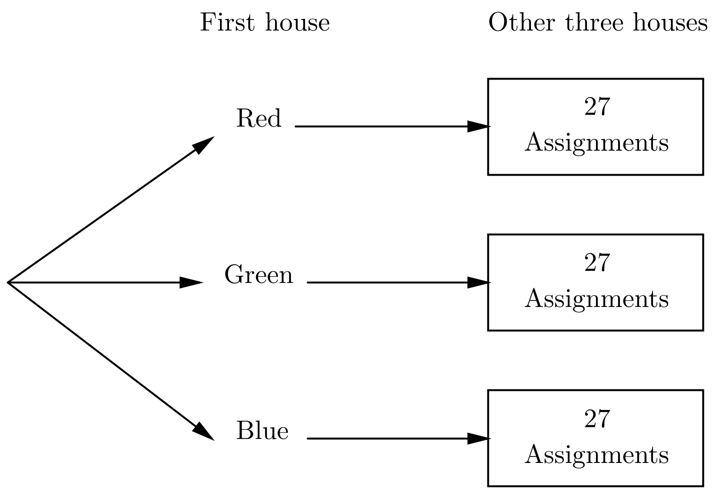


Fig. 4.2. The number of ways to paint 4 houses using 3 colors.

Counting Bit Strings

In computer systems, we frequently encounter strings of 0's and 1's, and these strings often are used as the names of objects. For example, we may purchase a computer with "64 megabytes of main memory." Each of the bytes has a name, and that name is a sequence of 26 *bits*, each of which is either a 0 or 1. The string of 0's and 1's representing the name is called a *bit string*.

Bit

Why 26 bits for a 64-megabyte memory? The answer lies in an assignment-counting problem. When we count the number of bit strings of length n, we may think of the "items" as the positions of the string, each of which may hold a 0 or a 1. The "values" are thus 0 and 1. Since there are two values, we have $k = 2$, and the number of assignments of 2 values to each of n items is 2^n.

If $n = 26$ — that is, we consider bit strings of length 26 — there are 2^{26} possible strings. The exact value of 2^{26} is 67,108,864. In computer parlance, this number is thought of as "64 million," although obviously the true number is about 5% higher. The box about powers of 2 tells us a little about the subject and tries to explain the general rules involved in naming the powers of 2.

EXERCISES

4.2.1: In how many ways can we paint

a) Three houses, each in any of four colors
b) Five houses, each in any of five colors
c) Two houses, each in any of ten colors

4.2.2: Suppose a computer password consists of eight to ten letters and/or digits. How many different possible passwords are there? Remember that an upper-case letter is different from a lower-case one.

4.2.3*: Consider the function `f` in Fig. 4.3. How many different values can `f` return?

```
int f(int x)
{
    int n;

    n = 1;
    if (x%2 == 0) n *= 2;
    if (x%3 == 0) n *= 3;
    if (x%5 == 0) n *= 5;
    if (x%7 == 0) n *= 7;
    if (x%11 == 0) n *= 11;
    if (x%13 == 0) n *= 13;
    if (x%17 == 0) n *= 17;
    if (x%19 == 0) n *= 19;
    return n;
}
```

Fig. 4.3. Function `f`.

4.2.4: In the game of "Hollywood squares," X's and O's may be placed in any of the nine squares of a tic-tac-toe board (a 3×3 matrix) in any combination (i.e., unlike ordinary tic-tac-toe, it is not necessary that X's and O's be placed alternately, so, for example, all the squares could wind up with X's). Squares may also be blank, i.e., not containing either an X or and O. How many different boards are there?

4.2.5: How many different strings of length n can be formed from the ten digits? A digit may appear any number of times in the string or not at all.

4.2.6: How many different strings of length n can be formed from the 26 lower-case letters? A letter may appear any number of times or not at all.

4.2.7: Convert the following into K's, M's, G's, T's, or P's, according to the rules of the box in Section 4.2: (a) 2^{13} (b) 2^{17} (c) 2^{24} (d) 2^{38} (e) 2^{45} (f) 2^{59}.

4.2.8*: Convert the following powers of 10 into approximate powers of 2: (a) 10^{12} (b) 10^{18} (c) 10^{99}.

4.3 Counting Permutations

In this section we shall address another fundamental counting problem: Given n distinct objects, in how many different ways can we order those objects in a line? Such an ordering is called a *permutation* of the objects. We shall let $\Pi(n)$ stand for the number of permutations of n objects.

As one example of where counting permutations is significant in computer science, suppose we are given n objects, $a_1, a_2, \ldots, a_n$, to sort. If we know nothing about the objects, it is possible that any order will be the correct sorted order, and thus the number of possible outcomes of the sort will be equal to $\Pi(n)$, the number of permutations of n objects. We shall soon see that this observation helps us argue that general-purpose sorting algorithms require time proportional to $n \log n$, and therefore that algorithms like merge sort, which we saw in Section 3.10 takes

K's and M's and Powers of 2

A useful trick for converting powers of 2 into decimal is to notice that 2^{10}, or 1024, is very close to one thousand. Thus 2^{30} is $(2^{10})^3$, or about 1000^3, that is, a billion. Then, $2^{32} = 4 \times 2^{30}$, or about four billion. In fact, computer scientists often accept the fiction that 2^{10} is *exactly* 1000 and speak of 2^{10} as "1K"; the K stands for "kilo." We convert 2^{15}, for example, into "32K," because

$$2^{15} = 2^5 \times 2^{10} = 32 \times \text{“1000”}$$

But 2^{20}, which is exactly 1,048,576, we call "1M," or "one million," rather than "1000K" or "1024K." For powers of 2 between 20 and 29, we factor out 2^{20}. Thus, 2^{26} is $2^6 \times 2^{20}$ or 64 "million." That is why 2^{26} bytes is referred to as 64 million bytes or 64 "megabytes."

Below is a table that gives the terms for various powers of 10 and their rough equivalents in powers of 2.

PREFIX	LETTER	VALUE
Kilo	K	10^3 or 2^{10}
Mega	M	10^6 or 2^{20}
Giga	G	10^9 or 2^{30}
Tera	T	10^{12} or 2^{40}
Peta	P	10^{15} or 2^{50}

This table suggests that for powers of 2 beyond 29 we factor out 2^{30}, 2^{40}, or 2 raised to whatever multiple-of-10 power we can. The remaining powers of 2 name the number of giga-, tera-, or peta- of whatever unit we are measuring. For example, 2^{43} bytes is 8 terabytes.

$O(n \log n)$ time, are to within a constant factor as fast as can be.

There are many other applications of the counting rule for permutations. For example, it figures heavily in more complex counting questions like combinations and probabilities, as we shall see in later sections.

✦ **Example 4.2.** To develop some intuition, let us enumerate the permutations of small numbers of objects. First, it should be clear that $\Pi(1) = 1$. That is, if there is only one object A, there is only one order: A.

Now suppose there are two objects, A and B. We may select one of the two objects to be first and then the remaining object is second. Thus there are two orders: AB and BA. Therefore, $\Pi(2) = 2 \times 1 = 2$.

Next, let there be three objects: A, B, and C. We may select any of the three to be first. Consider the case in which we select A to be first. Then the remaining two objects, B and C, can be arranged in either of the two orders for two objects to complete the permutation. We thus see that there are two orders that begin with A, namely ABC and ACB.

Similarly, if we start with B, there are two ways to complete the order, corresponding to the two ways in which we may order the remaining objects A and C. We thus have orders BAC and BCA. Finally, if we start with C first, we can order the remaining objects A and B in the two possible ways, giving us orders CAB and CBA. These six orders,

$$ABC,\ ACB,\ BAC,\ BCA,\ CAB,\ CBA$$

are all the possible orders of three elements. That is, $\Pi(3) = 3 \times 2 \times 1 = 6$.

Next, consider how many permutations there are for 4 objects: A, B, C, and D. If we pick A first, we may follow A by the objects B, C, and D in any of their 6 orders. Similarly, if we pick B first, we can order the remaining A, C, and D in any of their 6 ways. The general pattern should now be clear. We can pick any of the four elements first, and for each such selection, we can order the remaining three elements in any of the $\Pi(3) = 6$ possible ways. It is important to note that the number of permutations of the three objects does not depend on which three elements they are. We conclude that the number of permutations of 4 objects is 4 times the number of permutations of 3 objects. ❖

More generally,

$$\Pi(n+1) = (n+1)\Pi(n) \text{ for any } n \geq 1 \tag{4.1}$$

That is, to count the permutations of $n+1$ objects we may pick any of the $n+1$ objects to be first. We are then left with n remaining objects, and these can be permuted in $\Pi(n)$ ways, as suggested in Fig. 4.4. For our example where $n+1 = 4$, we have $\Pi(4) = 4 \times \Pi(3) = 4 \times 6 = 24$.

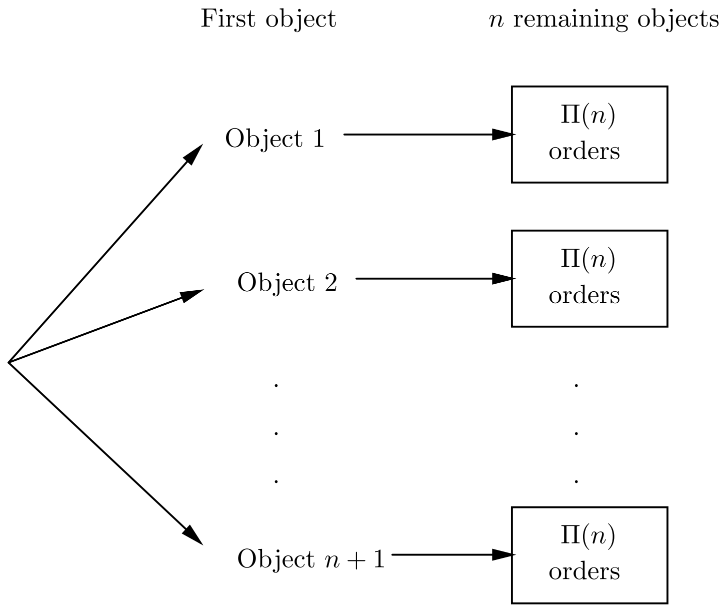


Fig. 4.4. The permutations of $n+1$ objects.

The Formula for Permutations

Equation (4.1) is the inductive step in the definition of the factorial function introduced in Section 2.5. Thus it should not be a surprise that $\Pi(n)$ equals $n!$. We can prove this equivalence by a simple induction.

STATEMENT $S(n)$: $\Pi(n) = n!$ for all $n \geq 1$.

BASIS. For $n = 1$, $S(1)$ says that there is 1 permutation of 1 object. We observed this simple point in Example 4.2.

INDUCTION. Suppose $\Pi(n) = n!$. Then $S(n+1)$, which we must prove, says that $\Pi(n+1) = (n+1)!$. We start with Equation (4.1), which says that

$$\Pi(n+1) = (n+1) \times \Pi(n)$$

By the inductive hypothesis, $\Pi(n) = n!$. Thus, $\Pi(n+1) = (n+1)n!$. Since

$$n! = n \times (n-1) \times \cdots \times 1$$

it must be that $(n+1) \times n! = (n+1) \times n \times (n-1) \times \cdots \times 1$. But the latter product is $(n+1)!$, which proves $S(n+1)$.

✦ **Example 4.3.** As a result of the formula $\Pi(n) = n!$, we conclude that the number of permutations of 4 objects is $4! = 4 \times 3 \times 2 \times 1 = 24$, as we saw above. As another example, the number of permutations of 7 objects is $7! = 5040$. ✦

How Long Does it Take to Sort?

One of the interesting uses of the formula for counting permutations is in a proof that sorting algorithms must take at least time proportional to $n \log n$ to sort n elements, unless they make use of some special properties of the elements. For example, as we note in the box on special-case sorting algorithms, we can do better than proportional to $n \log n$ if we write a sorting algorithm that works only for small integers.

However, if a sorting algorithm works on any kind of data, as long as it can be compared by some "less than" notion, then the only way the algorithm can decide on the proper order is to consider the outcome of a test for whether one of two elements is less than the other. A sorting algorithm is called a *general-purpose sorting algorithm* if its only operation upon the elements to be sorted is a comparison between two of them to determine their relative order. For instance, selection sort and merge sort of Chapter 2 each make their decisions that way. Even though we wrote them for integer data, we could have written them more generally by replacing comparisons like

General purpose sorting algorithm

```
if (A[j] < A[small])
```

on line (4) of Fig. 2.2 by a test that calls a Boolean-valued function such as

```
if (lessThan(A[j], A[small]))
```

Suppose we are given n distinct elements to sort. The answer — that is, the correct sorted order — can be any of the $n!$ permutations of these elements. If our algorithm for sorting arbitrary types of elements is to work correctly, it must be able to distinguish all $n!$ different possible answers.

Consider the first comparison of elements that the algorithm makes, say

```
lessThan(X,Y)
```

For each of the $n!$ possible sorted orders, either X is less than Y or it is not. Thus, the $n!$ possible orders are divided into two groups, those for which the answer to the first test is "yes" and those for which it is "no."

One of these groups must have at least $n!/2$ members. For if both groups have fewer than $n!/2$ members, then the total number of orders is less than $n!/2 + n!/2$, or less than $n!$ orders. But this upper limit on orders contradicts the fact that we started with exactly $n!$ orders.

Now consider the second test, on the assumption that the outcome of the comparison between X and Y was such that the larger of the two groups of possible orders remains (take either outcome if the groups are the same size). That is, at least $n!/2$ orders remain, among which the algorithm must distinguish. The second comparison likewise has two possible outcomes, and at least half the remaining orders will be consistent with one of these outcomes. Thus, we can find a group of at least $n!/4$ orders consistent with the first two tests.

We can repeat this argument until the algorithm has determined the correct sorted order. At each step, by focusing on the outcome with the larger population of consistent possible orders, we are left with at least half as many possible orders as at the previous step. Thus, we can find a sequence of tests and outcomes such that after the ith test, there are at least $n!/2^i$ orders consistent with all these outcomes.

Since we cannot finish sorting until every sequence of tests and outcomes is consistent with at most one sorted order, the number of tests t made before we finish must satisfy the equation

$$n!/2^t \leq 1 \qquad (4.2)$$

If we take logarithms base 2 of both sides of Equation (4.2) we have $\log_2 n! - t \leq 0$, or

$$t \geq \log_2(n!)$$

We shall see that $\log_2(n!)$ is about $n \log_2 n$. But first, let us consider an example of the process of splitting the possible orders.

✦ **Example 4.3.** Let us consider how the selection sort algorithm of Fig. 2.2 makes its decisions when given three elements (a, b, c) to sort. The first comparison is between a and b, as suggested at the top of Fig. 4.5, where we show in the box that all 6 possible orders are consistent before we make any tests. After the test, the orders `abc`, `acb`, and `cab` are consistent with the "yes" outcome (i.e., $a < b$), while the orders `bac`, `bca`, and `cba` are consistent with the opposite outcome, where $b > a$. We again show in a box the consistent orders in each case.

In the algorithm of Fig. 2.2, the index of the smaller becomes the value `small`. Thus, we next compare c with the smaller of a and b. Note that which test is made next depends on the outcome of previous tests.

After making the second decision, the smallest of the three is moved into the first position of the array, and a third comparison is made to determine which of the remaining elements is the larger. That comparison is the last comparison made by the algorithm when three elements are to be sorted. As we see at the bottom of Fig. 4.5, sometimes that decision is determined. For example, if we have already found $a < b$ and $c < a$, then c is the smallest and the last comparison of a and b must find a smaller.

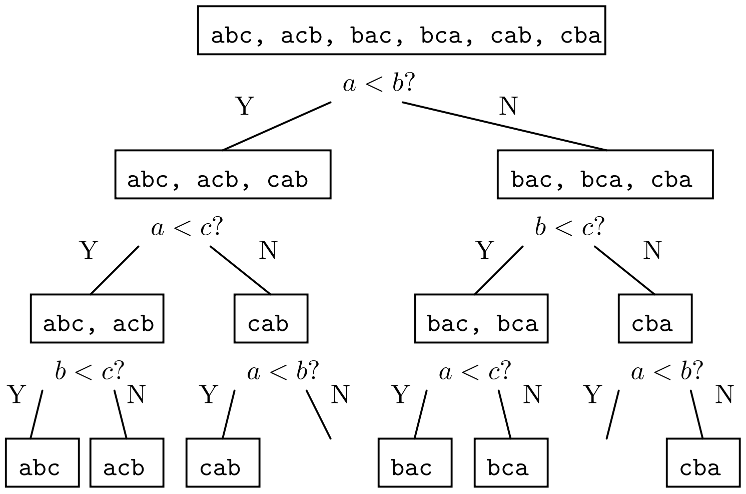


Fig. 4.5. Decision tree for selection sorting of 3 elements.

In this example, all paths involve 3 decisions, and at the end there is at most one consistent order, which is the correct sorted order. The two paths with no consistent order never occur. Equation (4.2) tells us that the number of tests t must be at least $\log_2 3!$, which is $\log_2 6$. Since 6 is between 2^2 and 2^3, we know that $\log_2 6$ will be between 2 and 3. Thus, at least some sequences of outcomes in any algorithm that sorts three elements must make 3 tests. Since selection sort makes only 3 tests for 3 elements, it is at least as good as any other sorting algorithm for 3 elements in the worst case. Of course, as the number of elements becomes large, we know that selection sort is not as good as can be done, since it is an $O(n^2)$ sorting algorithm and there are better algorithms such as merge sort. ✦

We must now estimate how large $\log_2 n!$ is. Since $n!$ is the product of all the integers from 1 to n, it is surely larger than the product of only the $\frac{n}{2}+1$ integers from $n/2$ through n. This product is in turn at least as large as $n/2$ multiplied by itself $n/2$ times, or $(n/2)^{n/2}$. Thus, $\log_2 n!$ is at least $\log_2\bigl((n/2)^{n/2}\bigr)$. But the latter is $\frac{n}{2}(\log_2 n - \log_2 2)$, which is

$$\frac{n}{2}(\log_2 n - 1)$$

For large n, this formula is approximately $(n \log_2 n)/2$.

A more careful analysis will tell us that the factor of 1/2 does not have to be there. That is, $\log_2 n!$ is very close to $n \log_2 n$ rather than to half that expression.

A Linear-Time Special-Purpose Sorting Algorithm

If we restrict the inputs on which a sorting algorithm will work, it can in one step divide the possible orders into more than 2 parts and thus work in less than time proportional to $n \log n$. Here is a simple example that works if the input is n distinct integers, each chosen in the range 0 to $2n - 1$.

```
(1)  for (i = 0; i < 2*n; i++)
(2)      count[i] = 0;
(3)  for (i = 0; i < n; i++)
(4)      count[a[i]]++;
(5)  for (i = 0; i < 2*n; i++)
(6)      if (count[i] > 0)
(7)          printf("%d\n", i);
```

We assume the input is in an array `a` of length n. In lines (1) and (2) we initialize an array `count` of length $2n$ to 0. Then in lines (3) and (4) we add 1 to the count for x if x is the value of `a[i]`, the ith input element. Finally, in the last three lines we print each of the integers i such that `count[i]` is positive. Thus we print those elements appearing one or more times in the input and, on the assumption the inputs are distinct, it prints all the input elements, sorted smallest first.

We can analyze the running time of this algorithm easily. Lines (1) and (2) are a loop that iterates $2n$ times and has a body taking $O(1)$ time. Thus, it takes $O(n)$ time. The same applies to the loop of lines (3) and (4), but it iterates n times rather than $2n$ times; it too takes $O(n)$ time. Finally, the body of the loop of lines (5) through (7) takes $O(1)$ time and it is iterated $2n$ times. Thus, all three loops take $O(n)$ time, and the entire sorting algorithm likewise takes $O(n)$ time. Note that if given an input for which the algorithm is not tailored, such as integers in a range larger than 0 through $2n - 1$, the program above fails to sort correctly.

We have shown only that any general-purpose sorting algorithm must have some input for which it makes about $n \log_2 n$ comparisons or more. Thus any general-purpose sorting algorithm must take at least time proportional to $n \log n$ in the worst case. In fact, it can be shown that the same applies to the "average" input. That is, the average over all inputs of the time taken by a general-purpose sorting algorithm must be at least proportional to $n \log n$. Thus, merge sort is about as good as we can do, since it has this big-oh running time for all inputs.

