5 Probabilistic Analysis and Randomized Algorithms

This chapter introduces probabilistic analysis and randomized algorithms. If you are unfamiliar with the basics of probability theory, you should read Sections C.1–C.4 of Appendix C, which review this material. We'll revisit probabilistic analysis and randomized algorithms several times throughout this book.

5.1 The hiring problem

Suppose that you need to hire a new office assistant. Your previous attempts at hiring have been unsuccessful, and you decide to use an employment agency. The employment agency sends you one candidate each day. You interview that person and then decide either to hire that person or not. You must pay the employment agency a small fee to interview an applicant. To actually hire an applicant is more costly, however, since you must fire your current office assistant and also pay a substantial hiring fee to the employment agency. You are committed to having, at all times, the best possible person for the job. Therefore, you decide that, after interviewing each applicant, if that applicant is better qualified than the current office assistant, you will fire the current office assistant and hire the new applicant. You are willing to pay the resulting price of this strategy, but you wish to estimate what that price will be.

The procedure HIRE-ASSISTANT on the facing page expresses this strategy for hiring in pseudocode. The candidates for the office assistant job are numbered 1 through n and interviewed in that order. The procedure assumes that after interviewing candidate i, you can determine whether candidate i is the best candidate you have seen so far. It starts by creating a dummy candidate, numbered 0, who is less qualified than each of the other candidates.

The cost model for this problem differs from the model described in Chapter 2. We focus not on the running time of HIRE-ASSISTANT, but instead on the fees paid for interviewing and hiring. On the surface, analyzing the cost of this algorithm

```
HIRE-ASSISTANT(n)
1  best = 0          // candidate 0 is a least-qualified dummy candidate
2  for i = 1 to n
3      interview candidate i
4      if candidate i is better than candidate best
5          best = i
6          hire candidate i
```

may seem very different from analyzing the running time of, say, merge sort. The analytical techniques used, however, are identical whether we are analyzing cost or running time. In either case, we are counting the number of times certain basic operations are executed.

Interviewing has a low cost, say c_i, whereas hiring is expensive, costing c_h. Letting m be the number of people hired, the total cost associated with this algorithm is $O(c_i n + c_h m)$. No matter how many people you hire, you always interview n candidates and thus always incur the cost $c_i n$ associated with interviewing. We therefore concentrate on analyzing $c_h m$, the hiring cost. This quantity depends on the order in which you interview candidates.

This scenario serves as a model for a common computational paradigm. Algorithms often need to find the maximum or minimum value in a sequence by examining each element of the sequence and maintaining a current "winner." The hiring problem models how often a procedure updates its notion of which element is currently winning.

Worst-case analysis

In the worst case, you actually hire every candidate that you interview. This situation occurs if the candidates come in strictly increasing order of quality, in which case you hire n times, for a total hiring cost of $O(c_h n)$.

Of course, the candidates do not always come in increasing order of quality. In fact, you have no idea about the order in which they arrive, nor do you have any control over this order. Therefore, it is natural to ask what we expect to happen in a typical or average case.

Probabilistic analysis

Probabilistic analysis is the use of probability in the analysis of problems. Most commonly, we use probabilistic analysis to analyze the running time of an algorithm. Sometimes we use it to analyze other quantities, such as the hiring cost in

procedure HIRE-ASSISTANT. In order to perform a probabilistic analysis, we must use knowledge of, or make assumptions about, the distribution of the inputs. Then we analyze our algorithm, computing an average-case running time, where we take the average, or expected value, over the distribution of the possible inputs. When reporting such a running time, we refer to it as the ***average-case running time***.

You must be careful in deciding on the distribution of inputs. For some problems, you may reasonably assume something about the set of all possible inputs, and then you can use probabilistic analysis as a technique for designing an efficient algorithm and as a means for gaining insight into a problem. For other problems, you cannot characterize a reasonable input distribution, and in these cases you cannot use probabilistic analysis.

For the hiring problem, we can assume that the applicants come in a random order. What does that mean for this problem? We assume that you can compare any two candidates and decide which one is better qualified, which is to say that there is a total order on the candidates. (See Section B.2 for the definition of a total order.) Thus, you can rank each candidate with a unique number from 1 through n, using $rank(i)$ to denote the rank of applicant i, and adopt the convention that a higher rank corresponds to a better qualified applicant. The ordered list $\langle rank(1), rank(2), \ldots, rank(n)\rangle$ is a permutation of the list $\langle 1, 2, \ldots, n\rangle$. Saying that the applicants come in a random order is equivalent to saying that this list of ranks is equally likely to be any one of the $n!$ permutations of the numbers 1 through n. Alternatively, we say that the ranks form a ***uniform random permutation***, that is, each of the possible $n!$ permutations appears with equal probability.