EXERCISES

4.3.1: Suppose we have selected 9 players for a baseball team.

a) How many possible batting orders are there?
b) If the pitcher has to bat last, how many possible batting orders are there?

4.3.2: How many comparisons does the selection sort algorithm of Fig. 2.2 make if there are 4 elements? Is this number the best possible? Show the top 3 levels of the decision tree in the style of Fig. 4.5.

4.3.3: How many comparisons does the merge sort algorithm of Section 2.8 make if there are 4 elements? Is this number the best possible? Show the top 3 levels of the decision tree in the style of Fig. 4.5.

4.3.4*: Are there more assignments of n values to n items or permutations of $n+1$ items? *Note*: The answer may not be the same for all n.

4.3.5*: Are there more assignments of $n/2$ values to n items than there are permutations of n items?

4.3.6**: Show how to sort n integers in the range 0 to $n^2 - 1$ in $O(n)$ time.

✦✦✦ 4.4 Ordered Selections

Sometimes we wish to select only some of the items in a set and give them an order. Let us generalize the function $\Pi(n)$ that counted permutations in the previous section to a two-argument function $\Pi(n, m)$, which we define to be the number of ways we can select m items from n in such a way that order matters for the selected items, but there is no order for the unselected items. Thus, $\Pi(n) = \Pi(n, n)$.

✦ **Example 4.5.** A horse race awards prizes to the first three finishers; the first horse is said to "win," the second to "place," and the third to "show." Suppose there are 10 horses in a race. How many different awards for win, place, and show are there?

Clearly, any of the 10 horses can be the winner. Given which horse is the winner, any of the 9 remaining horses can place. Thus, there are $10 \times 9 = 90$ choices for horses in first and second positions. For any of these 90 selections of win and place, there are 8 remaining horses. Any of these can finish third. Thus, there are $90 \times 8 = 720$ selections of win, place, and show. Figure 4.6 suggests all these possible selections, concentrating on the case where 3 is selected first and 1 is selected second. ✦

The General Rule for Selections Without Replacement

Let us now deduce the formula for $\Pi(n, m)$. Following Example 4.5, we know that there are n choices for the first selection. Whatever selection is first made, there will be $n-1$ remaining items to choose from. Thus, the second choice can be made in $n-1$ different ways, and the first two choices occur in $n(n-1)$ ways. Similarly, for the third choice we are left with $n-2$ unselected items, so the third choice can be made in $n-2$ different ways. Hence the first three choices can occur in $n(n-1)(n-2)$ distinct ways.

We proceed in this way until m choices have been made. Each choice is made from one fewer item than the choice before. The conclusion is that we may select m items from n without replacement but with order significant in

$$\Pi(n, m) = n(n-1)(n-2)\cdots(n-m+1) \tag{4.3}$$

different ways. That is, expression (4.3) is the product of the m integers starting and n and counting down.

Another way to write (4.3) is as $n!/(n-m)!$. That is,

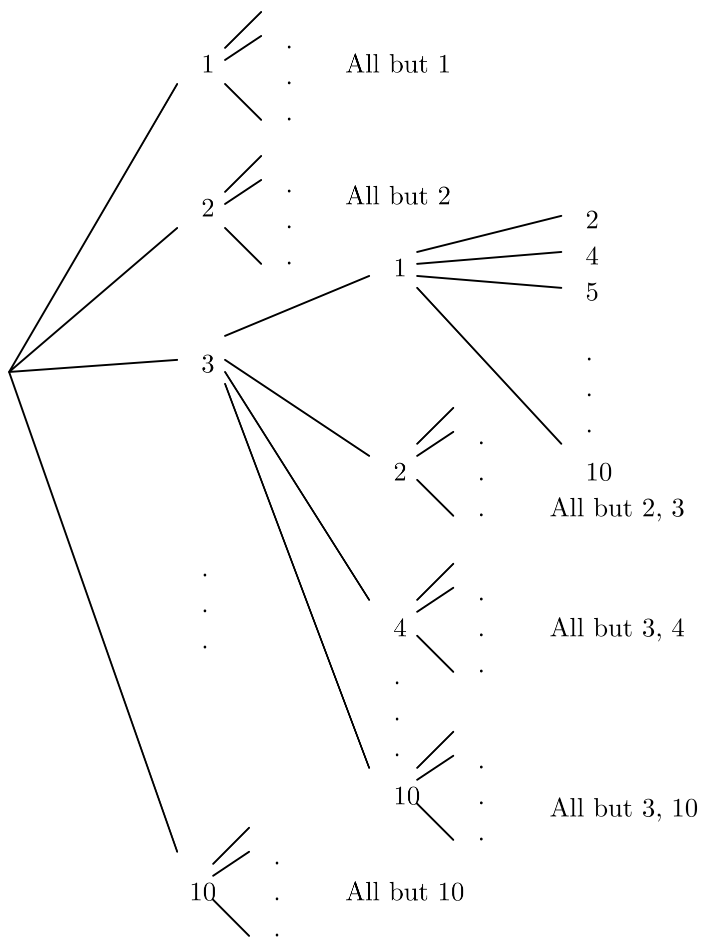


Fig. 4.6. Ordered selection of three things out of 10.

$$\frac{n!}{(n-m)!} = \frac{n(n-1)\cdots(n-m+1)(n-m)(n-m-1)\cdots(1)}{(n-m)(n-m-1)\cdots(1)}$$

The denominator is the product of the integers from 1 to $n-m$. The numerator is the product of the integers from 1 to n. Since the last $n-m$ factors in the numerator and denominator above are the same, $(n-m)(n-m-1)\cdots(1)$, they cancel and the result is that

$$\frac{n!}{(n-m)!} = n(n-1)\cdots(n-m+1)$$

This formula is the same as that in (4.3), which shows that $\Pi(n,m) = n!/(n-m)!$.

✦ **Example 4.6.** Consider the case from Example 4.5, where $n = 10$ and $m = 3$. We observed that $\Pi(10,3) = 10 \times 9 \times 8 = 720$. The formula (4.3) says that $\Pi(10,3) = 10!/7!$, or

Selections With and Without Replacement

The problem considered in Example 4.5 differs only slightly from the assignment problem considered in Section 4.2. In terms of houses and colors, we could almost see the selection of the first three finishing horses as an assignment of one of ten horses (the "colors") to each of three finishing positions (the "houses"). The only difference is that, while we are free to paint several houses the same color, it makes no sense to say that one horse finished both first and third, for example. Thus, while the number of ways to color three houses in any of ten colors is 10^3 or $10 \times 10 \times 10$, the number of ways to select the first three finishers out of 10 is $10 \times 9 \times 8$.

Selection with replacement

We sometimes refer to the kind of selection we did in Section 4.2 as *selection with replacement*. That is, when we select a color, say red, for a house, we "replace" red into the pool of possible colors. We are then free to select red again for one or more additional houses.

Selection without replacement

On the other hand, the sort of selection we discussed in Example 4.5 is called *selection without replacement*. Here, if the horse Sea Biscuit is selected to be the winner, then Sea Biscuit is not replaced in the pool of horses that can place or show. Similarly, if Secretariat is selected for second place, he is not eligible to be the third-place horse also.

$$\frac{10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1}{7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1}$$

The factors from 1 through 7 appear in both numerator and denominator and thus cancel. The result is the product of the integers from 8 through 10, or $10 \times 9 \times 8$, as we saw in Example 4.5. ✥

EXERCISES

4.4.1: How many ways are there to form a sequence of m letters out of the 26 letters, if no letter is allowed to appear more than once, for (a) $m = 3$ (b) $m = 5$.

4.4.2: In a class of 200 students, we wish to elect a President, Vice President, Secretary, and Treasurer. In how many ways can these four officers be selected?

4.4.3: Compute the following quotients of factorials: (a) 100!/97! (b) 200!/195!.

Mastermind

4.4.4: The game of Mastermind requires players to select a "code" consisting of a sequence of four pegs, each of which may be of any of six colors: red, green, blue, yellow, white, and black.

a) How may different codes are there?

b*) How may different codes are there that have two or more pegs of the same color? *Hint*: This quantity is the difference between the answer to (a) and another easily computed quantity.

c) How many codes are there that have no red peg?

d*) How many codes are there that have no red peg but have at least two pegs of the same color?

Quotients of Factorials

Note that in general, $a!/b!$ is the product of the integers between $b + 1$ and a, as long as $b < a$. It is much easier to calculate the quotient of factorials as

$$a \times (a - 1) \times \cdots \times (b + 1)$$

than to compute each factorial and divide, especially if b is not much less than a.

4.4.5*: Prove by induction on n that for any m between 1 and n, $\Pi(n, m) = n!/(n - m)!$.

4.4.6*: Prove by induction on $a - b$ that $a!/b! = a(a - 1)(a - 2) \cdots (b + 1)$.

4.5 Unordered Selections

There are many situations in which we wish to count the ways to select a set of items, but the order in which the selections are made does not matter. In terms of the horse race example of the previous section, we may wish to know which horses were the first three to finish, but we do not care about the order in which these three finished. Put another way, we wish to know how many ways we can select three horses out of n to be the top three finishers.

Example 4.7. Let us again assume $n = 10$. We know from Example 4.5 that there are 720 ways to select three horses, say A, B, and C, to be the win, place, and show horses, respectively. However, now we do not care about the order of finish of these three horses, only that A, B, and C were the first three finishers in some order. Thus, we shall get the answer "A, B, and C are the three best horses" in six different ways, corresponding to the ways that these three horses can be ordered among the top three. We know there are exactly six ways, because the number of ways to order 3 items is $\Pi(3) = 3! = 6$. However, if there is any doubt, the six ways are seen in Fig. 4.7.

Win	Place	Show
A	B	C
A	C	B
B	A	C
B	C	A
C	A	B
C	B	A

Fig. 4.7. Six orders in which a set of three horses may finish.

What is true for the set of horses A, B, and C is true of any set of three horses. Each set of three horses will appear exactly 6 times, in all of their possible orders, when we count the ordered selections of three horses out of 10. Thus, if we wish to count only the sets of three horses that may be the three top finishers, we must divide $\Pi(10, 3)$ by 6. Thus, there are $720/6 = 120$ different sets of three horses out of 10. ✦

✦ **Example 4.8.** Let us count the number of poker hands. In poker, each player is dealt five cards from a 52-card deck. We do not care in what order the five cards are dealt, just what five cards we have. To count the number of sets of five cards we may be dealt, we could start by calculating $\Pi(52, 5)$, which is the number of ordered selections of five objects out of 52. This number is $52!/(52-5)!$, which is $52!/47!$, or $52 \times 51 \times 50 \times 49 \times 48 = 311{,}875{,}200$.

However, just as the three fastest horses in Example 4.7 appear in $3! = 6$ different orders, any set of five cards to appear in $\Pi(5) = 5! = 120$ different orders. Thus, to count the number of poker hands without regard to order of selection, we must take the number of ordered selections and divide by 120. The result is $311{,}875{,}200/120 = 2{,}598{,}960$ different hands. ✦

Counting Combinations

Combinations of m things out of n

Let us now generalize Examples 4.7 and 4.8 to get a formula for the number of ways to select m items out of n without regard to order of selection. This function is usually written $\binom{n}{m}$ and spoken "n choose m" or "combinations of m things out of n." To compute $\binom{n}{m}$, we start with $\Pi(n, m) = n!/(n-m)!$, the number of ordered selections of m things out of n. We then group these ordered selections according to the set of m items selected. Since these m items can be ordered in $\Pi(m) = m!$ different ways, the groups will each have $m!$ members. We must divide the number of ordered selections by $m!$ to get the number of unordered selections. That is,

$$\binom{n}{m} = \frac{\Pi(n, m)}{\Pi(m)} = \frac{n!}{(n-m)! \times m!} \qquad (4.4)$$

✦ **Example 4.9.** Let us repeat Example 4.8, using formula (4.4) with $n = 52$ and $m = 5$. We have $\binom{52}{5} = 52!/(47! \times 5!)$. If we cancel the 47! with the last 47 factors of 52! and expand 5!, we can write

$$\binom{52}{5} = \frac{52 \times 51 \times 50 \times 49 \times 48}{5 \times 4 \times 3 \times 2 \times 1}$$

Simplifying, we get $\binom{52}{5} = 26 \times 17 \times 10 \times 49 \times 12 = 2{,}598{,}960$. ✦

A Recursive Definition of n Choose m

If we think recursively about the number of ways to select m items out of n, we can develop a recursive algorithm to compute $\binom{n}{m}$.

BASIS. $\binom{n}{0} = 1$ for any $n \geq 1$. That is, there is only one way to pick zero things out of n: pick nothing. Also, $\binom{n}{n} = 1$; that is, the only way to pick n things out of n is to pick them all.

INDUCTION. If $0 < m < n$, then $\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$. That is, if we wish to pick m things out of n, we can either

i) Not pick the first element, and then pick m things from the remaining $n-1$ elements. The term $\binom{n-1}{m}$ counts this number of possibilities.

or

ii) Pick the first element and then select $m-1$ things from among the remaining $n-1$ elements. The term $\binom{n-1}{m-1}$ counts these possibilities.

Incidently, while the idea of the induction should be clear — we proceed from the simplest cases of picking all or none to more complicated cases where we pick some but not all — we have to be careful to state what quantity the induction is "on." One way to look at this induction is that it is a complete induction on the product of n and the minimum of m and $n-m$. Then the basis case occurs when this product is 0 and the induction is for larger values of the product. We have to check for the induction that $n \times \min(m, n-m)$ is always greater than $(n-1) \times \min(m, n-m-1)$ and $(n-1) \times \min(m-1, n-m)$ when $0 < m < n$. This check is left as an exercise.

Pascal's triangle

This recursion is often displayed by *Pascal's triangle*, illustrated in Fig. 4.8, where the borders are all 1's (for the basis) and each interior entry is the sum of the two numbers above it to the northeast and northwest (for the induction). Then $\binom{n}{m}$ can be read from the $(m+1)$st entry of the $(n+1)$st row.

```
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
```

Fig. 4.8. The first rows of Pascal's triangle.

✦ **Example 4.10.** Consider the case where $n = 4$ and $m = 2$. We find the value of $\binom{4}{2}$ in the 3rd entry of the 5th row of Fig. 4.8. This entry is 6, and it is easy to check that $\binom{4}{2} = 4!/(2! \times 2!) = 24/(2 \times 2) = 6$. ✦

The two ways we have to compute $\binom{n}{m}$ — by formula (4.4) or by the above recursion — each compute the same value, naturally. We can argue so by appeal to physical reasoning. Both methods compute the number of ways to select m items out of n in an unordered fashion, so they must produce the same value. However, we can also prove the equality of the two approaches by an induction on n. We leave this proof as an exercise.

Running Time of Algorithms to Compute $\binom{n}{m}$

As we saw in Example 4.9, when we use formula (4.4) to compute $\binom{n}{m}$ we can cancel $(n-m)!$ in the denominator against the last $n-m$ factors in $n!$ to express $\binom{n}{m}$ as

$$\binom{n}{m} = \frac{n \times (n-1) \times \cdots \times (n-m+1)}{m \times (m-1) \times \cdots \times 1} \tag{4.5}$$

If m is small compared to n, we can evaluate the above formula much faster than we can evaluate (4.4). In principle, the fragment of C code in Fig. 4.9 does the job.

```
(1)      c = 1;
(2)      for (i = n; i > n-m; i--)
(3)          c *= i;
(4)      for (i = 2; i <= m; i++)
(5)          c /= i;
```

Fig. 4.9. Code to compute $\binom{n}{m}$.

Line (1) initializes c to 1; c will become the result, $\binom{n}{m}$. Lines (2) and (3) multiply c by each of the integers between $n-m+1$ and n. Then, lines (4) and (5) divide c by each of the integers between 2 and m. Thus, Fig. 4.9 implements the formula of Equation (4.5).

For the running time of Fig. 4.9, we have only to observe that the two loops, lines (2) – (3) and lines (4) – (5), each iterate m times and have a body that takes $O(1)$ time. Thus, the running time is $O(m)$.

In the case that m is close to n but $n-m$ is small, we can interchange the role of m and $n-m$. That is, we can cancel factors of $n!$ and $m!$, getting $n(n-1)\cdots(m+1)$ and divide that by $(n-m)!$. This approach gives us an alternative to (4.5), which is

$$\binom{n}{m} = \frac{n \times (n-1) \times \cdots \times (m+1)}{(n-m) \times (n-m-1) \times \cdots \times 1} \tag{4.6}$$

Likewise, there is a code fragment similar to Fig. 4.9 that implements formula (4.6) and takes time $O(n-m)$. Since both $n-m$ and m must be n or less for $\binom{n}{m}$ to be defined, we know that either way, $O(n)$ is a bound on the running time. Moreover, when m is either close to 0 or close to n, then the running time of the better of the two approaches is much less than $O(n)$.

However, Fig. 4.9 is flawed in an important way. It starts by computing the product of a number of integers and then divides by an equal number of integers. Since ordinary computer arithmetic can only deal with integers of a limited size (often, about two billion is as large as an integer can get), we run the risk of computing an intermediate result after line (3) of Fig. 4.9 that overflows the limit on integer size. That may be the case even though the value of $\binom{n}{m}$ is small enough to be represented in the computer.

A more desirable approach is to alternate multiplications and divisions. Start by multiplying by n, then divide by m. Multiply by $n-1$; then divide by $m-1$, and so on. The problem with this approach is that we have no reason to believe the result will be an integer at each stage. For instance, in Example 4.9 we would begin by multiplying by 52 and dividing by 5. The result is already not an integer.

Formulas for $\binom{n}{m}$ Must Yield Integers

It may not be obvious why the quotients of many factors in Equations (4.4), (4.5), or (4.6) must always turn out to be an integer. The only simple argument is to appeal to physical reasoning. The formulas all compute the number of ways to choose m things out of n, and this number must be some integer.

It is much harder to argue this fact from properties of integers, without appealing to the physical meaning of the formulas. It can in fact be shown by a careful analysis of the number of factors of each prime in numerator and denominator. As a sample, look at the expression in Example 4.9. There is a 5 in the denominator, and there are 5 factors in the numerator. Since these factors are consecutive, we know one of them must be divisible by 5; it happens to be the middle factor, 50. Thus, the 5 in the denominator surely cancels.

Thus, we need to convert to floating-point numbers before doing any calculation. We leave this modification as an exercise.

Now, let us consider the recursive algorithm to compute $\binom{n}{m}$. We can implement it by the simple recursive function of Fig. 4.10.

```
          /* compute n choose m for 0 <= m <= n */
          int choose(int n, int m)
          {
              int n, m;

(1)           if (m < 0 || m > n) {/* error conditions */
(2)               printf("invalid input\n");
(3)               return 0;
              }
(4)           else if (m == 0 || m == n) /* basis case */
(5)               return 1;
              else /* induction */
(6)               return (choose(n-1, m-1) + choose(n-1, m));
          }
```

Fig. 4.10. Recursive function to compute $\binom{n}{m}$.

The function of Fig. 4.10 is not efficient; it creates an exponential explosion in the number of calls to `choose`. The reason is that when called with n as its first argument, it usually makes two recursive calls at line (6) with first argument $n-1$. Thus, we might expect the number of calls made to double when n increases by 1. Unfortunately, the exact number of recursive calls made is harder to count. The reason is that the basis case on lines (4) and (5) can apply not only when $n = 1$, but for higher n, provided m has the value 0 or n.

We can prove a simple, but slightly pessimistic upper bound as follows. Let $T(n)$ be the running time of Fig. 4.10 with first argument n. We can prove that $T(n)$ is $O(2^n)$ simply. Let a be the total running time of lines (1) through (5), plus that part of line (6) that is involved in the calls and return, but not the time of the recursive calls themselves. Then we can prove by induction on n:

STATEMENT $S(n)$: If `choose` is called with first argument n and some second argument m between 0 and n, then the running time $T(n)$ of the call is at most $a(2^n - 1)$.

BASIS. $n = 1$. Then it must be that either $m = 0$ or $m = 1 = n$. Thus, the basis case on lines (4) and (5) applies and we make no recursive calls. The time for lines (1) through (5) is included in a. Since $S(1)$ says that $T(1)$ is at most $a(2^1 - 1) = a$, we have proved the basis.

INDUCTION. Assume $S(n)$; that is, $T(n) \le a(2^n - 1)$. To prove $S(n+1)$, suppose we call `choose` with first argument $n+1$. Then Fig. 4.10 takes time a plus the time of the two recursive calls on line (6). By the inductive hypothesis, each call takes at most time $a(2^n - 1)$. Thus, the total time consumed is at most

$$a + 2a(2^n - 1) = a(1 + 2^{n+1} - 2) = a(2^{n+1} - 1)$$

This calculation proves $S(n+1)$ and proves the induction.

We have thus proved that $T(n) \le a(2^n - 1)$. Dropping the constant factor and the low-order terms, we see that $T(n)$ is $O(2^n)$.

Curiously, while in our analyses of Chapter 3 we easily proved a smooth and tight upper bound on running time, the $O(2^n)$ bound on $T(n)$ is smooth but not tight. The proper smooth, tight upper bound is slightly less: $O(2^n/\sqrt{n})$. A proof of this fact is quite difficult, but we leave as an exercise the easier fact that the running time of Fig. 4.10 is proportional to the value it returns: $\binom{n}{m}$. An important observation is that the recursive algorithm of Fig. 4.10 is much less efficient that the linear algorithm of Fig. 4.9. This example is one where recursion hurts considerably.

The Shape of the Function $\binom{n}{m}$

For a fixed value of n, the function of m that is $\binom{n}{m}$ has a number of interesting properties. For a large value of n, its form is the bell-shaped curve suggested in Fig. 4.11. We immediately notice that this function is symmetric around the midpoint $n/2$; this is easy to check using formula (4.4) that states $\binom{n}{m} = \binom{n}{n-m}$.

Bell curve

The maximum height at the center, that is, $\binom{n}{n/2}$, is approximately $2^n/\sqrt{\pi n/2}$. For example, if $n = 10$, this formula gives 258.37, while $\binom{10}{5} = 252$.

The "thick part" of the curve extends for approximately $\sqrt{n}$ on either side of the midpoint. For example, if $n = 10,000$, then for m between 4900 and 5100 the value of $\binom{10,000}{m}$ is close to the maximum. For m outside this range, the value of $\binom{10,000}{m}$ falls off very rapidly.

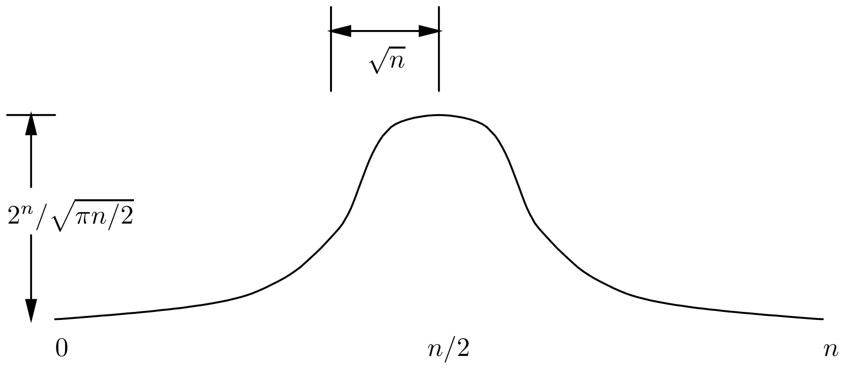


Fig. 4.11. The function $\binom{n}{m}$ for fixed n.

Binomial Coefficients

The function $\binom{n}{m}$, in addition to its use in counting, gives us the *binomial coefficients*. These numbers are found when we expand a two-term polynomial (a *binomial*) raised to a power, such as $(x + y)^n$.

Binomial

When we expand $(x+y)^n$, we get 2^n terms, each of which is $x^m y^{n-m}$ for some m between 0 and n. That is, from each of the factors $x + y$ we may choose either x or y to contribute as a factor in a particular term. The coefficient of $x^m y^{n-m}$ in the expansion is the number of terms that are composed of m choices of x and the remaining $n - m$ choices of y.

✦ **Example 4.11.** Consider the case $n = 4$, that is, the product

$$(x + y)(x + y)(x + y)(x + y)$$

There are 16 terms, of which only one is x^4y^0 (or just x^4). This term is the one we get if we select x from each of the four factors. On the other hand, there are four terms x^3y, corresponding to the fact that we can select y from any of the four factors and x from the remaining three factors. Symmetrically, we know that there is one term y^4 and four terms xy^3.

How many terms x^2y^2 are there? We get such a term if we select x from two of the four factors and y from the remaining two. Thus, we must count the number of ways to select two of the factors out of four. Since the order in which we select the two doesn't matter, this count is $\binom{4}{2} = 4!/(2! \times 2!) = 24/4 = 6$. Thus, there are six terms x^2y^2. The complete expansion is

$$(x + y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$$

Notice that the coefficients of the terms on the right side of the equality, $(1, 4, 6, 4, 1)$, are exactly a row of Pascal's triangle in Fig. 4.8. That is no coincidence, as we shall see. ✦

We can generalize the idea that we used to calculate the coefficient of x^2y^2 in Example 4.11. The coefficient of $x^m y^{n-m}$ in the expansion of $(x + y)^n$ is $\binom{n}{m}$. The reason is that we get a term $x^m y^{n-m}$ whenever we select m of the n factors to provide an x and the remainder of the factors to provide y. The number of ways to choose m factors out of n is $\binom{n}{m}$.

There is another interesting consequence of the relationship between binomial coefficients and the function $\binom{n}{m}$. We just observed that

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

Let $x = y = 1$. Then $(x+y)^n = 2^n$. All powers of x and y are 1, so the above equation becomes

$$2^n = \sum_{m=0}^{n} \binom{n}{m}$$

Put another way, the sum of all the binomial coefficients for a fixed n is 2^n. In particular, each coefficient $\binom{n}{m}$ is less than 2^n. The implication of Fig. 4.11 is that for m around $n/2$, $\binom{n}{m}$ is quite close to 2^n. Since the area under the curve of Fig. 4.11 is 2^n, we see why only a few values near the middle can be large.

EXERCISES

4.5.1: Compute the following values: (a) $\binom{7}{3}$ (b) $\binom{8}{3}$ (c) $\binom{10}{7}$ (d) $\binom{12}{11}$.

4.5.2: In how many ways can we choose a set of 5 different letters out of the 26 possible lower-case letters?

4.5.3: What is the coefficient of

a) x^3y^4 in the expansion $(x+y)^7$
b) x^5y^3 in the expansion of $(x+y)^8$

4.5.4*: At Real Security, Inc., computer passwords are required to have four digits (of 10 possible) and six letters (of 52 possible). Letters and digits may repeat. How many different possible passwords are there? *Hint*: Start by counting the number of ways to select the four positions holding digits.

4.5.5*: How many sequences of 5 letters are there in which exactly two are vowels?

4.5.6: Rewrite the fragment of Fig. 4.9 to take advantage of the case when $n - m$ is small compared with n.

4.5.7: Rewrite the fragment of Fig. 4.9 to convert to floating-point numbers and alternately multiply and divide.

4.5.8: Prove that if $0 \leq m \leq n$, then $\binom{n}{m} = \binom{n}{n-m}$

a) By appealing to the meaning of the function $\binom{n}{m}$
b) By using Equation 4.4

4.5.9*: Prove by induction on n that the recursive definition of $\binom{n}{m}$ correctly defines $\binom{n}{m}$ to be equal to $n!/\big((n-m)! \times m!\big)$.

4.5.10**: Show by induction on n that the running time of the recursive function `choose(n,m)` of Fig. 4.10 is at most $c\binom{n}{m}$ for some constant c.

4.5.11*: Show that $n \times \min(m, n-m)$ is always greater than

$$(n-1) \times \min(m, n-m-1)$$

and $(n-1) \times \min(m-1, n-m)$ when $0 < m < n$.

✦ 4.6 Orderings With Identical Items

In this section, we shall examine a class of selection problems in which some of the items are indistinguishable from one another, but the order of appearance of the items matters when items can be distinguished. The next section will address a similar class of problems where we do not care about order, and some items are indistinguishable.

Anagrams ✦ **Example 4.12.** *Anagram* puzzles give us a list of letters, which we are asked to rearrange to form a word. We can solve such problems mechanically if we have a dictionary of legal words and we can generate all the possible orderings of the letters. Chapter 10 considers efficient ways to check whether a given sequence of letters is in the dictionary. But now, considering combinatorial problems, we might start by asking how many different potential words we must check for presence in the dictionary.

For some anagrams, the count is easy. Suppose we are given the letters `abenst`. There are six letters, which may be ordered in $\Pi(6) = 6! = 720$ ways. One of these 720 ways is `absent`, the "solution" to the puzzle.

However, anagrams often contain duplicate letters. Consider the puzzle `eilltt`. There are not 720 different sequences of these letters. For example, interchanging the positions in which the two `t`'s appear does not make the word different.

Suppose we "tagged" the `t`'s and `l`'s so we could distinguish between them, say $\texttt{t}_1$, $\texttt{t}_2$, $\texttt{l}_1$, and $\texttt{l}_2$. Then we would have 720 orders of tagged letters. However, pair of orders that differ only in the position of the tagged `l`'s, such as $\texttt{l}_1\texttt{it}_2\texttt{t}_1\texttt{l}_2\texttt{e}$ and $\texttt{l}_2\texttt{it}_2\texttt{t}_1\texttt{l}_1\texttt{e}$, are not really different. Since all 720 orders group into pairs differing only in the subscript on the `l`'s, we can account for the fact that the `l`'s are really identical if we divide the number of strings of letters by 2. We conclude that the number of different anagrams in which the `t`'s are tagged but the `l`'s are not is 720/2=360.

Similarly, we may pair the strings with only `t`'s tagged if they differ only in the subscript of the `t`'s. For example, $\texttt{lit}_1\texttt{t}_2\texttt{le}$ and $\texttt{lit}_2\texttt{t}_1\texttt{le}$ are paired. Thus, if we divide by 2 again, we have the number of different anagram strings with the tags removed from both `t`'s and `l`'s. This number is $360/2 = 180$. We conclude that there are 180 different anagrams of `eilltt`. ✦

We may generalize the idea of Example 4.12 to a situation where there are n items, and these items are divided into k groups. Members of each group are indistinguishable, but members of different groups are distinguishable. We may let m_i be the number of items in the ith group, for $i = 1, 2, \ldots, k$.

✦ **Example 4.13.** Reconsider the anagram problem `eilltt` from Example 4.12. Here, there are six items, so $n = 6$. The number of groups k is 4, since there are

4 different letters. Two of the groups have one member (`e` and `i`), while the other two groups have two members. We may thus take $i_1 = i_2 = 1$ and $i_3 = i_4 = 2$. ✦

If we tag the items so members of a group are distinguishable, then there are $n!$ different orders. However, if there are i_1 members of the first group, these tagged items may appear in $i_1!$ different orders. Thus, when we remove the tags from the items in group 1, we cluster the orders into sets of size $i_1!$ that become identical. We must thus divide the number of orders by $i_1!$ to account for the removal of tags from group 1.

Similarly, removing the tags from each group in turn forces us to divide the number of distinguishable orders by $i_2!$, by $i_3!$, and so on. For those i_j's that are 1, this division is by $1! = 1$ and thus has no effect. However, we must divide by the factorial of the size of each group of more than one item. That is what happened in Example 4.12. There were two groups with more than one member, each of size 2, and we divided by 2! twice. We can state and prove the general rule by induction on k.

STATEMENT $S(k)$: If there are n items divided into k groups of sizes $i_1, i_2, \ldots, i_k$ respectively, items within a group are not distinguishable, but items in different groups are distinguishable, then the number of different distinguishable orders of the n items is

$$\frac{n!}{\prod_{j=1}^{k} i_j!} \tag{4.7}$$

BASIS. If $k = 1$, then there is one group of indistinguishable items, which gives us only one distinguishable order no matter how large n is. If $k = 1$ then i_1 must be n, and formula (4.7) reduces to $n!/n!$, or 1. Thus, $S(1)$ holds.

INDUCTION. Suppose $S(k)$ is true, and consider a situation with $k+1$ groups. Let the last group have $m = i_{k+1}$ members. These items will appear in m positions, and we can choose these positions in $\binom{n}{m}$ different ways. Once we have chosen the m positions, it does not matter which items in the last group we place in these positions, since they are indistinguishable.

Having chosen the positions for the last group, we have $n - m$ positions left to fill with the remaining k groups. The inductive hypothesis applies and tells us that each selection of positions for the last group can be coupled with $(n-m)!/\prod_{j=1}^{k} i_j!$ distinguishable orders in which to place the remaining groups in the remaining positions. This formula is just (4.7) with $n - m$ replacing n since there are only $n - m$ items remaining to be placed. The total number of ways to order the $k + 1$ groups is thus

$$\frac{\binom{n}{m}(n-m)!}{\prod_{j=1}^{k} i_j!} \tag{4.8}$$

Let us replace $\binom{n}{m}$ in (4.8) by its equivalent in factorials: $n!/\big((n-m)!m!\big)$. We then have

$$\frac{n!}{(n-m)!m!}\frac{(n-m)!}{\prod_{j=1}^{k} i_j!} \tag{4.9}$$

We may cancel $(n-m)!$ from numerator and denominator in (4.8). Also, remember that m is i_{k+1}, the number of members in the $(k+1)$st group. We thus discover that the number of orders is

$$\frac{n!}{\prod_{j=1}^{k+1} i_j!}$$

This formula is exactly what is given by $S(k+1)$.

✦ **Example 4.14.** An explorer has rations for two weeks, consisting of 4 cans of Tuna, 7 cans of Spam, and 3 cans of Beanie Weenies. If he opens one can each day, in how many orders can he consume the rations? Here, there are 14 items divided into groups of 4, 7, and 3 identical items. In terms of Equation (4.7), $n = 14$, $k = 3$, $i_1 = 4$, $i_2 = 7$, and $i_3 = 3$. The number of orders is thus

$$\frac{14!}{4!7!3!}$$

Let us begin by canceling the 7! in the denominator with the last 7 factors in 14! of the numerator. That gives us

$$\frac{14 \times 13 \times 12 \times 11 \times 10 \times 9 \times 8}{4 \times 3 \times 2 \times 1 \times 3 \times 2 \times 1}$$

Continuing to cancel factors in the numerator and denominator, we find the resulting product is 120,120. That is, there are over a hundred thousand ways in which to consume the rations. None sounds very appetizing. ✦

EXERCISES

4.6.1: Count the number of anagrams of the following words: (a) `error` (b) `street` (c) `allele` (d) `Mississippi`.

4.6.2: In how many ways can we arrange in a line

a) Three apples, four pears, and five bananas

b) Two apples, six pears, three bananas, and two plums

4.6.3*: In how many ways can we place a white king, a black king, two white knights, and a black rook on the chessboard?

4.6.4*: One hundred people participate in a lottery. One will win the grand prize of $1000, and five more will win consolation prizes of a $50 savings bond. How many different possible outcomes of the lottery are there?

4.6.5: Write a simple formula for the number of orders in which we may place $2n$ objects that occur in n pairs of two identical objects each.

4.7 Distribution of Objects to Bins

Our next class of counting problems involve the selection of a bin in which to place each of several objects. The objects may or may not be identical, but the bins are distinguishable. We must count the number of ways in which the bins can be filled.

❖ **Example 4.15.** Kathy, Peter, and Susan are three children. We have four apples to distribute among them, without cutting apples into parts. In how many different ways may the children receive apples?

There are sufficiently few ways that we can enumerate them. Kathy may receive anything from 0 to 4 apples, and whatever remains can be divided between Peter and Susan in only a few ways. If we let (i, j, k) represent the situation in which Kathy receives i apples, Peter receives j, and Susan receives k, the 15 possibilities are as shown in Fig. 4.12. Each row corresponds to the number of apples given to Kathy.

(0,0,4)	(0,1,3)	(0,2,2)	(0,3,1)	(0,4,0)
(1,0,3)	(1,1,2)	(1,2,1)	(1,3,0)	
(2,0,2)	(2,1,1)	(2,2,0)		
(3,0,1)	(3,1,0)			
(4,0,0)				

Fig. 4.12. Four apples can be distributed to three children in 15 ways.

There is a trick to counting the number of ways to distribute identical objects to bins. Suppose we have four letter `A`'s representing apples. Let us also use two `*`'s which will represent partitions between the apples belonging to different children. We order the `A`'s and `*`'s as we like and interpret all `A`'s before the first `*` as being apples belonging to Kathy. Those `A`'s between the two `*`'s belong to Peter, and the `A`'s after the second `*` are apples belonging to Susan. For instance, `AA*A*A` represents the distribution (2,1,1), where Kathy gets two apples and the other two children get one each. Sequence `AAA*A*` represents the situation (3,1,0), where Kathy gets three, Peter gets one, and Susan gets none.

Thus, each distribution of apples to bins is associated with a unique string of 4 `A`'s and 2 `*`'s. How many such strings are there? Think of the six positions forming such a string. Any four of those positions may be selected to hold the `A`'s; the other two will hold the `*`'s. The number of ways to select 4 items out of 6 is $\binom{6}{4}$, as we learned in Section 4.5. Since $\binom{6}{4} = 15$, we again conclude that there are 15 ways to distribute four apples to three children. ❖

The General Rule for Distribution to Bins

We can generalize the problem of Example 4.15 as follows. Suppose we are given n bins; these correspond to the three children in the example. Suppose also we are given m identical objects to place arbitrarily into the bins. How many distributions into bins are there.

We may again think of strings of `A`'s and `*`'s. The `A`'s now represent the objects, and the `*`'s represent boundaries between the bins. If there are n objects, we use n

A's, and if there are m bins, we need $m-1$ *'s to serve as boundaries between the portions for the various bins. Thus, strings are of length $n+m-1$.

We may choose any n of these positions to hold A's, and the rest will hold *'s. There are thus $\binom{n+m-1}{n}$ strings of A's and *'s, so there are this many distributions of objects to bins. In Example 4.15 we had $n = 4$ and $m = 3$, and we concluded that there were $\binom{n+m-1}{n} = \binom{6}{4}$ distributions.

Chuck-a-Luck ✦ **Example 4.16.** In the game of Chuck-a-Luck we throw three dice, each with six sides numbered 1 through 6. Players bet a dollar on a number. If the number does not come up, the dollar is lost. If the number comes up one or more times, the player wins as many dollars as there are occurrences of the number.

We would like to count the "outcomes," but there may initially be some question about what an "outcome" is. If we were to color the dice different colors so we could tell them apart, we could see this counting problem as that of Section 4.2, where we assign one of six numbers to each of three dice. We know there are $6^3 = 216$ ways to make this assignment.

However, the dice ordinarily aren't distinguishable, and the order in which the numbers come up doesn't matter; it is only the occurrences of each number that determines which players get paid and how much. For instance, we might observe that 1 comes up on two dice and 6 comes up on the third. The 6 might have appeared on the first, second, or third die, but it doesn't matter which.

Thus, we can see the problem as one of distribution of identical objects to bins. The "bins" are the numbers 1 through 6, and the "objects" are the three dice. A die is "distributed" to the bin corresponding to the number thrown on that die. Thus, there are $\binom{6+3-1}{3} = \binom{8}{3} = 56$ different outcomes in Chuck-a-Luck. ✦

Distributing Distinguishable Objects

We can extend the previous formula to allow for distribution into m bins of a collection of n objects that fall into k different classes. Objects within a class are indistinguishable from each other, but are distinguishable from those of other classes. Let us use symbol a_i to represent members of the ith class. We may thus form strings consisting of

1. For each class i, as many a_i's as there are members of that class.
2. $m-1$ *'s to represent the boundaries between the m bins.

The length of these strings is thus $n+m-1$. Note that the *'s form a $(k+1)$st class, with m members.