Section 5.2 contains a probabilistic analysis of the hiring problem.

Randomized algorithms

In order to use probabilistic analysis, you need to know something about the distribution of the inputs. In many cases, you know little about the input distribution. Even if you do know something about the distribution, you might not be able to model this knowledge computationally. Yet, probability and randomness often serve as tools for algorithm design and analysis, by making part of the algorithm behave randomly.

In the hiring problem, it may seem as if the candidates are being presented to you in a random order, but you have no way of knowing whether they really are. Thus, in order to develop a randomized algorithm for the hiring problem, you need greater control over the order in which you'll interview the candidates. We will, therefore, change the model slightly. The employment agency sends you a list of the n candidates in advance. On each day, you choose, randomly, which candidate to interview. Although you know nothing about the candidates (besides their names), we have made a significant change. Instead of accepting the order given

to you by the employment agency and hoping that it's random, you have instead gained control of the process and enforced a random order.

More generally, we call an algorithm ***randomized*** if its behavior is determined not only by its input but also by values produced by a ***random-number generator***. We assume that we have at our disposal a random-number generator RANDOM. A call to RANDOM(a, b) returns an integer between a and b, inclusive, with each such integer being equally likely. For example, RANDOM$(0, 1)$ produces 0 with probability $1/2$, and it produces 1 with probability $1/2$. A call to RANDOM$(3, 7)$ returns any one of $3, 4, 5, 6$, or 7, each with probability $1/5$. Each integer returned by RANDOM is independent of the integers returned on previous calls. You may imagine RANDOM as rolling a $(b - a + 1)$-sided die to obtain its output. (In practice, most programming environments offer a ***pseudorandom-number generator***: a deterministic algorithm returning numbers that "look" statistically random.)

When analyzing the running time of a randomized algorithm, we take the expectation of the running time over the distribution of values returned by the random number generator. We distinguish these algorithms from those in which the input is random by referring to the running time of a randomized algorithm as an ***expected running time***. In general, we discuss the average-case running time when the probability distribution is over the inputs to the algorithm, and we discuss the expected running time when the algorithm itself makes random choices.

Exercises

5.1-1

Show that the assumption that you are always able to determine which candidate is best, in line 4 of procedure HIRE-ASSISTANT, implies that you know a total order on the ranks of the candidates.

★ ***5.1-2***

Describe an implementation of the procedure RANDOM(a, b) that makes calls only to RANDOM$(0, 1)$. What is the expected running time of your procedure, as a function of a and b?

★ ***5.1-3***

You wish to implement a program that outputs 0 with probability $1/2$ and 1 with probability $1/2$. At your disposal is a procedure BIASED-RANDOM that outputs either 0 or 1, but it outputs 1 with some probability p and 0 with probability $1 - p$, where $0 < p < 1$. You do not know what p is. Give an algorithm that uses BIASED-RANDOM as a subroutine, and returns an unbiased answer, returning 0 with probability $1/2$ and 1 with probability $1/2$. What is the expected running time of your algorithm as a function of p?

5.2 Indicator random variables

In order to analyze many algorithms, including the hiring problem, we use indicator random variables. Indicator random variables provide a convenient method for converting between probabilities and expectations. Given a sample space S and an event A, the ***indicator random variable*** $\mathrm{I}\{A\}$ associated with event A is defined as

$$\mathrm{I}\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs}, \\ 0 & \text{if } A \text{ does not occur}. \end{cases} \tag{5.1}$$

As a simple example, let us determine the expected number of heads obtained when flipping a fair coin. The sample space for a single coin flip is $S = \{H, T\}$, with $\Pr\{H\} = \Pr\{T\} = 1/2$. We can then define an indicator random variable X_H, associated with the coin coming up heads, which is the event H. This variable counts the number of heads obtained in this flip, and it is 1 if the coin comes up heads and 0 otherwise. We write

$$\begin{aligned} X_H &= \mathrm{I}\{H\} \\ &= \begin{cases} 1 & \text{if } H \text{ occurs}, \\ 0 & \text{if } T \text{ occurs}. \end{cases} \end{aligned}$$

The expected number of heads obtained in one flip of the coin is simply the expected value of our indicator variable X_H:

$$\begin{aligned} \mathrm{E}[X_H] &= \mathrm{E}[\mathrm{I}\{H\}] \\ &= 1 \cdot \Pr\{H\} + 0 \cdot \Pr\{T\} \\ &= 1 \cdot (1/2) + 0 \cdot (1/2) \\ &= 1/2 . \end{aligned}$$

Thus the expected number of heads obtained by one flip of a fair coin is $1/2$. As the following lemma shows, the expected value of an indicator random variable associated with an event A is equal to the probability that A occurs.