We learned how to count the number of such strings in Section 4.6. There are

$$\frac{(n+m-1)!}{(m-1)!\prod_{j=1}^{k} i_j!}$$

strings, where i_j is the number of members of the jth class.

✦ **Example 4.17.** Suppose we have three apples, two pears, and a banana to distribute to Kathy, Peter, and Susan. Then $m = 3$, the number of "bins," which is the number of children. There are $k = 3$ groups, with $i_1 = 3$, $i_2 = 2$, and $i_3 = 1$. Since there are 6 objects in all, $n = 6$, and the strings in question are of length

Comparison of Counting Problems

In this and the previous five sections we have considered six different counting problems. Each can be thought of as assigning objects to certain positions. For example, the assignment problem of Section 4.2 may be thought of as one where we are given n positions (corresponding to the houses) and and infinite supply of objects of k different types (the colors). We can classify these problems along three axes:

1. Do they place all the given objects?
2. Is the order in which objects are assigned important?
3. Are all the objects distinct, or are some indistinguishable?

Here is a table indicating the differences between the problems mentioned in each of these sections.

SECTION	TYPICAL PROBLEM	MUST USE ALL?	ORDER IMPORTANT?	IDENTICAL OBJECTS?
4.2	Painting houses	N	Y	N
4.3	Sorting	Y	Y	N
4.4	Horse race	N	Y	N
4.5	Poker hands	N	N	Y
4.6	Anagrams	Y	Y	Y
4.7	Apples to children	Y	N	Y

The problems of Sections 4.2 and 4.4 are not differentiated in the table above. The distinction is one of replacement, as discussed in the box on "Selections With and Without Replacement" in Section 4.4. That is, in Section 4.2 we had an infinite supply of each "color" and could select a color many times. In Section 4.4, a "horse" selected is not available for later selections.

$n+m-1 = 8$. The strings consist of three `A`'s standing for apples, two `P`'s standing for pears, one `B` standing for the banana, and two `*`'s, the boundaries between the shares of the children. The formula for the number of distributions is thus

$$\frac{(n+m-1)!}{(m-1)!i_1!i_2!i_3!} = \frac{8!}{2!3!2!1!} = 1680$$

ways in which these fruits may be distributed to Kathy, Peter, and Susan. ✦

EXERCISES

4.7.1: In how many ways can we distribute

a) Six apples to four children
b) Four apples to six children
c) Six apples and three pears to five children
d) Two apples, five pears, and six bananas to three children

4.7.2: How many outcomes are there if we throw

(a) Four indistinguishable dice

b) Five indistinguishable dice

4.7.3*: How many ways can we distribute seven apples to three children so that each child gets at least one apple?

4.7.4*: Suppose we start at the lower-left corner of a chessboard and move to the upper-right corner, making moves that are each either one square up or one square right. In how many ways can we make the journey?

4.7.5*: Generalize Exercise 4.7.4. If we have a rectangle of n squares by m squares, and we move only one square up or one square right, in how many ways can we move from lower-left to upper-right?

4.8 Combining Counting Rules

The subject of combinatorics offers myriad challenges, and few are as simple as those discussed so far in this chapter. However, the rules learned so far are valuable building blocks that may be combined in various ways to count more complex structures. In this section, we shall learn three useful "tricks" for counting:

1. Express a count as a sequence of choices.
2. Express a count as a difference of counts.
3. Express a count as a sum of counts for subcases.

Breaking a Count Into a Sequence of Choices

One useful approach to be taken, when faced with the problem of counting some class of arrangements is to describe the things to be counted in terms of a series of choices, each of which refines the description of a particular member of the class. In this section we present a series of examples intended to suggest some of the possibilities.

Example 4.18. Let us count the number of poker hands that are one-pair hands. A hand with one pair consists of two cards of one rank and three cards of ranks[1] that are different and also distinct from the rank of the pair. We can describe all one-pair hands by the following steps.

1. Select the rank of the pair.
2. Select the three ranks for the other three cards from the remaining 12 ranks.
3. Select the suits for the two cards of the pair.
4. Select the suits for each of the other three cards.

[1] The 13 ranks are Ace, King, Queen, Jack, and 10 through 2.

If we multiply all these numbers together, we shall have the number of one-pair hands. Note that the order in which the cards appear in the hand is not important, as we discussed in Example 4.8, and we have made no attempt to specify the order.

Now, let us take each of these factors in turn. We can select the rank of the pair in 13 different ways. Whichever rank we select for the pair, we have 12 ranks left. We must select 3 of these for the remaining cards of the hand. This is a selection in which order is unimportant, as discussed in Section 4.5. We may perform this selection in $\binom{12}{3} = 220$ ways.

Now, we must select the suits for the pair. There are four suits, and we must select two of them. Again we have an unordered selection, which we may do in $\binom{4}{2} = 6$ ways. Finally, we must select a suit for each of the three remaining cards. Each has 4 choices of suit, so we have an assignment like those of Section 4.2. We may make this assignment in $4^3 = 64$ ways.

The total number of one-pair hands is thus $13 \times 220 \times 6 \times 64 = 1,098,240$. This number is over 40% of the total number of 2,598,960 poker hands. ✦

Computing a Count as a Difference of Counts

Another useful technique is to express what we want to count as the difference between some more general class C of arrangements and those in C that do not meet the condition for the thing we want to count.

✦ **Example 4.19.** There are a number of other poker hands — two pairs, three of a kind, four of a kind, and full house — that can be counted in a manner similar to Example 4.18. However, there are other hands that require a different approach.

First, let us consider a straight-flush, which is five cards of consecutive rank (a straight) of the same suit (a flush). First, each straight begins with one of the ten ranks Ace through 10 as the lowest card. That is, the straights are Ace-2-3-4-5, 2-3-4-5-6, 3-4-5-6-7, and so on, up to 10-Jack-Queen-King-Ace. Once the ranks are determined, the straight-flush can be completely specified by giving the suit. Thus, we can count the straight-flushes by

1. Select the lowest rank in the straight (10 choices).
2. Select the suit (4 choices).

Thus, there are $10 \times 4 = 40$ straight-flushes.

Now, let us count the straights, that is, those hands whose ranks are consecutive but that are not straight-flushes. We shall first count all those hands with consecutive ranks, regardless of whether the suits are the same, and then subtract the 40 straight-flushes. To count hands with consecutive ranks, we can

1. Select the low rank (10 choices).
2. Assign a suit to each of the five ranks ($4^5 = 1024$ choices, as in Section 4.2).

The number of straights and straight-flushes is thus $10 \times 1024 =$ 10,240. When we subtract the straight-flushes, we are left with 10,240−40 = 10,200 hands that are classified as straights.

Next, let us count the number of flushes. Again, we shall first include the straight-flushes and then subtract 40. We can define a flush by

1. Select the suit (4 choices).
2. Select the five ranks out of thirteen ranks in any of $\binom{13}{5} = 1287$ ways, as in Section 4.5.

We conclude that the number of flushes is $4 \times 1287 - 40 = 5108$. ✦

Expressing a Count as a Sum of Subcases

Our third "trick" is to be methodical when faced with a problem that is too hard to solve directly. We break the problem of counting a class C into two or more separate problems, where each member of class C is covered by exactly one of the subproblems.

✦ **Example 4.20.** Suppose we toss a sequence of 10 coins. In how many sequences will 8 or more of the coins be heads? If we wanted to know how many sequences had exactly 8 heads, we could answer the problem by the method of Section 4.5. That is, there are 10 coins, and we wish to select 8 of them to be heads. We can do so in $\binom{10}{8} = 45$ ways.

To solve the problem of counting sequences of 8 or more heads, we break it into the three subproblems of counting sequences with exactly 8 heads, exactly 9 heads, and exactly 10 heads. We already did the first. The number of sequences with 9 heads is $\binom{10}{9} = 10$, and there is $\binom{10}{10} = 1$ sequence with all 10 heads. Thus, the number of sequences with 8 or more heads is $45 + 10 + 1 = 56$. ✦

✦ **Example 4.21.** Let us reconsider the problem of counting the outcomes in Chuck-a-Luck, which we solved in Example 4.16. Another approach is to divide the problem into three subproblems, depending on whether the number of different numbers showing is 3, 2, or 1.

a) We can count the number of outcomes with three different numbers using the technique of Section 4.5. That is, we must select 3 out of the 6 possible numbers on a die, and we can do so in $\binom{6}{3} = 20$ different ways.

b) Next, we must count the number of outcomes with two of one number and one of another. There are 6 choices for the number that appears twice, and no matter which number we choose to appear twice, there are 5 choices for the number that appears once. There are thus $6 \times 5 = 30$ outcomes with two of one number and one of another.

c) One number on all three dice can occur in 6 different ways, one for each of the numbers on a die.

Thus, the number of possible outcomes is $20 + 30 + 6 = 56$, as we also deduced in Example 4.16. ✦

EXERCISES

4.8.1*: Count the number of poker hands of the following types:

a) Two pairs
b) Three of a kind
c) Full house
d) Four of a kind

Be careful when counting one type of hand not to include hands that are better. For example, in (a), make sure that the two pairs are different (so you don't really have four of a kind) and the fifth card is different from the pairs (so you don't have a full house).

Blackjack

4.8.2*: A *blackjack* consists of two cards, one of which is an Ace and the other of which is a 10-point card, either a 10, Jack, Queen, or King.

a) How many different blackjacks are there in a 52-card deck?
b) In the game blackjack, one card is dealt down and the other is dealt up. Thus, in a sense order of the two cards matters. In this case, how many different blackjacks are there?

Pinochle deck

c) In a pinochle deck there are eight cards of each rank 9, 10, Jack, Queen, King, Ace (two indistinguishable cards of each suit) and no other cards. How many blackjacks are there, assuming order is unimportant?

4.8.3: How many poker hands are "nothing" (i.e., not one-pair or better)? You may use the results of Examples 4.18 and 4.19 as well as the answer to Exercise 4.8.1.

4.8.4: If we toss 12 coins in sequence, how many have

a) At least 9 heads
b) At most 4 heads
c) Between 5 and 7 heads
d) Fewer than 2 or more than 10 heads

4.8.5*: How many outcomes in Chuck-a-Luck have at least one 1?

4.8.6*: How many anagrams of the word `little` are there in which the two `t`'s are not adjacent?

Bridge

4.8.7**: A bridge hand consists of 13 of the 52 cards. We often classify hands by "distribution," that is, the way cards are grouped into suits. For example, a hand of 4-3-3-3 distribution has four of one suit and three of each of the other suits. A hand with 5-4-3-1 distribution has one suit of five cards, one of four, one of three, and one of one card. Count the number of hands with the following distributions: (a) 4-3-3-3 (b) 5-4-3-1 (c) 4-4-3-2 (d) 9-2-2-0.

4.9 Introduction to Probability Theory

Probability theory, along with its general importance, has many uses in computer science. One important application is the estimation of the running time of programs in the case of average or typical inputs. This evaluation is important for those algorithms whose worst-case running time is very much larger than the average running time. We shall see examples of such evaluations shortly.

Another use of probability is in designing algorithms for making decisions in the presence of uncertainty. For example, we can use probability theory to design algorithms for making the best possible medical diagnosis from available information or algorithms for allocating resources on the basis of expected future needs.

Probability Spaces

Experiment, outcome

When we speak of a *probability space*, we mean a finite set of points, each of which represents one possible *outcome* of an *experiment.* Each point x has associated with it a nonnegative real number called the *probability of* x, such that the sum of the probabilities of all the points is 1. One also speaks of probability spaces that have infinite numbers of points, although there is little application for these in computer science, and we shall not deal with them here.

Commonly, the points of a probability space have equal probability. Unless we state otherwise, you may assume that the probabilities of the various points in a probability space are equal. Thus, if there are n points in the probability space the probability of each point is $1/n$.

✦ **Example 4.22.** In Fig. 4.13 is a probability space with six points. The points are each identified with one of the numbers from 1 to 6, and we may think of this space as representing the outcomes of the "experiment" of throwing a single fair die. That is, one of the six numbers will appear on top of the die, and each number is equally likely to appear, that is, 1/6th of the time. ✦

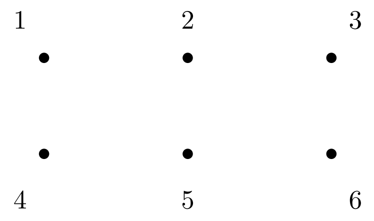


Fig. 4.13. A probability space with six points.

Event

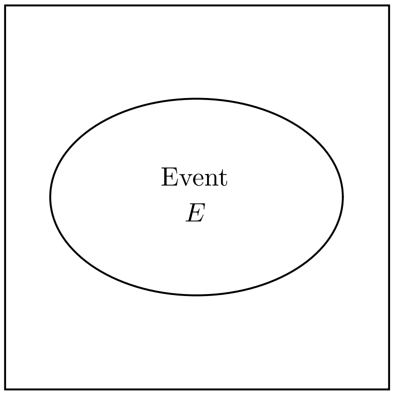
Any subset of the points in a probability space is called an *event.* The *probability* of an event E, denoted $\text{PROB}(E)$, is the sum of the probabilities of the points in E. If the points are all equally likely, then we can compute the probability of E by dividing the number of points in E by the number of points in the total probability space.

Probability Calculations

Often, the calculation of the probability of an event involves combinatorics. We must count the number of points in the event as well as the number of points in the entire probability space. When points are equally likely, the ratio of these two counts is the probability of the event. We shall give a series of examples to illustrate the calculation of probabilities in this fashion.

Infinite Probability Spaces

In certain circumstances, we can imagine that a probability space has an infinite number of points. The probability of any given point may be infinitesimal, and we can only associate finite probabilities with some collections of points. For a simple example, here is a probability space, the square, whose points are all the points of the plane within the square.



We may suppose that any point in the square is equally likely to be chosen. The "experiment" may be thought of as throwing a dart at the square in such a way that the dart is equally likely to wind up anywhere within the square, but not outside it. Although any point has only infinitesimal probability of being hit, the probability of a region of the square is the ratio of the area of the region to the area of the entire square. Thus, we can compute the probability of certain events.

For example, we show within the probability space an event E consisting of an ellipse contained within the square. Let us suppose the area of the ellipse is 29% of the area of the square. Then PROB(E) is 0.29. That is, if the dart is thrown at random at the square, 29% of the time the dart will land within the ellipse.

✦ **Example 4.23.** Figure 4.14 shows the probability space representing the throw of two dice. That is, the experiment is the tossing of two dice, in order, and observing the numbers on their upper faces. Assuming the dice are fair, there are 36 equally likely points, or outcomes of the experiment, so each point has probability 1/36. Each point corresponds to the assignment of one of six values to each die. For example, $(2, 3)$ represents the outcome where 2 appears on the first die and 3 on the second. Pair $(3, 2)$ represents 3 on the first die and 2 on the second.

The outlined region represents the event "craps," that is, a total 7 or 11 on the two dice. There are eight points in this event, six where the total is 7 and two where the total is 11. The probability of throwing craps is thus 8/36, or about 22%. ✦

✦ **Example 4.24.** Let us calculate the probability that a poker hand is a one-pair hand. We learned in Example 4.8 that there are 2,598,960 different poker hands. Consider the experiment of dealing a poker hand fairly, that is, with all hands equally likely. Thus, the probability space for this experiment has 2,598,960

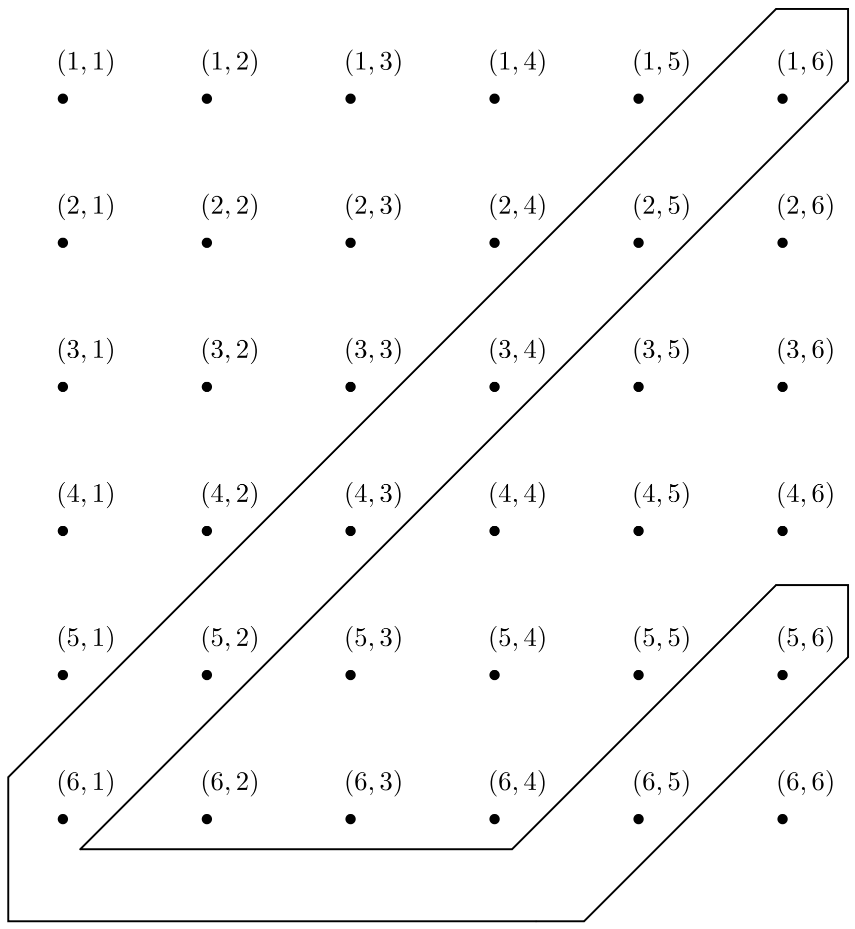


Fig. 4.14. The event "craps" in the probability space for the toss of two dice.

points. We also learned in Example 4.18 that 1,098,240 of these points represent hands classified as one pair. Assuming all hands are equally likely to be dealt, the probability of the event "one pair" is 1,098,240/2,598,960, or about 42%. ✦

✦ **Example 4.25.** In the game of Keno, twenty of the numbers from 1 to 80 are selected at random. Before the selection, players may guess some numbers; we shall concentrate on the "5-spot game," where players guess five numbers. A player who guesses correctly three, four, or five of the twenty selected numbers is rewarded; the amount of the payoff increases with the number of correct guesses. We shall calculate the probability that a player guesses exactly three numbers correctly in a 5-spot game. The probabilities of guessing four or five correctly are left as exercises.

Keno

To begin, the appropriate probability space has one point for each possible selection of twenty numbers from 1 to 80. The number of such selections is

$$\binom{80}{20} = \frac{80!}{20!60!}$$

When are Outcomes Random?

We have assumed in our examples that certain experiments have "random" outcomes; that is, all possible outcomes are equally likely. In some cases the justification for this assumptions comes from physics. For instance, when throwing fair (unweighted) dice, we assume that it is not physically possible to control the toss in such a way that one face is more likely than others to appear on top. That is a valid assumption in practice. Similarly, we assume that a fairly shuffled deck will not bias the outcome, and any card is equally likely to be found in any position in the deck.

In other cases, we find that what we imagine is random is actually not random at all, but is the result of a process that is predictable in principle but unpredictable in practice. For example, the numbers in a Keno game may be generated by a computer executing a particular algorithm, and yet without access to secret information used by the computer, it is impossible to predict the outcome.

Random number generator

All computer-generated "random" sequences of outcomes are the result of a special kind of algorithm called a *random number generator.* Designing one of these algorithms requires knowledge of some specialized mathematics and is beyond the scope of this book. However, we can offer an example of a random number generator that works fairly well in practice, called a *linear congruential generator.*

We specify constants $a \geq 2$, $b \geq 1$, $x_0 \geq 0$, and a modulus $m > \max(a, b, x_0)$. We can generate a sequence of numbers x_0, x_1, $x_2, \ldots$ using the formula

$$x_{n+1} = (ax_n + b) \bmod m$$

For suitable choices of a, b, m, and x_0, the resulting sequence of numbers will appear quite random, even though they are constructed from the "seed" x_0 by a specific algorithm.

Sequences generated by a random number generator have many uses. For instance, we can select numbers in a Keno game by taking numbers from the sequence described above, dividing each by 80, taking the remainder, and adding 1 to get a "random" number from 1 to 80. We keep doing so, throwing away repeated numbers, until twenty numbers are selected. The game will be perceived as fair, as long as no one knows the generation algorithm and the seed.

a number so huge that we are fortunate we do not have to write it down.

Now we must count the number of selections of twenty numbers out of eighty that include three of the five numbers chosen by the player and seventeen numbers from the seventy-five that the player has not chosen. We can choose three of the five in $\binom{5}{3} = 10$ ways, and we can choose seventeen from the remaining seventy-five in $\binom{75}{17} = \frac{75!}{17!58!}$ ways.

The ratio of the number of outcomes where the player picks three out of five to the total number of selections is thus

$$\frac{10\frac{75!}{17!58!}}{\frac{80!}{20!60!}}$$

If we multiply top and bottom by $\frac{20!60!}{80!}$, the above becomes

$$10(\frac{75!}{17!58!})(\frac{20!60!}{80!})$$

Now, we find in the numerator and denominator pairs of factorials that are close and can almost be canceled. For instance, 75! in the numerator and 80! in the denominator can be replaced by the product of the five numbers from 80 down to 76 in the denominator. The resulting simplification is

$$\frac{10 \times 60 \times 59 \times 20 \times 19 \times 18}{80 \times 79 \times 78 \times 77 \times 76}$$

Now we have a computation that involves manageable numbers. The result is about 0.084. That is, about 8.4% of the time the player guesses three out of five correctly. ✦

Fundamental Relationships

Let us close this section by observing several important properties of probabilities. First, if p is the probability of any event, then

$$0 \leq p \leq 1$$

That is, any event consists of 0 or more points, so its probability cannot be negative. Also, no event can consist of more points than are in the entire probability space, so its probability cannot exceed 1.

Complement event

Second, let E be an event in a probability space P. Then the *complement event* $\bar{E}$ for E is the set of points of P that are not in event E. We may observe that

$$\textsc{prob}(E) + \textsc{prob}(\bar{E}) = 1$$

or put another way, $\textsc{prob}(\bar{E}) = 1 - \textsc{prob}(E)$. The reason is that every point in P is either in E or in $\bar{E}$, but not both.

EXERCISES

4.9.1: Using the probability space for the toss of two fair dice shown in Fig. 4.14, give the probabilities of the following events:

a) A six is thrown (i.e., the sum of the dice is 6)
b) A ten is thrown
c) The sum of the dice is odd
d) The sum of the dice is between 5 and 9

4.9.2*: Calculate the probabilities of the following events. The probability space is the deal of two cards in order from an ordinary 52-card deck.

a) At least one card is an Ace
b) The cards are of the same rank
c) The cards are of the same suit
d) The cards are of the same rank and suit
e) The cards are the same either in rank or in suit
f) The first card is of a higher rank than the second card

4.9.3*: A dart is thrown at a one foot square on the wall, with equal likelihood of entering the square at any point. What is the probability that the dart is thrown

a) Within three inches of the center?
b) Within three inches of the border?

Note that for this exercise, the probability space is an infinite one-foot square, with all points within it equally likely.

4.9.4: Calculate the probability of the player in a 5-spot game of Keno guessing

a) Four out of five
b) All five

4.9.5: Write a C program to implement a linear congruential random number generator. Plot a histogram of the frequencies of the least significant digits of the first 100 numbers generated. What property should this histogram have?

4.10 Conditional Probability

In this section we shall develop a number of formulas and strategies for thinking about relationships among the probabilities of several events. One important development is a notion of independent experiments, where the outcome of one experiment does not affect the outcome of others. We shall also use our techniques to calculate probabilities in some complicated situations.

These developments depend upon a notion of "conditional probability." Informally, if we conduct an experiment and we find that an event E has occurred, it may or may not be the case that the point representing the outcome is also in some other event F. Figure 4.15 suggests this situation. The conditional probability of F given E is the probability that F has also occurred.

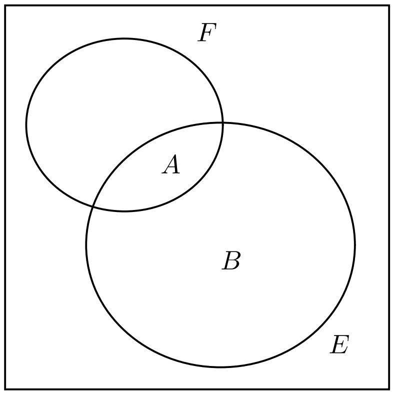


Fig. 4.15. The conditional probability of F given E is the probability that the outcome is in A divided by the probability that the outcome is in A or B.

Formally, if E and F are two events in a probability space, we say the *conditional probability of F given E*, denoted $\text{PROB}(F/E)$, is the sum of the probabilities of the points that are in both E and F divided by the sum of the probabilities of the points in E. In Fig. 4.15, region A is those points that are in both E and F, and B is those points that are in E but not F. If all points are equally likely, then

PROB(F/E) is the number of points in A divided by the sum of the numbers of points in A and B.

✦ **Example 4.26.** Let us consider the probability space of Fig. 4.14, which represents the toss of two dice. Let the event E be the six points in which the first die comes out 1, and let the event F be the six points in which the second die comes out 1. The situation is shown in Fig. 4.16. There is one point in both E and F, namely the point $(1, 1)$. There are five points in E that are not in F. Thus, the conditional probability PROB(F/E) is $1/6$. That is, the chance that the second die is 1, given that the first die is 1, is $1/6$.

We may notice that this conditional probability is exactly the same as the probability of F itself. That is, since F has 6 out of the 36 points in the space, PROB$(F) = 6/36 = 1/6$. Intuitively, the probability of throwing 1 on the second die is not affected by the fact that 1 has been thrown on the first die. We shall soon define the notion of "independent experiments," such as the throwing of dice in sequence, where the outcome of one experiment does not influence the outcome of the others. In these cases, if E and F are events representing outcomes of the two experiments, we expect that PROB$(F/E) =$ PROB(F). We have just seen an example of this phenomenon. ✦

✦ **Example 4.27.** Suppose our experiment is the deal of two cards, in order, from the usual 52-card deck. The number of points in this experiment of selection without replacement (as in Section 4.4) is $52 \times 51 = 2652$. We shall assume that the deal is fair, so each point has the same probability.

Let the event E be that the first card is an Ace, and let event F be that the second card is an Ace. Then the number of points in E is $4 \times 51 = 204$. That is, the first card must be one of the four Aces, and the second card can be any of 51 cards, excluding the Ace that was chosen first. Thus, PROB$(E) = 204/2652 = 1/13$. That result matches our intuition. All 13 ranks being equally likely, we expect that one time in 13 an Ace will appear first.

Similarly, the number of points in event F is $51 \times 4 = 204$. We can choose any of the 4 Aces for the second card and any of the remaining 51 cards for the first card. The fact that the first card is theoretically dealt first is irrelevant. There are thus 204 outcomes in which an Ace appears in the second position. Therefore, PROB$(F) = 1/13$ just as for E. Again, this result meets our intuition that one time in 13 an Ace will be dealt as the second card.

Now, let us compute PROB(F/E). Of the 204 points in E, there are 12 that have an Ace in the second position and therefore are also in F. That is, all points in E have an Ace in the first position. We may select this Ace in 4 different ways, corresponding to the 4 suits. For each selection, we have 3 different choices of Ace for the second position. Thus, the number of choices of two Aces with order considered is 4×3 according to the technique of Section 4.4.

Therefore, the conditional probability PROB(F/E) is $12/204$, or $1/17$. We notice that the conditional probability of F given E is not the same as the probability of F in this example. That also makes intuitive sense. The probability of getting an Ace in the second position goes down when we know there is an Ace in the first position. For then, there are only 3 Aces remaining out of 51 cards, and $3/51 = 1/17$.

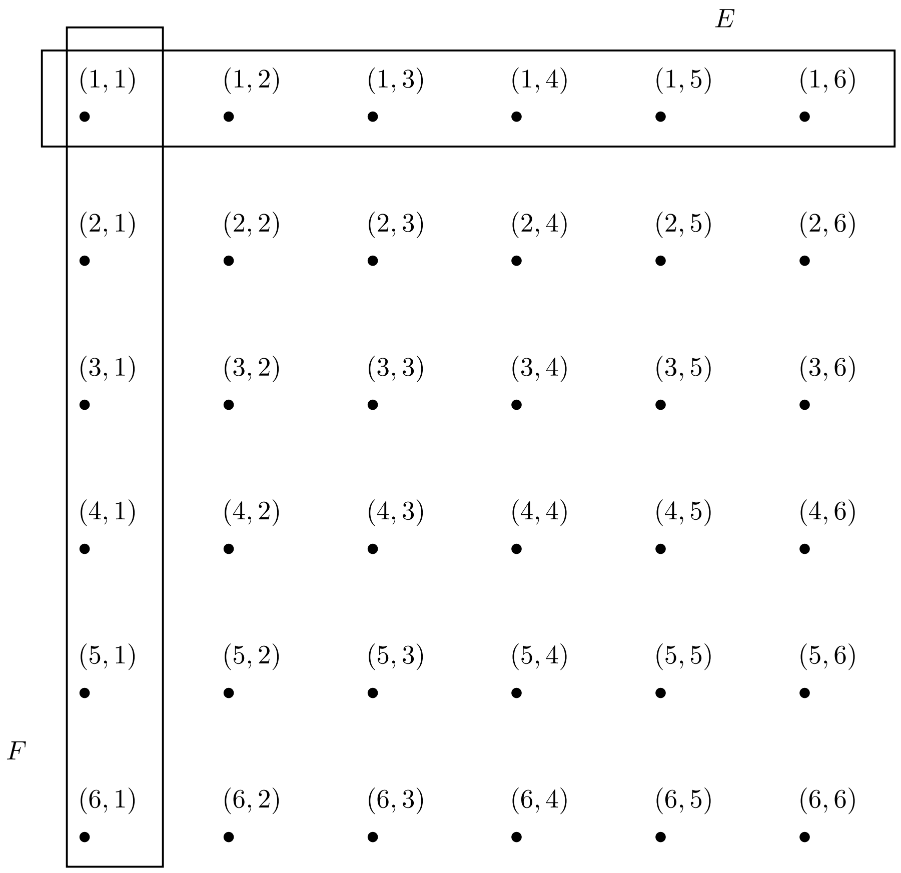


Fig. 4.16. Events representing 1 on the first or second dice.

In comparison, if we know nothing about the first card, there are 4 Aces out of 52 that we may receive as the second card. ❖

Independent Experiments

As we suggested in Examples 4.23, 4.26, and 4.27, we sometimes form a probability space representing the outcomes of two or more experiments. In the simplest cases, the points in this joint probability space are lists of outcomes, one for each of the experiments. Figure 4.16 is an example of a probability space that is joint between two experiments. In other situations, where there is a connection between the outcomes of the experiments, the joint space may have some points missing. Example 4.27 discussed such a case, where the joint space represented the deal of two cards and the pairs of outcomes in which the two cards are identical are not possible.

Joint probability space

There is an intuitive notion of the outcome of one experiment X being "independent" of previous experiments in the sequence, meaning that the probabilities of the various outcomes of X do not depend on the outcomes of the previous experiments. Thus, in Example 4.26 we argued that the roll of the second die is independent of the roll of the first die, while in Example 4.27 we saw that the second card dealt was not independent of the first, since the first card was then unavailable.

In defining independence, we shall focus on two experiments. However, since either experiment may itself be the sequence of several experiments, we effectively cover the case of many experiments. We must begin with a probability space that represents the outcome of two successive experiments, X_1 and X_2.

✦ **Example 4.28.** Figure 4.14 illustrates a joint probability space in which experiment X_1 is the throw of the first die and X_2 is the throw of the second die. Here, every pair of outcomes is represented by one point, and the points have equal probability, 1/36.

In Example 4.27 we discussed a space of 2652 points representing the selection of two cards in order. This space consists of all pairs (C, D) in which C and D are cards, and $C \neq D$. Again, each of these points has the same probability, 1/2652. ✦

In the probability space representing the outcomes of X_1 followed by X_2, there are events that represent the outcome of one of the experiments. That is, if a is a possible outcome of experiment X_1, then there is an event consisting of all those points in which the outcome of the first experiment is a. Let us call this event E_a. Similarly, if b is a possible outcome of X_2, then there is an event F_b consisting of all those points in which the outcome of the second experiment is b.

✦ **Example 4.29.** In Fig. 4.16, E is E_1, the event of all points for which the outcome of the first experiment is 1. Likewise, F is the event F_1, whose points are those where the outcome of the second experiment is 1. More generally, each row corresponds to one of the six possible outcomes of the first experiment, and each column corresponds to one of the six possible outcomes of the second experiment. ✦

Independence

Formally, we say that experiment X_2 is *independent of* experiment X_1 if for all outcomes a for X_1 and all outcomes b for X_2, $\text{PROB}(F_b/E_a) = \text{PROB}(F_b)$. That is, no matter what the outcome of experiment X_1, the conditional probability of each outcome of X_2 is the same as it is in the whole probability space.

✦ **Example 4.30.** Returning to the probability space of Fig. 4.16 representing the toss of two dice, let a and b each be any of the numbers from 1 to 6. Let E_a be the event that the first die is a and F_b be the event that the second die is b. We note that the probability of each of these events is 1/6; they are each one row or column. For any a and b, $\text{PROB}(F_b/E_a)$ is also 1/6. We argued this fact for the case $a = b = 1$ in Example 4.26, but the same argument applies to any two outcomes a and b, because their events have exactly one point in common. Thus, the tosses of the two dice are independent.

On the other hand, in the card-based Example 4.27, we do not get independence. Here, the experiment X_1 is the selection of the first card, and the experiment X_2 is the selection of the second card from the remaining deck. Consider an event like $F_{A\spadesuit}$ — that is, the second card is the Ace of Spades. It is easy to count that the probability of this event, PROB($F_{A♠}$), is 1/52.

Now, consider an event like $E_{3♣}$, the first card is the Three of Clubs. The number of points in both $E_{3♣}$ and $F_{A♠}$ is 1, namely the point (3♣, A♠). The total number of points in $E_{3♣}$ is 51, namely those points of the form (3♣, C) where C is any card but the Three of Clubs. Thus, the conditional probability PROB($F_{A♠}/E_{3♣}$) is 1/51, not 1/52 as it would have to be if the two experiments were independent.

For a more extreme example, consider the event $E_{A♠}$ in which the first card is the Ace of Spades. Since $E_{A♠}$ and $F_{A♠}$ have no point in common, PROB($F_{A♠}/E_{A♠}$) is 0 instead of 1/52. ✦

A Distributive Law for Probabilities

Region

Sometimes it is easier to calculate the probability of an event if we first divide the probability space into regions[2] that partition the space. That is, every point is in exactly one region. Typically, the probability space represents the outcome of a sequence of experiments, and the regions, which are themselves events, correspond to the possible outcomes of one of these experiments.

Suppose we wish to calculate the probability of event E in a space of n points that is divided into k regions, $R_1, R_2, \ldots, R_k$. For simplicity, we shall assume that all points have equal probability, although the conclusion we draw holds as well if the points have differing probabilities. Let event E consist of m points. Let region R_i have r_i points, for $i = 1, 2, \ldots, k$. Finally, let the number of points of E that lie in region R_i be e_i. Note that $\sum_{i=1}^{k} r_i = n$, and $\sum_{i=1}^{k} e_i = m$. The reason in each case is that points are each in one and only one region.

We know PROB(E) $= m/n$, because m/n is the fraction of points in E. If we replace m by the sum of the e_i's we get

$$\text{PROB}(E) = \sum_{i=1}^{k} \frac{e_i}{n}$$

Next, introduce factor r_i in both the numerator and denominator in each term of the sum above. The result is

$$\text{PROB}(E) = \sum_{i=1}^{k} (\frac{e_i}{r_i})(\frac{r_i}{n})$$

Now, notice that $r_i/n = \text{PROB}(R_i)$; that is, r_i/n is the fraction of the entire space in region R_i. Also, e_i/r_i is PROB(E/R_i), the conditional probability of event E given event R_i. Put another way, e_i/r_i is the fraction of the points in region R_i that are in E. The result is the following formula for the probability of an event E.

$$\text{PROB}(E) = \sum_{i=1}^{k} \text{PROB}(E/R_i)\text{PROB}(R_i) \tag{4.10}$$

Informally, the probability of E is the sum over all regions of the probability of being in that region times the probability of E within that region.

[2] A "region" is synonymous with an "event," that is, a subset of a probability space. However, we shall use the term region to emphasize the fact that the space is being partitioned into events that completely cover the space and do not overlap.

✦ **Example 4.31.** The diagram in Fig. 4.17 suggests how Equation (4.10) is to be applied. There we see a probability space that has been divided vertically into three regions, R_1, R_2, and R_3. There is an event E, which we doubly outline. We let a through f be the numbers of points in the six sets shown.

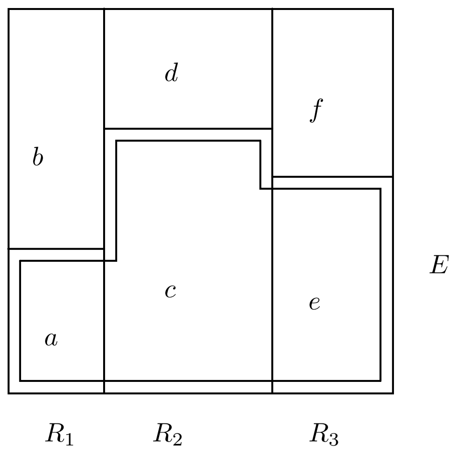


Fig. 4.17. A probability space divided into regions.

Let $n = a + b + c + d + e + f$. Then PROB$(R_1) = (a + b)/n$, PROB$(R_2) = (c + d)/n$, and PROB$(R_3) = (e + f)/n$. Next, the conditional probabilities of E in the three regions are PROB$(E/R_1) = a/(a + b)$, PROB$(E/R_2) = c/(c + d)$, and PROB$(E/R_3) = e/(e + f)$. Now, to evaluate formula (4.10), we have

$$\text{PROB}(E) = \text{PROB}(E/R_1)\text{PROB}(R_1) + \text{PROB}(E/R_2)\text{PROB}(R_2)+ \\ \text{PROB}(E/R_3)\text{PROB}(R_3)$$

This formula, in terms of the parameters a through f, is

$$\begin{aligned}\text{PROB}(E) &= \Big(\frac{a}{a+b}\Big)\Big(\frac{a+b}{n}\Big) \ + \ \Big(\frac{c}{c+d}\Big)\Big(\frac{c+d}{n}\Big) \ + \ \Big(\frac{e}{e+f}\Big)\Big(\frac{e+f}{n}\Big) \\ &= \frac{a}{n} + \frac{c}{n} + \frac{e}{n}\end{aligned}$$

Notice that the same result can be obtained by simply comparing the numbers of points in the three areas labeled a, c, and e with the size of the entire space. This fraction, $(a + c + e)/n$, is exactly what the formula above gives for the probability of E. That observation illustrates Equation (4.10). ✦

✦ **Example 4.32.** Let us use Equation (4.10) to compute the probability of the event E that two cards dealt in order are both Aces. The probability space is the 2652 points discussed in Example 4.27. We shall divide this space into two regions:

R_1: Those points in which the first card is an Ace. There are $4 \times 51 = 204$ such points, since we may pick the first card to be an Ace in 4 ways, and there are then 51 choices of second card.

R_2: The remaining 2448 points.

Equation (4.10) in this case becomes

$$\textsc{prob}(E) = \textsc{prob}(E/R_1)\textsc{prob}(R_1) + \textsc{prob}(E/R_2)\textsc{prob}(R_2)$$

Clearly $\textsc{prob}(E/R_2)$, the conditional probability of E given R_2, is 0. There is no way to get two Aces if the first card is not an Ace. We must thus compute $\textsc{prob}(E/R_1)\textsc{prob}(R_1)$, and this value is $\textsc{prob}(E)$. Now $\textsc{prob}(R_1) = 204/2652 = 1/13$. In other words, the chance of getting an Ace as the first card is 1/13. Since there are thirteen ranks, that probability makes sense.