Lemma 5.1
Given a sample space S and an event A in the sample space S, let $X_A = \mathrm{I}\{A\}$. Then $\mathrm{E}[X_A] = \Pr\{A\}$.

Proof By the definition of an indicator random variable from equation (5.1) and the definition of expected value, we have

$$\begin{aligned} \mathrm{E}[X_A] &= \mathrm{E}[\mathrm{I}\{A\}] \\ &= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\overline{A}\} \\ &= \Pr\{A\} , \end{aligned}$$

where $\overline{A}$ denotes $S - A$, the complement of A. ■

Although indicator random variables may seem cumbersome for an application such as counting the expected number of heads on a flip of a single coin, they are useful for analyzing situations that perform repeated random trials. In Appendix C, for example, indicator random variables provide a simple way to determine the expected number of heads in n coin flips. One option is to consider separately the probability of obtaining 0 heads, 1 head, 2 heads, etc. to arrive at the result of equation (C.41) on page 1199. Alternatively, we can employ the simpler method proposed in equation (C.42), which uses indicator random variables implicitly. Making this argument more explicit, let X_i be the indicator random variable associated with the event in which the ith flip comes up heads: $X_i = \mathrm{I}\{\text{the } i\text{th flip results in the event } H\}$. Let X be the random variable denoting the total number of heads in the n coin flips, so that

$$X = \sum_{i=1}^{n} X_i \ .$$

In order to compute the expected number of heads, take the expectation of both sides of the above equation to obtain

$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] \ . \tag{5.2}$$

By Lemma 5.1, the expectation of each of the random variables is $\mathrm{E}[X_i] = 1/2$ for $i = 1, 2, \ldots, n$. Then we can compute the sum of the expectations: $\sum_{i=1}^{n} \mathrm{E}[X_i] = n/2$. But equation (5.2) calls for the expectation of the sum, not the sum of the expectations. How can we resolve this conundrum? Linearity of expectation, equation (C.24) on page 1192, to the rescue: *the expectation of the sum always equals the sum of the expectations*. Linearity of expectation applies even when there is dependence among the random variables. Combining indicator random variables with linearity of expectation gives us a powerful technique to compute expected values when multiple events occur. We now can compute the expected number of heads:

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} \mathrm{E}[X_i] \\
&= \sum_{i=1}^{n} 1/2 \\
&= n/2 \ .
\end{aligned}$$

Thus, compared with the method used in equation (C.41), indicator random variables greatly simplify the calculation. We use indicator random variables throughout this book.

Analysis of the hiring problem using indicator random variables

Returning to the hiring problem, we now wish to compute the expected number of times that you hire a new office assistant. In order to use a probabilistic analysis, let's assume that the candidates arrive in a random order, as discussed in Section 5.1. (We'll see in Section 5.3 how to remove this assumption.) Let X be the random variable whose value equals the number of times you hire a new office assistant. We could then apply the definition of expected value from equation (C.23) on page 1192 to obtain

$$\mathrm{E}[X] = \sum_{x=1}^{n} x \Pr\{X = x\} ,$$

but this calculation would be cumbersome. Instead, let's simplify the calculation by using indicator random variables.

To use indicator random variables, instead of computing $\mathrm{E}[X]$ by defining just one variable denoting the number of times you hire a new office assistant, think of the process of hiring as repeated random trials and define n variables indicating whether each particular candidate is hired. In particular, let X_i be the indicator random variable associated with the event in which the ith candidate is hired. Thus,

$$\begin{aligned} X_i &= \mathrm{I}\{\text{candidate } i \text{ is hired}\} \\ &= \begin{cases} 1 & \text{if candidate } i \text{ is hired} , \\ 0 & \text{if candidate } i \text{ is not hired} , \end{cases} \end{aligned}$$

and

$$X = X_1 + X_2 + \cdots + X_n . \tag{5.3}$$

Lemma 5.1 gives

$$\mathrm{E}[X_i] = \Pr\{\text{candidate } i \text{ is hired}\} ,$$

and we must therefore compute the probability that lines 5–6 of HIRE-ASSISTANT are executed.