Now, we need to compute $\textsc{prob}(E/R_1)$. If the first card is an Ace, then there remain 3 Aces out of 51 cards. Thus, $\textsc{prob}(E/R_1) = 3/51 = 1/17$. We conclude that $\textsc{prob}(E) = (1/17)(1/13) = 1/221$. ✦

✦ **Example 4.33.** Let us apply Equation (4.10) to the problem of computing the probability of the event E that at least one 1 appears in the toss of three dice, as in the game Chuck-a-Luck described in Example 4.16. First, we must understand that the notion of an "outcome" described in that example does not match the notion of a point in a probability space. In Example 4.16 we established that there were 56 different "outcomes," which were defined to be the numbers of occurrences of 1 through 6 on the faces of the dice. For instance, "a 4, a 5, and a 6" is one possible outcome; "two 3's and a 4" is another. However, not all outcomes in this sense have the same probability. In particular, outcomes with three different numbers showing have twice the probability of outcomes with two of one number and six times the probability of an outcome where all three dice show the same number.

While we could use the probability space whose points are "outcomes " in the sense of Example 4.16, it is more natural to consider the order in which the dice are rolled and thus develop a probability space whose points have equal probability. There are $6^3 = 216$ different outcomes corresponding to the roll of three dice, in order, and each of these outcomes has probability 1/216.

We could calculate the probability of at least one 1 in a direct manner, without using Equation (4.10). First, calculate the number of rolls in which no 1 appears. We can assign to each of the three dice any of the numbers from 2 to 6. There are thus $5^3 = 125$ points in the space that have no 1, and $216 - 125 = 91$ points that do have a 1. Therefore, $\textsc{prob}(E) = 91/216$, or about 42%.

The approach above is short but requires that we use several "tricks." Another way to calculate the probability "by force" is to divide the space into three regions, corresponding to the cases in which there are one, two, or three different numbers showing. Let R_i be the region of points with i different numbers. We can calculate the probabilities of the various regions as follows. For R_1 there are only six points, one for each of the numbers 1 through 6 that may appear on all three dice. For R_3 there are $6 \times 5 \times 4 = 120$ ways to select three different numbers out of six, according to the rule of Section 4.4. Thus, R_2 must have the remaining $216 - 6 - 120 = 90$ points.[3] The probabilities of the regions are $\textsc{prob}(R_3) = 6/216 = 1/36$, $\textsc{prob}(R_2) = 90/216 = 5/12$, and $\textsc{prob}(R_1) = 120/216 = 5/9$.

Next, we can calculate the conditional probabilities. If there are three numbers out of the possible six showing, the probability is 1/2 that one of them is 1. If two

[3] We can compute this number directly by multiplying 6 ways that we can choose the number to appear twice, times 5 ways we can pick the number on the remaining dice, times $\binom{3}{1} = 3$ ways we can pick the die that has the unique number. Note that $6 \times 5 \times 3 = 90$.

numbers are showing, the probability is 1/3 that 1 appears at least once. If only one number shows, there is a 1/6 chance that it is 1. Thus, $\text{PROB}(E/R_1) = 1/6$, $\text{PROB}(E/R_2) = 1/3$, and $\text{PROB}(E/R_3) = 1/2$. We can put all these probabilities together to evaluate (4.10). The result is

$$\begin{aligned}\text{PROB}(E) &= (1/6)(1/36) + (1/3)(5/12) + (1/2)(5/9)\\ &= 1/216 + 5/36 + 5/18 = 91/216\end{aligned}$$

This fraction agrees with the direct calculation, of course. If we see the “trick” of the direct calculation, that approach is considerably easier. However, breaking the problem into regions is frequently a more reliable way to guarantee success. ✦

A Product Rule for Independent Experiments

A common type of probability problem asks for the probability of a sequence of outcomes to a sequence of independent experiments. In that case, Equation (4.10) has an especially simple form, and it tells us that the probability of a sequence of outcomes is the product of the probabilities of each outcome by itself.

First, whenever we divide the probability space into k regions of equal size, we know that $\text{PROB}(R_i) = 1/k$ for all i. Thus (4.10) reduces to

$$\text{PROB}(E) = \sum_{i=1}^{k} \frac{1}{k}\text{PROB}(E/R_i) \tag{4.11}$$

A useful way to look at (4.11) is that the probability of E is the average over all regions of the probability of E given we are in that region.

Now, consider a probability space that represents the outcomes of two independent experiments X_1 and X_2. We may divide this space into k regions, each the set of points for which the outcome of X_1 has a particular value. Then each of the regions has the same probability, $1/k$.

Suppose we want to calculate the probability of the event E in which X_1 has the outcome a and X_2 has the outcome b. We may use formula (4.11). If R_i is not the region corresponding to outcome a for X_1, then $\text{PROB}(E/R_i) = 0$. Thus, all but the term for the region of a drops out of (4.11). If R_a is that region, we get

$$\text{PROB}(E) = \frac{1}{k}\text{PROB}(E/R_a) \tag{4.12}$$

What is $\text{PROB}(E/R_a)$? It is the probability that X_1 has outcome a and X_2 has outcome b, given that X_1 has outcome a. Since we are given that X_1 has outcome a, $\text{PROB}(E/R_a)$ is just the probability that X_2 has outcome b given that X_1 has outcome a. Since X_1 and X_2 are independent, $\text{PROB}(E/R_a)$ is just the probability that X_2 has outcome b. If there are m possible outcomes for X_2, then $\text{PROB}(E/R_a)$ is $1/m$. Then (4.12) becomes

$$\text{PROB}(E) = (\frac{1}{k})(\frac{1}{m})$$

We may generalize the above reasoning to any number of experiments. To do so, we let experiment X_1 be a sequence of experiments and show by induction on the total number of independent experiments that the probability of all having a particular sequence of outcomes is the product of the probabilities of each outcome.

Using Independence to Simplify Calculations

There are many opportunities to simplify probability calculations if we know that experiments are independent. The product rule is one such example. Another is that whenever event E is the set of points with a particular outcome from experiment X_1, and F is the set of points with a particular outcome of another, independent experiment X_2, then $\text{PROB}(E/F) = \text{PROB}(E)$.

In principle, telling whether two experiments are independent is a complex task involving examination of the probability space representing pairs of outcomes of each experiment. However, often we can appeal to the physics of the situation to conclude that experiments are independent without doing this calculation. For example, when we throw dice in sequence, there is no physical reason why the outcome of one would influence the other, so they must be independent experiments. Contrast the situation of dice with the deal of several cards from a deck. Since cards dealt are unavailable to be dealt at a future step, we do not expect successive cards to be independent of each other. We in fact observed this lack of independence in Example 4.29.

✦ **Example 4.34.** The probability that the last four digits of a phone number are 1234 is 0.0001. The selection of each digit is an experiment with ten possible outcomes: 0 through 9. Moreover, each selection is independent of the other selections, since we are performing a "selection with replacement" as in Section 4.2. The probability that the first digit is 1 is 1/10. Similarly, the probability that the second digit is 2 is 1/10, and likewise for the other two digits. The probability of the event that the four digits are 1234 in order is $(1/10)^4 = 0.0001$. ✦

EXERCISES

4.10.1: Using the space of Fig. 4.14, give the conditional probabilities of the following pairs of events.

a) The second die is even, given that the first die is odd
b) The first die is even, given that the second die is at least 3
c) The sum of the dice is at least 7, given that the first die is 4
d) The second die is 3, given that the sum of the dice is 8

4.10.2: Divide the Chuck-a-Luck (see Example 4.16) probability space into three regions, as in Example 4.33. Use this division and Equation 4.10 to compute the probability that

a) There are at least two 1's showing
b) All three dice are 1
c) There is exactly one 1 showing

4.10.3: Show that in Chuck-a-Luck, the probability of any event in which all three dice have different values is twice the probability of any event where one number appears exactly twice and six times the probability of any event in which all three dice show the same number.

4.10.4*: Show by induction on n that if there are n experiments, each of which is independent of those that go before, then the probability of any sequence of outcomes is the product of the probabilities of each outcome in its own experiment.

4.10.5*: Show that if $\text{PROB}(F/E) = \text{PROB}(F)$, then $\text{PROB}(E/F) = \text{PROB}(E)$. Thus, show that if experiment X_1 is independent of experiment X_2, then X_2 is independent of X_1.

4.10.6*: Consider the set of sequences of seven letters chosen from `W` and `L`. We may think of these sequences as representing the outcomes of a match of seven games, where `W` means the first team wins the game and `L` means the second team wins the game. The match is won by the first team to win four games (thus, some games may never get played, but we need to include their hypothetical outcomes in the points in order that we have a probability space of equally likely points).

a) What is the probability that a team will win the match, given that it has won the first game?

b) What is the probability that a team will win the match, given that it has won the first two games?

c) What is the probability that a team will win the match, given that it has won two out of the first three games?

4.10.7**: There are three prisoners, A, B, and C. They are told one and only one is to be shot and that the guard knows who. A asks the guard to tell him the name of one of the other prisoners who will *not* be shot. The guard answers that B will not be shot.

A reasons that either he or C will be shot, so the probability that A will be shot is 1/2. On the other hand, reasons A, no matter who is to be shot, the guard knows somebody besides A who will not be shot, so he always has an answer to A's question. Therefore, the asking and answering of the question provides no information about whether or not A is to be shot, so the probability that A will be shot is still 1/3, as it was before the question was asked.

What is the true probability that A will be shot after the sequence of events described above? *Hint*: You need to construct an appropriate probability space, one that represents not only the experiment in which a prisoner is chosen to be shot but also the possibility that the guard has a choice of whether to answer "B" or "C," and the experiment in which he chooses one if so.

4.10.8*: Suppose that E is an event in a space that is partitioned into k regions $R_1, R_2, \ldots, R_k$. Show that

$$\text{PROB}(R_j/E) = \frac{\text{PROB}(R_j)\text{PROB}(E/R_j)}{\sum_{i=1}^{k} \text{PROB}(R_i)\text{PROB}(E/R_i)}$$

Bayes' Rule

This formula is called *Bayes' Rule.* It gives a value for the probability of R_j given that E has been observed. For Example 4.31, calculate $\text{PROB}(R_1/E)$, $\text{PROB}(R_2/E)$, and $\text{PROB}(R_3/E)$ using Bayes' rule.

4.11 Probabilistic Reasoning

An important application of probability in computing is in the design of systems that predict events. An example is a medical diagnosis system. Ideally, the process of diagnosis consists of performing tests or observing symptoms until the outcome of the tests and presence or absence of certain symptoms is sufficient for the physician to determine what disease the patient has. In practice, however, diagnoses are rarely certain. Rather, a diagnosis is the most likely disease, or a disease whose conditional probability, given the outcomes of the experiments that are the tests and observations of symptoms, is highest.

Let us consider an overly simple example that has the flavor of diagnosis using probability. Suppose it is known that when a patient has a headache, the probability that he has the flu is 50%. That is,

$$\textsc{prob}(\text{Flu}/\text{Headache}) = 0.5$$

In the above, we interpret Flu as the name of an event that can be interpreted as "the patient has the flu." Similarly, Headache is the name of the event that the patient complains of a headache.

Suppose we also know that when the patient's temperature is measured at 102 (Fahrenheit) or above, the probability is 60% that he has the flu. If we allow Fever to be the name of the event that the patient's temperature is at least 102, then we can write this observation as

$$\textsc{prob}(\text{Flu}/\text{Fever}) = 0.6$$

Now, consider the following diagnosis situation. A patient comes to the doctor complaining of a headache. The doctor takes his temperature and finds it is 102. What is the probability that the patient has the flu?

The situation is suggested by Fig. 4.18. There we see the three events Flu, Headache, and Fever, which together divide the space into 8 regions, which we indicate by letters a through h. For example, c is the event that the patient has a headache and flu, but not a fever.

The given information about probabilities puts some constraints on the sizes of the events in Fig. 4.18. Let us use the letters a through h as standing not only for the regions indicated in Fig. 4.18 but as the probabilities of those events. Then the condition that $\textsc{prob}(\text{Flu}/\text{Headache}) = 0.5$ says that the sum of regions $c + f$ is half the total size of the headache event, or put another way:

$$c + f = d + g \tag{4.13}$$

Similarly, the fact that $\textsc{prob}(\text{Flu}/\text{Fever}) = 0.6$ says that $e + f$ is 3/5 of the total size of the Fever event, or:

$$e + f = \tfrac{3}{2}(g + h) \tag{4.14}$$

Now, let us interpret the question: what is the probability of flu, given both a fever and a headache? The fact that there is both fever and headache says that we are either in region f or region g. In region f the diagnosis of flu is correct, and in g it is not. Thus, the probability of flu is $f/(f + g)$.

What is the value of $f/(f + g)$? The answer may be surprising. We have absolutely no information about the probability of flu; it could be 0 or 1 or anything in between. Here are two examples of how the points of the probability space of Fig. 4.18 could actually be distributed.

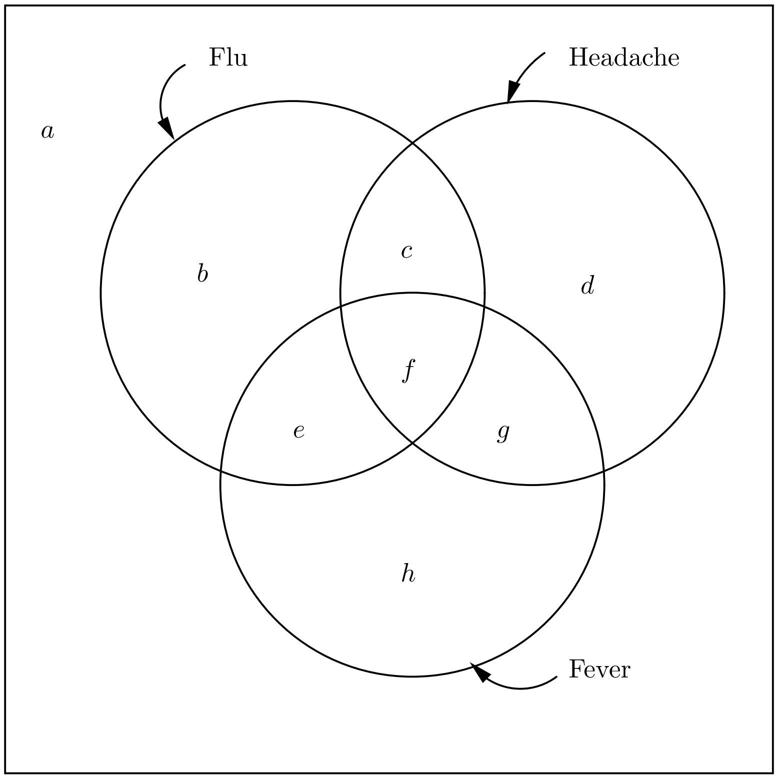


Fig. 4.18. The events Flu, Headache, and Fever.

✦ **Example 4.35.** Suppose that in Fig. 4.18 the probabilities associated with the various events are: $d = f = 0.3$, $a = h = 0.2$, and the four other regions have probability 0. Note that these values satisfy the constraining equations (4.13) and (4.14). In this example, $f/(f+g) = 1$; that is, a patient with both a headache and fever is certain to have the flu. Then the probability space of Fig. 4.18 actually looks like that of Fig. 4.19. There we see that whenever a patient has both a fever and headache, he has the flu, and conversely, whenever he has the flu, he has both fever and headache.[4] ✦

✦ **Example 4.36.** Another example is given by the probabilities $c = g = 0.2$, $a = e = 0.3$, with other probabilities 0. Again, Equations (4.13) and (4.14) are satisfied. Now, however, $f/(f+g) = 0$. That is, if you have both a fever and headache, then you are certain *not* to have the flu, a rather doubtful statement, but one that is not ruled out by Equations (4.13) and (4.14). The situation is shown in Fig. 4.20. ✦

[4] Although there are other examples where $b \neq 0$ — that is, one can have the flu and yet have neither fever nor headache — yet $f/(f+g) = 1$.

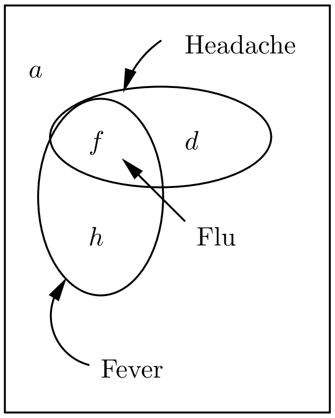


Fig. 4.19. Example of space where Fever and Headache guarantee Flu.

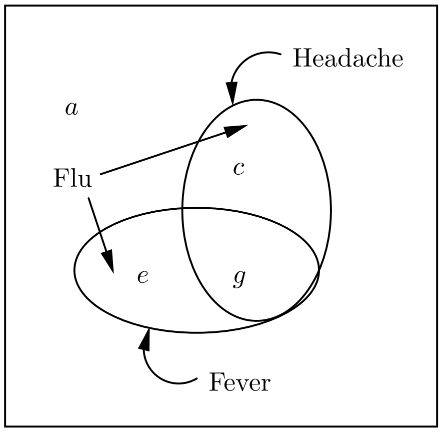


Fig. 4.20. Example of space where Fever and Headache guarantee no Flu.

Probability of the OR of Two Events

If we cannot tell anything about what happens when a patient has both fever and headache in the previous scenario, we might wonder whether there is anything we *can* say. In simpler situations there is indeed some limit to how probabilities behave when events are combined. Perhaps the most straightforward situation occurs when we combine two events with *disjunction* or logical-`OR`.

Disjunction

✥ **Example 4.37.** Referring again to the situation of Fig. 4.18, suppose we are told that at any time, 2% of the population has a fever and 3% of the population has a headache. That is, the size of the event Fever is 0.02, and the size of the event Headache is 0.03. What fraction of the population has either a fever or a headache, or both?

The answer is that between 3% and 5% of the population has at least one. To see why, let us do some calculation in terms of the eight regions defined in Fig. 4.18.

If Fever has probability 0.02, that says

$$e + h + f + g = 0.02 \tag{4.15}$$

If Headache has probability 0.03, then

$$c + d + f + g = 0.03 \tag{4.16}$$

We are asked what is the size of the regions that are in either Fever or Headache or both; that is, how large is $e + h + f + g + c + d$?

If we add (4.15) and (4.16), we get $e + h + 2(f + g) + c + d = 0.05$, or put another way,

$$e + h + f + g + c + d = 0.05 - (f + g) \tag{4.17}$$

Since the probability of Fever-or-Headache is the left side of (4.17), it is also the right side, or $0.05 - (f + g)$.

At the minimum, $f + g$ is 0, so the probability of Fever-or-Headache can be as high as 0.05, but no higher. That is, it is possible that fever and headache never occur together. Then the regions f and g are empty, $e + h = 0.02$, and $c + d = 0.03$. In this case, the probability of Fever-or-Headache is the sum of the probability of Fever and the Probability of Headache.

What is the maximum value $f + g$ could have? Surely $f + g$ is no bigger than the entire Fever event and no bigger than the entire Headache event. Since Fever is smaller, we see that $f + g \leq 0.02$. Thus, the smallest the probability of Fever-or-Headache could be is $0.05 - 0.02$, or 0.03. This result happens to be the probability of Headache, the larger of the two events. That is no coincidence. Another way to look at it is that the smallest size Fever-or-Headache could have occurs when the smaller of the two events is wholly contained within the larger. In this example, that occurs when $e + h = 0$, and Fever is contained within Headache. In that case, you cannot have a fever unless you also have a headache, so the probability of Fever-or-Headache is the same as the probability of Headache alone, or 0.03. ✦

Rule for sums

We can generalize our explorations of Example 4.37 to any two events. The *rule for sums* is as follows. If E and F are any two events, and G is the event that either E or F or both occurs, then

$$\max\bigl(\text{PROB}(E), \text{PROB}(F)\bigr) \leq \text{PROB}(G) \leq \text{PROB}(E) + \text{PROB}(F) \tag{4.18}$$

That is, the probability of E-or-F is between the larger of the probabilities of E and F and the sum of those probabilities.

The same idea holds within any other event H. That is, all the probabilities in (4.18) may be made conditional on event H, giving us the more general rule

$$\begin{aligned}\max\bigl(\text{PROB}(E/H), \text{PROB}(F/H)\bigr) &\leq \text{PROB}(G/H) \\ &\leq \text{PROB}(E/H) + \text{PROB}(F/H)\end{aligned} \tag{4.19}$$

✦ **Example 4.38.** Suppose, in the scenario of Fig. 4.18 we are told that 70% of all people with the flu have a fever, and 80% of all people with the flu have a headache. Then in (4.19), Flu is the event H, E is the event Fever, F is Headache, and G is Headache-or-Fever. We are told that $\text{PROB}(E/H) = \text{PROB}(\text{Fever}/\text{Flu}) = 0.7$, and $\text{PROB}(F/H) = \text{PROB}(\text{Headache}/\text{Flu}) = 0.8$.

Rule (4.19) says that $\text{PROB}(G/H)$ is at least the larger of 0.7 and 0.8. That is, if you have the flu, then the probability that you have a fever or headache or both is at least 0.8. Rule (4.19) also says that $\text{PROB}(G/H)$ is at most

$$\text{PROB}(E/H) + \text{PROB}(F/H)$$

or $0.7+0.8 = 1.5$. However, that upper bound is not useful. We know that no event can have probability greater than 1, so 1 is a better upper bound on $\text{PROB}(G/H)$. ✦

Probability of the AND of Events

Suppose we are again told that the probability of Fever is 0.02 and the probability of Headache is 0.03. What is the probability of Fever-and-Headache, that is, the probability that a person has both fever and headache? As for the `OR` of two events, we cannot tell exactly, but sometimes we can put some limits on the probability of the *conjunction* (logical `AND`) of two events.

Conjunction

In terms of Fig. 4.18, we are asking how big $f+g$ can be. We already observed in connection with the `OR` of events that $f + g$ is largest when the smaller of the two events, Fever in this case, is wholly contained within the other. Then, all the probability of Fever is concentrated in $f+g$, and we have $f+g = 0.02$. That is, the probability of Fever-and-Headache is at most 0.02, the probability of Fever alone. In general, the probability of the `AND` of two events cannot exceed the probability of the smaller.

How small can $f + g$ be? Evidently there is nothing that prevents Fever and Headache from being disjoint, so $f + g$ could be 0. That is, there may be no one who has both a fever and headache.

Yet the above idea does not generalize completely. Suppose that instead of the tiny probabilities 0.02 and 0.03 for the events Fever and Headache, we found that 60% had a fever and 70% had a headache. Is it possible that no one has both a fever and a headache? If $f + g = 0$ in this situation, then $e + h = 0.6$, and $c + d = 0.7$. Then $e + h + c + d = 1.3$, which is impossible. That is, we would have in Fig. 4.18 an event, $e + h + c + d$, with a probability greater than 1.

Evidently, the size of the `AND` of two events cannot be smaller than the sum of the probabilities of the events minus 1. If not, then the `OR` of the same two events has a probability greater than 1. This observation is summarized in the *rule for products*. If E and F are two events, and G is the event that both of E and F occur, then

Rule for products

$$\text{PROB}(E) + \text{PROB}(F) - 1 \le \text{PROB}(G) \le \min\big(\text{PROB}(E), \text{PROB}(F)\big)$$

As with the rule for sums, the same idea applies to probabilities that are conditional upon some other event H. That is,

$$\begin{aligned}\text{PROB}(E/H) + \text{PROB}(F/H) - 1 &\le \text{PROB}(G/H)\\ &\le \min\big(\text{PROB}(E/H), \text{PROB}(F/H)\big)\end{aligned} \qquad (4.20)$$

✦ **Example 4.38.** Again referring to Fig. 4.18, suppose that 70% of those with the flu have a fever and 80% have a headache. How many have both a fever and a headache? According to (4.20) with H the event Flu, the probability of both a fever and headache, given that the person has the flu, is at least $0.7 + 0.8 - 1 = 0.5$ and at most $\min(0.7, 0.8) = 0.7$. ✦

Summary of Rules Involving Several Events

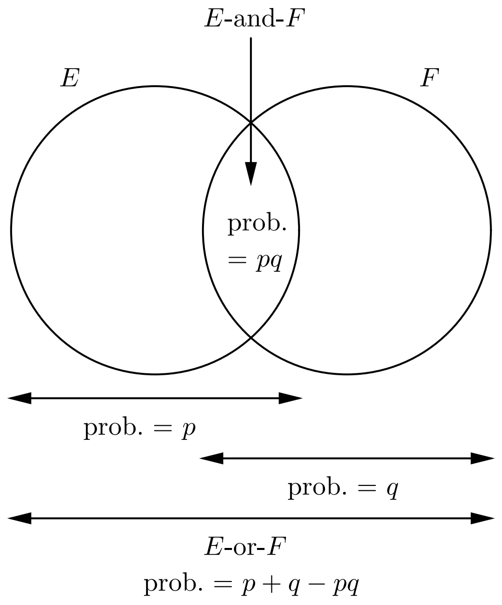
The following summarizes rules of this section and rules about independent events from the last section. Suppose E and F are events with probabilities p and q, respectively. Then

- The probability of event E-or-F (i.e., at least one of E and F) is at least $\max(p, q)$ and at most $p + q$ (or 1 if $p + q > 1$).
- The probability of event E-and-F (i.e., both E and F) is at most $\min(p, q)$ and at least $p + q - 1$ (or 0 if $p + q < 1$).
- If E and F are independent events, then the probability of E-and-F is pq.
- If E and F are independent events, then the probability of E-or-F is $p+q-pq$.

The latter rule requires some thought. The probability of E-or-F is $p+q$ minus the fraction of the space that is in both events, since the latter space is counted twice when we add the probabilities of E and F. The points in both E and F are exactly the event E-and-F, whose probability is pq. Thus,

$$\text{PROB}(E\text{-or-}F) = \text{PROB}(E) + \text{PROB}(F) - \text{PROB}(E\text{-and-}F) = p + q - pq$$

The diagram below illustrates the relationships between these various events.



Some Ways to Deal With Relationships Among Events

Compound event

In applications that require us to compute the probability of *compound* events (events that are the `AND` or `OR` of several other events), often we do not need to know exact probabilities. Rather, we need to determine what is the most likely situation or a situation that has high probability (i.e., probability close to 1). Thus, the range of probabilities that a compound event may not present a great problem, as long as we can deduce that the probability of the event is "high."

For instance, in the medical diagnosis problem introduced in Example 4.35, we may never be able to deduce with probability 1 that the patient has the flu. But as long as the combination of observed symptoms and symptoms not present in the patient together allow us to conclude that the probability he has the flu is very high, then it may make good sense to treat the patient for flu.

However, we observed in Example 4.35 that we could say essentially nothing about the probability that a patient with both a headache and fever has the flu, even though we know that each symptom by itself strongly suggests the flu. A real reasoning system needs more information or more rules from which it can estimate probabilities. As a simple example, we might be given explicitly the probability PROB(Flu/Headache-and-Fever). That would settle the question immediately.

However, if there are n events $E_1, E_2, \ldots, E_n$ that in some combination might let us conclude another event F, then we need to give explicitly $2^n - 1$ different probabilities. These are the conditional probabilities of F given each set of one or more of $E_1, E_2, \ldots, E_n$.

✦ **Example 4.40.** For the case $n = 2$, such as Example 4.35, we need only give three conditional probabilities. Thus we might assert, as we did previously, that PROB(Flu/Fever) = 0.6 and PROB(Flu/Headache) = 0.5. Then, we might add information such as PROB(Flu/Fever-and-Headache) = 0.9. ✦

Implication of events

To avoid having to specify an exponential number of conditional probabilities, there are many types of restrictions that have been used to help us deduce or estimate probabilities. A simple one is a statement that one event *implies* another; that is, the first event is a subset of the second event. Often, such information tells us something useful.

✦ **Example 4.41.** Suppose we state that whenever one has the flu, one is sure to have a headache. In terms of Fig. 4.18, we are saying that regions b and e are empty. Suppose also that whenever one has the flu, one also has a fever. Then region c of Fig. 4.18 is also empty. Figure 4.22 suggests the simplification to Fig. 4.18 that results from these two assumptions.

Under the conditions that b, c, and e are all 0, and again assuming that PROB(Flu/Headache) = 0.5 and PROB(Flu/Fever) = 0.6, we can rewrite Equations (4.13) and (4.14) as

$$f = d + g$$
$$f = \frac{3}{2}(g + h)$$

Since d and h are both at least 0, the first equation says $f \geq g$ and the second says $f \geq 3g/2$.

Now, let us see what we know about the probability of flu, given both fever and headache, that is, PROB(Flu/Fever-and-Headache). This conditional probability in either Fig. 4.18 or Fig. 4.22 is $f/(f + g)$. Since $f \geq 3g/2$, we conclude that $f/(f+g) \geq 0.6$. That is, the probability is at least 0.6 that a patient with headache and fever has the flu. ✦

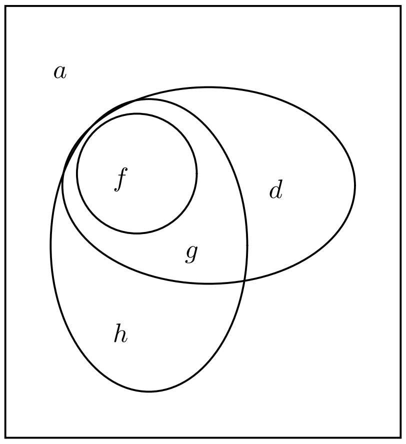


Fig. 4.22. Here, Flu implies both Headache and Fever.

We can generalize Example 4.41 to apply to any three events, two of which are implied by the third. Suppose E, F, and G are events, and

$$\text{PROB}(E/G) = \text{PROB}(F/G) = 1$$

That is, whenever G occurs, E and F are certain to occur as well. Suppose further that $\text{PROB}(G/E) = p$ and $\text{PROB}(G/F) = q$. Then

$$\text{PROB}(G/E\text{-and-}F) \geq \max(p, q) \tag{4.21}$$

The reason Equation (4.21) holds can be seen from Fig. 4.22, if we interpret Flu as G, Fever as E, and Headache as F. Then $p = f/(f+g+h)$ and $q = f/(f+g+d)$. Since d and h are at least 0, it follows that $p \leq f/(f+g)$ and $q \leq f/(f+g)$. But $f/(f+g)$ is $\text{PROB}(G/E\text{-and-}F)$. Thus, this conditional probability is equal to or greater than the larger of p and q.

EXERCISES

4.11.1: Generalize the rule of sums and the rule of products to more than two events. That is, if $E_1, E_2, \ldots, E_n$ are events with probabilities $p_1, p_2, \ldots, p_n$, respectively,

a) What can we say about the probability that at least one of the n events occur?
b) What can we say about the probability that all n events occur?

4.11.2*: If $\text{PROB}(F/E) = p$, what, if anything, can we say about

a) $\text{PROB}(F/\bar{E})$
b) $\text{PROB}(\bar{F}/E)$
c) $\text{PROB}(\bar{F}/\bar{E})$

Recall that $\bar{E}$ is the complement event for E and $\bar{F}$ is the complement event for F.

Other Applications of Probabilistic Reasoning

We have seen in this section a tiny illustration of an important application of probabilistic reasoning: medical diagnosis. Here are some of the other areas in which similar ideas appear in computer solutions.

- *Systems diagnosis.* A device fails, exhibiting some incorrect behavior. For example, a computer's screen is blank but the disk is running. What has caused the problem?
- *Corporate planning.* Given probabilities of economic conditions, such as a rise in inflation, a decrease in supply of a certain commodity, and so on, what strategies have the greatest probability of success?
- *Intelligent appliances.* State-of-the-art appliances of many sorts use probabilistic reasoning (often referred to as "fuzzy logic") to make decisions for the user. For example, a washing machine can spin and weigh its load and predict the most likely kind of fabric (e.g., wash-and-wear or woolens), adjusting the cycle accordingly.

Fuzzy logic

4.11.3: An intelligent building control tries to predict whether it will be a "cold" night, which we shall take to mean that the nighttime low is at least 20 degrees colder than the daytime high. It knows that when the sunlight falling on its sensor is high just before sunset, there is a 60% probability of a cold night (because there is apparently no cloud cover, allowing heat to radiate more easily from the earth). It also knows that if the change in temperature the hour after sunset is at least 5 degrees, then the probability is 70% of a cold night. Refer to these three events as Cold, High, and Dropping, respectively. Let us suppose also that PROB(High) = 0.4 and PROB(Dropping) = 0.3.

a) Give upper and lower limits on PROB(High-and-Dropping).

b) Give upper and lower limits on PROB(High-or-Dropping).

c) Suppose we are also told that whenever it is going to be cold at night, the sunlight sensor reads high, and the temperature drops at least 4 degrees after sunset, i.e., PROB(High/Cold) and PROB(Dropping/Cold) are both 1. Give upper and lower limits on PROB(Cold/High-and-Dropping).

d**) Under the same assumption as part (c), give upper and lower limits on

PROB(Cold/High-or-Dropping)

Note that this problem requires reasoning not covered in the section.

4.11.4*: In many situations, such as Example 4.35, two or more events mutually reinforce a conclusion. That is, we expect intuitively that whatever

PROB(Flu/Headache)

may be, being told that the patient has a fever as well as a headache increases the probability of flu. Say that event E *reinforces* event F in the conclusion G if PROB(G/E-and-F) $\geq$ PROB(G/F). Show that if events E and F each reinforce the other in the conclusion G, then Equation (4.21) holds. That is, the conditional

Reinforcing events

probability of G given E and F is at least the larger of the conditional probability of G given E and the conditional probability of G given F.

✦ 4.12 Expected Value Calculations

Commonly, the possible outcomes of an experiment have associated values. In this section, we shall use simple gambling games as examples, where money is won or lost depending on the outcome of the experiment. In the next section, we shall discuss more complex examples from computer science, where we compute the expected running time of certain algorithms.

Payoff function

Suppose we have a probability space and a *payoff function* f on the points of that space. The *expected value* of f is the sum over all points x of $f(x)\text{PROB}(x)$. We denote this value by $\text{EV}(f)$. When all points are equally likely, we can compute the expected value $\text{EV}(f)$ by

1. Summing $f(x)$ for all x in the space and then
2. Dividing by the number of points in the space.

Mean value

The expected value is sometimes called the *mean* value and it can be thought of as a "center of gravity."

✦ **Example 4.42.** Suppose the space is the six points representing the outcomes of the throw of a fair die. These points are naturally thought of as integers 1 through 6. Let the payoff function be the identity function; that is, $f(i) = i$ for $i = 1, 2, \ldots, 6$. Then the expected value of f is

$$\begin{aligned}\text{EV}(f) &= \big(f(1) + f(2) + f(3) + f(4) + f(5) + f(6)\big)/6 \\ &= (1 + 2 + 3 + 4 + 5 + 6)/6 = 21/6 = 3.5\end{aligned}$$

That is, the expected value of the number on a die is 3.5.

As another example, let g be the payoff function $g(i) = i^2$. Then, for the same experiment — the throw of one die — the expected value of g is

$$\begin{aligned}\text{EV}(g) &= (1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2)/6 \\ &= (1 + 4 + 9 + 16 + 25 + 36)/6 = 91/6 = 15.17\end{aligned}$$

Informally, the expected value for the square of the number thrown on a die is 15.17 ✦

✦ **Example 4.43.** Let us reconsider the game of Chuck-a-Luck, first introduced in Example 4.16. The payoff rules for this game are as follows. A player bets one dollar on a number. If that number comes up one or more times, then the player receives as many dollars as his number appears. If the number does not come up at all, then the player loses his dollar.

The probability space for Chuck-a-Luck is the 216 points consisting of the triples of numbers between 1 and 6. These points represent the outcome of the toss of three dice. Let us suppose that the player bets on number 1. It should be clear that the expected value of the player's winnings or losings does not depend on the number bet on, as long as the dice are fair.

The payoff function f for this game is:

0. $g(i,j,k) = -1$ if none of i, j, or k is 1. That is, the player loses a dollar if there are no 1's.

1. $g(i,j,k) = 1$ if exactly one of i, j, or k is 1.

2. $g(i,j,k) = 2$ if exactly two of i, j, or k are 1.

3. $g(i,j,k) = 3$ if all three of i, j, or k are 1.

Our problem then is to average g over the 216 points. Since enumeration of all these points is tedious, we are better off trying to count the number of points with each of the four different outcomes.

First, let us count the number of triples with no 1's. There are five numbers to chose from in each position, so we have an assignment problem as in Section 4.2. There are thus $5^3 = 125$ points with no 1's. These 125 points contribute -125 to the sum of the payoffs, by rule (0) above.

Next, let us count the number of triples with exactly one 1. The 1 can appear in any of the three places. For each place that holds the 1, there are choices of five numbers for each of the other two places. Thus, the number of points with exactly one 1 is $3 \times 5 \times 5 = 75$. These points contribute $+75$ to the payoff by rule (1).

The number of points with all three 1's is clearly one, so this possibility contributes $+3$ to the payoff. The remaining $216 - 125 - 75 - 1 = 15$ points must have two 1's, so these 15 points contribute $+30$ to the payoff by rule (2).

Finally, we can compute the expected value of the game by adding the payoffs from each of the four types of points and dividing by the total number of points. Thus calculation is

$$\text{EV}(f) = (-125 + 75 + 30 + 3)/216 = -17/216 = -0.079$$

That is, on the average, the player loses almost 8 cents for every dollar wagered. This result may be surprising, since the game superficially looks like an even bet. This point is discussed in the exercises. ✦

As Example 4.43 suggests, it is sometimes easier to group the points according to the value of their payoff function. In general, suppose we have a probability space with a payoff function f, and f has a finite number of different values that it produces. For instance, in Example 4.43 f produced only the values -1, 1, 2, and 3. For each value v produced by f, let E_v be the event consisting of the points x such that $f(x) = v$. That is, E_v is the set of points on which f produces value v. Then

$$\text{EV}(f) = \sum_v v \ \text{PROB}(E_v) \tag{4.22}$$

In the common case where the points have equal probability, let n_v be the number of points in event E_v and let n be the total number of points in the space. Then $\text{PROB}(E_v)$ is n_v/n, and we may write

$$\text{EV}(f) = \Bigl(\sum_v v n_v\Bigr)/n$$

✦ **Example 4.44.** In Example 4.25, we introduced the game of Keno, and computed the probability of guessing three out of 5 correct. Now let us compute the expected value of the payoff in the 5-spot game of Keno. Recall that in the 5-spot game, the player guesses five numbers from 1 to 80. When the game is played, twenty numbers from 1 to 80 are selected. The player wins if three or more of those twenty numbers are among the five he selected.

However, the payoff depends on how many of the player's five numbers are correct. Typically, for a \$1 bet, the player receives \$2 if he guesses three out of five (i.e., the player has a net gain of \$1). If he guesses four out of five, he receives \$15, and for guessing all five, he is rewarded with \$300. If he guesses fewer than three correctly, the player receives nothing, and loses his \$1.

In Example 4.25, we calculated the probability of guessing three out of five to be 0.08394 (to four significant places). We can similarly calculate that the probability of guessing four out of five is 0.01209, and the probability of guessing all five is 0.0006449. Then, the probability of guessing fewer than three is 1 minus these three fractions, or 0.90333. The payoffs for fewer than three, for three, four, and five are -1, $+1$, $+14$, and $+299$, respectively. Thus, we may apply formula (4.22) to get the expected payoff of the 5-spot game of Keno. It is

$$0.90333 \times -1 + 0.08394 \times 1 + 0.01209 \times 14 + 0.0006449 \times 299 = -0.4573$$

Thus, the player loses almost 46 cents of every dollar he bets in this game. ✦

EXERCISES

4.12.1: Show that if we throw three dice, the expected number of 1's that will appear is 1/2.

4.12.2*: Since we win when there is a 1 and lose when there is not, why does not the fact in Exercise 4.12.1 imply that Chuck-a-Luck is an even game (i.e., the expected payoff by betting on 1, or any other number, is 0)?