Candidate i is hired, in line 6, exactly when candidate i is better than each of candidates 1 through $i - 1$. Because we have assumed that the candidates arrive in a random order, the first i candidates have appeared in a random order. Any one of these first i candidates is equally likely to be the best qualified so far. Candidate i has a probability of $1/i$ of being better qualified than candidates 1 through $i - 1$ and thus a probability of $1/i$ of being hired. By Lemma 5.1, we conclude that

$$\mathrm{E}[X_i] = 1/i \ . \tag{5.4}$$

Now we can compute $\mathrm{E}[X]$:

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] && \text{(by equation (5.3))} && (5.5)\\
&= \sum_{i=1}^{n} \mathrm{E}[X_i] && \text{(by equation (C.24), linearity of expectation)} \\
&= \sum_{i=1}^{n} \frac{1}{i} && \text{(by equation (5.4))} \\
&= \ln n + O(1) && \text{(by equation (A.9), the harmonic series) .} && (5.6)
\end{aligned}$$

Even though you interview n people, you actually hire only approximately $\ln n$ of them, on average. We summarize this result in the following lemma.

Lemma 5.2
Assuming that the candidates are presented in a random order, algorithm HIRE-ASSISTANT has an average-case total hiring cost of $O(c_h \ln n)$.

Proof The bound follows immediately from our definition of the hiring cost and equation (5.6), which shows that the expected number of hires is approximately $\ln n$. ■

The average-case hiring cost is a significant improvement over the worst-case hiring cost of $O(c_h n)$.

Exercises

5.2-1
In HIRE-ASSISTANT, assuming that the candidates are presented in a random order, what is the probability that you hire exactly one time? What is the probability that you hire exactly n times?

5.2-2
In HIRE-ASSISTANT, assuming that the candidates are presented in a random order, what is the probability that you hire exactly twice?

5.2-3
Use indicator random variables to compute the expected value of the sum of n dice.

5.2-4

This exercise asks you to (partly) verify that linearity of expectation holds even if the random variables are not independent. Consider two 6-sided dice that are rolled independently. What is the expected value of the sum? Now consider the case where the first die is rolled normally and then the second die is set equal to the value shown on the first die. What is the expected value of the sum? Now consider the case where the first die is rolled normally and the second die is set equal to 7 minus the value of the first die. What is the expected value of the sum?

5.2-5

Use indicator random variables to solve the following problem, which is known as the ***hat-check problem***. Each of n customers gives a hat to a hat-check person at a restaurant. The hat-check person gives the hats back to the customers in a random order. What is the expected number of customers who get back their own hat?

5.2-6

Let $A[1:n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an ***inversion*** of A. (See Problem 2-4 on page 47 for more on inversions.) Suppose that the elements of A form a uniform random permutation of $\langle 1, 2, \ldots, n \rangle$. Use indicator random variables to compute the expected number of inversions.

5.3 Randomized algorithms

In the previous section, we showed how knowing a distribution on the inputs can help us to analyze the average-case behavior of an algorithm. What if you do not know the distribution? Then you cannot perform an average-case analysis. As mentioned in Section 5.1, however, you might be able to use a randomized algorithm.

For a problem such as the hiring problem, in which it is helpful to assume that all permutations of the input are equally likely, a probabilistic analysis can guide us when developing a randomized algorithm. Instead of *assuming* a distribution of inputs, we *impose* a distribution. In particular, before running the algorithm, let's randomly permute the candidates in order to enforce the property that every permutation is equally likely. Although we have modified the algorithm, we still expect to hire a new office assistant approximately $\ln n$ times. But now we expect this to be the case for *any* input, rather than for inputs drawn from a particular distribution.