4.12.3: Suppose that in a 4-spot game of Keno, where the player guesses four numbers, the payout is as follows: for guessing two, \$1 (i.e., the player gets his dollar back); for guessing three, \$4; for guessing all four, \$50. What is the expected value of the payout?

4.12.4: Suppose in a 6-spot game of Keno, the payouts are as follows: for guessing three, \$1; for four, \$4; for five, \$25; for guessing all six, \$1000. What is the expected value of the payout?

Fair game

4.12.5: Suppose we play a Chuck-a-Luck type of game with six dice. The player pays \$1 to play, bets on a number, and throws the dice. He is rewarded with \$1 for every time his selected number appears. For instance, if it appears once, the net payout is 0; if it appears twice, the net payout is +1, and so on. Is this a *fair game* (i.e., is the expected value of the payout 0)?

4.12.6*: Based on the style of payout suggested by Exercise 4.12.5, we could modify the payout of the standard 3-dice form of Chuck-a-Luck so that the player pays some amount to play. He is then rewarded with \$1 for every time his number appears. What is the proper amount the player should pay in order that this be a fair game?

4.13 Some Programming Applications of Probability

In this section, we shall consider two types of uses for probability calculations in computer science. The first is an analysis of the expected running time of an algorithm. The second is a new type of algorithm, often called a "Monte Carlo" algorithm, because it takes a risk of being incorrect. As we shall see, by adjusting parameters, it is possible to make Monte-Carlo algorithms correct with as high a probability as we like, except that we cannot reach probability 1, or absolute certainty.

A Probabilistic Analysis

Let us consider the following simple problem. Suppose we have an array of n integers, and we ask whether an integer x is an entry of the array `A[0..n-1]`. The algorithm in Fig.4.23 does as well as any. Note that it returns a type which we called `BOOLEAN`, defined to be a synonym for `int` in Section 1.6. Also in that section were defined the constants `TRUE` and `FALSE`, which stand for 1 and 0, respectively.

```
        BOOLEAN find(int x, int A[], int n)
        {
            int i;

(1)         for(i = 0; i < n; i++)
(2)             if(A[i] == x)
(3)                 return TRUE;
(4)         return FALSE;
        }
```

Fig. 4.23. Finding an element x in an array A of size n.

The loop of lines (1) – (3) examines each entry of the array, and if x is found there, immediately terminates the loop with `TRUE` as our answer. If x is never found, we reach line (4) and return `FALSE`. Let the time taken by the body of the loop and the loop incrementation and test be c. Let d be the time of line (4) and the initialization of the loop. Then if x is not found, the running time of the function of Fig. 4.23 is $cn + d$, which is $O(n)$.

However, suppose x is found; what is the running time of Fig. 4.23 then? Clearly, the earlier x is found, the less time it takes. If x were somehow guaranteed to be in `A[0]`, the time would be $O(1)$, since the loop would iterate only once. But if x were always at or near the end, the time would be $O(n)$.

Surely the worst case is when we find x at the last step, so $O(n)$ is a smooth and tight upper bound on the worst case. However, is it possible that the average case is much better than $O(n)$? In order to address this question, we need to define a probability space whose points represent the possible places in which x can be found. The simplest assumption is that x is equally likely to be placed in any of the n entries of array A. If so, then our space has n points, one representing each of the integers from 0 to $n - 1$, which are the bounds of the index of array A.

Our question then becomes: in this probability space, what is the expected value of the running time of the function of Fig. 4.23? Consider a point i in the space; i can be anything from 0 to $n-1$. If x is in `A[i]`, then the loop will iterate $i+1$ times. An upper bound on the running time is thus $ci+d$. This bound is off slightly in the constant d, since line (4) is never executed. However, the difference does not matter, since d will disappear when we translate to a big-oh expression anyway.

We must thus find the expected value of the function $f(i) = ci+d$ on this probability space. We sum $ci+d$ where i ranges from 0 to $n-1$, and then divide by n, the number of points. That is,

$$\text{EV}(f) = \Bigl(\sum_{i=0}^{n-1} ci + d\Bigr)/n = \bigl(cn(n-1)/2 + dn\bigr)/n = c(n-1)/2 + d$$

For large n, this expression is about $cn/2$. Thus, $O(n)$ is the smooth and tight upper bound on this expected value. That is, the expected value is, to within a constant factor of about 2, the same as the worst case. This result makes intuitive sense. If x is equally likely to be anywhere in the array, it will "typically" be half way down the array, and we therefore do about half the work that would be done if x were not in the array at all, or if it were at the last element.

Algorithms That Use Probability

Deterministic algorithm

The algorithm of Fig. 4.23 was deterministic, in the sense that it always does the same thing on the same data. Only the analysis of the expected running time uses probability calculations. Almost every algorithm we meet is deterministic. However, there are some problems that are better solved with an algorithm that is not deterministic, but uses a selection from a probability space in some essential way. Making such a selection from an imagined probability space is not hard; we use a random number generator as discussed in the box in Section 4.9.

Monte-Carlo algorithm

One common type of probabilistic algorithm, called a *Monte-Carlo algorithm*, makes a random selection at each iteration. Based on this selection, it will either say "true," in which case that is guaranteed to be the correct answer, or it will say "I don't know," in which case the correct answer could be either "true" or "false." The possibilities are suggested by the probability space in Fig.4.24.

The probability that the algorithm will say "true" given that the answer is true is $a/(a+b)$. That is, this probability is the conditional probability of event a in Fig. 4.24 given a or b. As long as this probability p is greater than 0, we can iterate as many times as we like, and get rapidly decreasing probability of a failure. By "failure," we mean that the correct answer is "true," but on no iteration does the algorithm tell us so.

Since each iteration is an independent experiment, if the correct answer is "true" and we iterate n times, the probability that the algorithm will never say true is $(1-p)^n$. As long as $1-p$ is strictly less than 1, we know that $(1-p)^n$ will decrease rapidly as n increases. For example, if $p = 1/2$, then $1-p$ is also 1/2. The quantity $(0.5)^n$ is about 1/1000 for $n = 10$ (see the box in Section 4.2); it is 1/1000000 for $n = 20$, and so on, decreasing by a factor of about 1000 every time n increases by 10.

The Monte-Carlo algorithm, then, is to run the experiment n times. If the answer to any experiment is "true," then the algorithm answers "true." If all answers are "false," then the algorithm answers "false." Thus,

1. If the correct answer is "false," the algorithm will surely answer "false."

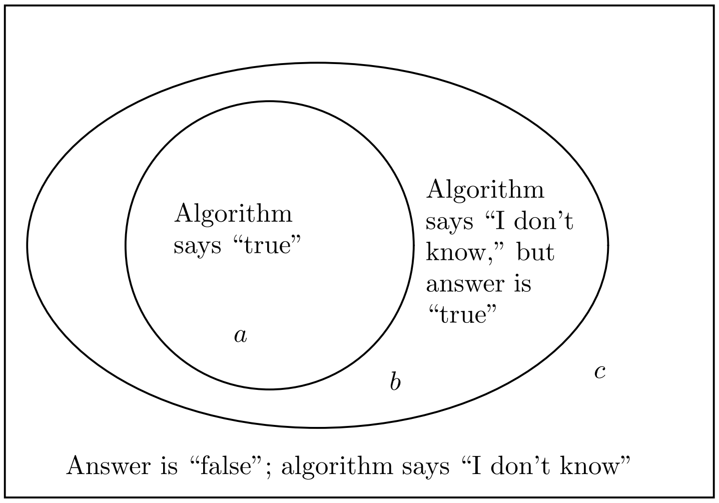

Algorithm says "true"
a
Algorithm says "I don't know," but answer is "true"
b
c
Answer is "false"; algorithm says "I don't know"

Fig. 4.24. Possible outcomes of one iteration of a Monte-Carlo algorithm.

2. If the correct answer is "true," then the algorithm answers "false" with probability $(1-p)^n$, which we assume is very small because n is chosen large enough to make it small. The algorithm answers "true" with probability $1-(1-p)^n$, which is presumably very close to 1.

Thus, there are no failures when the correct answer is "false" and very few failures when the correct answer is "true."

✦ **Example 4.45.** Here is an example of a problem that can be solved more efficiently using a Monte-Carlo algorithm. The XYZ computer company orders boxes of chips that are supposed to be tested at the factory to make sure they are all good. However, XYZ believes that some boxes have been leaving the factory untested, in which case the probability that any given chip is bad is 1/10. A simple approach solution would be for XYZ itself to test all the chips it receives, but this process is expensive and time-consuming. If there are n chips in a box, the test of a box takes $O(n)$ time.

A better approach is to use a Monte-Carlo algorithm. Select from each box k chips at random to be tested. If a chip is bad, answer "true" — the box had not been tested at the factory or else the bad chip would have been discovered. If the chip is good, answer "I don't know," and go on to the next chip. If all k chips that we test are good, declare the entire box to be good.

In terms of Fig. 4.24, region c represents the case that the chips are chosen from a good box; region b is the case that the box is untested, but the chip happens to be good; region a is the case that the box is untested, and the chip is bad. Our assumption that 1/10 of the chips are bad if the box is untested says that the area of the circle, a, is one-tenth of the area of the ellipse enclosing regions a and b.

Let us compute the probability of failure — all k chips are good, but the box is untested. The probability of saying "I don't know" after testing one chip is $1-\frac{1}{10}=0.9$. Since the events of testing each chip are independent, the probability that we shall say "I don't know" to each of k chips is $(0.9)^k$. Suppose we pick

$k = 131$. Then the probability of failure, $(0.9)^{131}$, is almost exactly 0.000001, or one in a million. That is, if the box is good, we shall never find a bad chip in it, so we surely say the box is good. If the box is untested, then with probability 0.999999 we find a bad chip among the 131 we test and say the box needs full testing. With probability 0.000001, the box is untested, but we say it is a good box and do not test the remaining chips in the box.

The running time of this algorithm is $O(1)$. That is, the time to test at most 131 chips is a constant, independent of n, the number of chips in the box. Thus, compared with the more obvious algorithm of testing all the chips, the cost has gone down from $O(n)$ to $O(1)$ per box, at the cost of making an error once in every million untested boxes.

Moreover, we can make the probability of error as small as we like, by changing the number of chips we test before concluding we have a good box. For example, if we double the number of tested chips, to 262, then we square the probability of failure, which then becomes one in a trillion, or 10^{-12}. Also, we could save a constant factor in time at the cost of a higher failure rate. For instance, if we halved the number of chips tested, to 66 per box, we would have a failure rate of about one in a thousand untested boxes. ✦

EXERCISES

4.13.1: Which of 377, 383, 391 is prime?

4.13.2: Suppose that we used the function of Fig. 4.23 to search for element x, but the probability of finding x in entry i is proportional to $n - i$. That is, we can imagine a probability space with $n(n+1)/2$ points, n of which represent situations where x is in `A[0]`, $n - 1$ points represent situations where x is in `A[1]`, and so on, down to one point that represents the possibility that x is in `A[n-1]`. What is the expected running time of the algorithm for this probability space?

4.13.3: In 1993, the National Basketball Association held a lottery in which the 11 teams that did not make the playoffs participated. At stake was the first choice in the player draft. The team with the worst record was given 11 tickets in the lottery, the next-worst team got 10, and so on, until the 11th-worst team was given one ticket. A ticket was picked at random and the first-place draft was awarded to the holder of that ticket. What is the expected value of the function $f(t)$ that is the finishing place (from the bottom) of the holder of the selected ticket t?

4.13.4**: The lottery described in Exercise 4.13.3 continued. The winning team had all their tickets removed, and another ticket was picked for second place in the draft. That winner's remaining tickets were removed, and a third ticket was selected for third place in the draft. What are the expected values of the finishing positions of the teams that got second and third places in the draft?

4.13.5*: Suppose that we are given an array of size n, either sorted or filled at random with integers. We wish to construct a Monte-Carlo algorithm that will either say "true" if it finds the array unsorted or will say "I don't know." By repeating the test k times, we'd like to know there is no more than a 2^{-k} probability of failure. Suggest such an algorithm. *Hint*: Make sure that your tests are independent. As an example of tests that are *not* independent, we might test whether `A[0]<A[1]` and that `A[1]<A[2]`. These tests are independent. However, if we then tested that

Testing Whether an Integer is a Prime

While Example 4.45 is not really a program, it still exhibits a useful algorithmic principle, and is in fact a realistic portrayal of a technique for measuring the reliability of manufactured goods. There are some interesting computer algorithms that use the Monte-Carlo idea as well.

Cryptography

Perhaps first among these is the problem of testing whether a number is a prime. That question is not an idle one of number theory. It turns out that many of the central ideas of computer security involve knowing that a very large number is a prime. Roughly, when we encrypt information using n-digit primes, decrypting without knowing the secret key appears to involve guessing from among almost all of the 10^n possibilities. By making n sufficiently large, we can ensure that "breaking" the code requires either tremendous luck or more computer time than is available.

Thus, we would like a way to test whether a very large number is a prime, and to do so in time that is much less than the value of that prime; ideally we'd like it to take time proportional to the number of digits (i.e., proportional to the logarithm of the number). It seems that the problem of detecting composite numbers (nonprimes) is not hard. For example, every even number except 2 is composite, so it looks like half the problem is solved already. Similarly, those divisible by 3 have a sum of digits that is divisible by 3, so we can write a recursive algorithm to test for divisibility by 3 that is only a tiny bit slower than linear in the number of digits. However, the problem is still tricky for many numbers. For example, one of 377, 383, and 391 is prime. Which?

There is a Monte-Carlo algorithm that tests for composite numbers. On each iteration, it has at least probability 1/2 of saying "true" if the number is composite, and never says "true" if the number is prime. The following is not the exact algorithm, but it works except for a small fraction of the composite numbers. The complete algorithm is beyond the scope of this book.

Fermat's theorem

The algorithm is based on *Fermat's theorem*, which states that if p is a prime, and a is any integer between 1 and $p-1$, then a^{p-1} leaves a remainder of 1 when divided by p. Moreover, it happens, except for that small number of "bad" composite numbers, that if a is chosen at random between 1 and $p-1$, then the probability is at least 1/2 that a^{p-1} will have a remainder other than 1 when divided by p. For example, let $p = 7$. Then $1^6, 2^6, \ldots, 6^6$ are respectively 1, 64, 729, 4096, 15625, and 46656. Their remainders when divided by 7 are all 1. However, if $p = 6$, a composite number, we have $1^5, 2^5, \ldots, 5^5$ equal to 1, 32, 243, 1024, and 3125, respectively, whose remainders when divided by 6 are 1, 2, 3, 4, and 5. Only 20% are 1.

Thus, the "algorithm" for testing whether a number p is a prime is to select k integers from 1 to $p-1$, independently and at random. If for any selected a we find the remainder of a^{p-1}/p to be other than 1, we say p is composite; otherwise, we say it is prime. If it weren't for the "bad" composites, we could say that the probability of failure is at most 2^{-k}, since composites meet the test for a given a with probability at least 1/2. If we make k something like 40, we have only a one in a trillion chance of accepting a composite as prime. To handle the "bad" composites, a more complex test is needed, however. This test is still polynomial in the number of digits in p, as is the simple test given above.

`A[0]<A[2]`, the test would not be independent, since knowing the first two hold we can be sure the third holds.

4.13.6**: Suppose we are given an array of size n filled with integers in the range 1 to n. These integers were either selected to be different, or they were selected at random and independently, so we can expect some equalities among entries in the array. Give a Monte-Carlo algorithm that has running time $O(\sqrt{n})$ and has at most a probability of 10^{-6} of saying the array was filled with distinct integers when in fact it was filled at random.

4.14 Summary of Chapter 4

The reader should remember the following formulas and paradigm problems for counting.

- The number of assignments of k values to n objects is k^n. The paradigm problem is painting n houses, each in any of k colors.
- We can arrange n distinct items in $n!$ different ways.
- We can select k items out of n and arrange the k selected items in any order in $n!/(n-k)!$ different ways. The paradigm problem is choosing the win, place, and show horses ($k = 3$) in a race with n horses.
- The number of ways to select m objects out of n, without order, is $\binom{n}{m}$, or $n!/\bigl(m!(n-m)!\bigr)$. The paradigm problem is dealing poker hands, where $n = 52$ and $m = 5$.
- If we want to order n items, some of which are identical, the number of ways to do so is computed as follows. Start with $n!$. Then, if one value appears $k > 1$ times among the n items, divide by $k!$. Perform this division for each value that appears more than once. The paradigm problem is counting anagrams of a word of length n, where we must divide $n!$ by $k!$ for each letter that appears k times in the word and $k > 1$.
- If we want to distribute n identical objects into m bins, we can do so in $\binom{n+m-1}{n}$ ways. The paradigm problem is distributing apples to children.
- If as above, some of the objects are not identical, we count the number of ways to distribute them to bins as follows. Start with $(n+m-1)!/(m-1)!$. Then, if there is a group of k identical objects, and $k > 1$, divide by $k!$. Perform the division for each value that appears more than once. The paradigm problem is distributing fruits of various kinds to children.

In addition, the reader should remember the following points about probability.

- A probability space consists of points, each of which is the outcome of an experiment. Each point x has associated with it a nonnegative number called the *probability of x*. The sum of the probabilities of the points in a probability space is 1.

- An event is a subset of the points in a probability space. The probability of an event is the sum of the probabilities of the points in the event. The probability of any event lies between 0 and 1.
- If all points are equally likely, the conditional probability of event F given event E is the fraction of the points in event E that are also in event F.
- Event E is independent of event F if the conditional probability of E given F is the same as the probability of E. If E is independent of F, then F is independent of E.
- The rule of sums says that the probability that one of two events E and F occurs is at least the larger of their probabilities and no greater than the sum of their probabilities (or no greater than 1 if the sum is above 1).
- The rule of products says that the probability that the outcome of an experiment is both in event E and in event F is no greater than the smaller of their probabilities and at least the sum of their probabilities minus 1 (or at least 0 if the latter is negative).

Finally, there are a number of applications to computer science of principles we learned in this chapter.

- Any sorting algorithm that works on arbitrary types of data that can be compared by "less than" requires time at least proportional to $n \log n$ to sort n items.
- The number of bit strings of length n is 2^n.
- Random number generators are programs that produce sequences of numbers that appear to be the result of independent experiments, although of course they are completely determined by the program.
- Systems for probabilistic reasoning need a way to express the probabilities of compound events that are formed from the occurrence of several events. The rules of sums and products sometimes help. We also learned some other simplifying assumptions that let us put bounds on the probability of compound events.
- Monte-Carlo algorithms use random numbers to produce either a desired result ("true") or no result at all. By repeating the algorithm a constant number of times, we can solve the problem at hand by concluding the answer is ("false") if none of the repetitions produces the answer "true." By selecting the number of repetitions, we can adjust the probability of incorrectly concluding "false" to be as low as we like, but not 0.

4.15 Bibliographic Notes for Chapter 4

A venerable and excellent introduction to combinatorics is Liu [1968]. Graham, Knuth, and Patashnik [1989] is a deeper discussion of the subject. Feller [1968] is the classic book on probability theory and its applications.

The Monte-Carlo algorithm for testing whether a number is a prime is from Rabin [1976]. A discussion of this algorithm and other interesting issues involving computer security and algorithms that use randomness in an important way can be found in Dewdeney [1993]. A more advanced discussion of these topics is presented in Papadimitriou [1994].

Dewdeney, A. K. [1993]. *The Turing Omnibus*, Computer Science Press, New York.

Feller, W. [1968]. *An Introduction to Probability Theory and Its Applications*, Third Edition, Wiley, New York.

Graham, R. L., D. E. Knuth, and O. Patashnik [1989]. *Concrete Mathematics: A Foundation for Computer Science*, Addison-Wesley, Reading, Mass.

Liu, C.-L. [1968]. *An Introduction to Combinatorial Mathematics*, McGraw-Hill, New York.

Papadimitriou, C. H. [1994]. *Computational Complexity*, Addison-Wesley, Reading, Mass.

Rabin, M. O. [1976]. "Probabilistic algorithms," in *Algorithms and Complexity: New Directions and Recent Trends* (J. F. Traub, ed.), pp. 21–39, Academic Press, New York.