Let us further explore the distinction between probabilistic analysis and randomized algorithms. In Section 5.2, we claimed that, assuming that the candidates

arrive in a random order, the expected number of times you hire a new office assistant is about $\ln n$. This algorithm is deterministic: for any particular input, the number of times a new office assistant is hired is always the same. Furthermore, the number of times you hire a new office assistant differs for different inputs, and it depends on the ranks of the various candidates. Since this number depends only on the ranks of the candidates, to represent a particular input, we can just list, in order, the ranks $\langle rank(1), rank(2), \ldots, rank(n)\rangle$ of the candidates. Given the rank list $A_1 = \langle 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\rangle$, a new office assistant is always hired 10 times, since each successive candidate is better than the previous one, and lines 5–6 of HIRE-ASSISTANT are executed in each iteration. Given the list of ranks $A_2 = \langle 10, 9, 8, 7, 6, 5, 4, 3, 2, 1\rangle$, a new office assistant is hired only once, in the first iteration. Given a list of ranks $A_3 = \langle 5, 2, 1, 8, 4, 7, 10, 9, 3, 6\rangle$, a new office assistant is hired three times, upon interviewing the candidates with ranks 5, 8, and 10. Recalling that the cost of our algorithm depends on how many times you hire a new office assistant, we see that there are expensive inputs such as A_1, inexpensive inputs such as A_2, and moderately expensive inputs such as A_3.

Consider, on the other hand, the randomized algorithm that first permutes the list of candidates and then determines the best candidate. In this case, we randomize in the algorithm, not in the input distribution. Given a particular input, say A_3 above, we cannot say how many times the maximum is updated, because this quantity differs with each run of the algorithm. The first time you run the algorithm on A_3, it might produce the permutation A_1 and perform 10 updates. But the second time you run the algorithm, it might produce the permutation A_2 and perform only one update. The third time you run the algorithm, it might perform some other number of updates. Each time you run the algorithm, its execution depends on the random choices made and is likely to differ from the previous execution of the algorithm. For this algorithm and many other randomized algorithms, *no particular input elicits its worst-case behavior*. Even your worst enemy cannot produce a bad input array, since the random permutation makes the input order irrelevant. The randomized algorithm performs badly only if the random-number generator produces an "unlucky" permutation.

For the hiring problem, the only change needed in the code is to randomly permute the array, as done in the RANDOMIZED-HIRE-ASSISTANT procedure. This simple change creates a randomized algorithm whose performance matches that obtained by assuming that the candidates were presented in a random order.

```
RANDOMIZED-HIRE-ASSISTANT(n)
1  randomly permute the list of candidates
2  HIRE-ASSISTANT(n)
```

Lemma 5.3
The expected hiring cost of the procedure RANDOMIZED-HIRE-ASSISTANT is $O(c_h \ln n)$.

Proof Permuting the input array achieves a situation identical to that of the probabilistic analysis of HIRE-ASSISTANT in Section 5.2. ■

By carefully comparing Lemmas 5.2 and 5.3, you can see the difference between probabilistic analysis and randomized algorithms. Lemma 5.2 makes an assumption about the input. Lemma 5.3 makes no such assumption, although randomizing the input takes some additional time. To remain consistent with our terminology, we couched Lemma 5.2 in terms of the average-case hiring cost and Lemma 5.3 in terms of the expected hiring cost. In the remainder of this section, we discuss some issues involved in randomly permuting inputs.

Randomly permuting arrays

Many randomized algorithms randomize the input by permuting a given input array. We'll see elsewhere in this book other ways to randomize an algorithm, but now, let's see how we can randomly permute an array of n elements. The goal is to produce a ***uniform random permutation***, that is, a permutation that is as likely as any other permutation. Since there are $n!$ possible permutations, we want the probability that any particular permutation is produced to be $1/n!$.

You might think that to prove that a permutation is a uniform random permutation, it suffices to show that, for each element $A[i]$, the probability that the element winds up in position j is $1/n$. Exercise 5.3-4 shows that this weaker condition is, in fact, insufficient.

Our method to generate a random permutation permutes the array ***in place***: at most a constant number of elements of the input array are ever stored outside the array. The procedure RANDOMLY-PERMUTE permutes an array $A[1:n]$ in place in $\Theta(n)$ time. In its ith iteration, it chooses the element $A[i]$ randomly from among elements $A[i]$ through $A[n]$. After the ith iteration, $A[i]$ is never altered.

```
RANDOMLY-PERMUTE(A, n)
1  for i = 1 to n
2      swap A[i] with A[RANDOM(i, n)]
```

We use a loop invariant to show that procedure RANDOMLY-PERMUTE produces a uniform random permutation. A ***k-permutation*** on a set of n elements is a se-

quence containing k of the n elements, with no repetitions. (See page 1180 in Appendix C.) There are $n!/(n-k)!$ such possible k-permutations.

Lemma 5.4
Procedure RANDOMLY-PERMUTE computes a uniform random permutation.

Proof We use the following loop invariant:

> Just prior to the ith iteration of the **for** loop of lines 1–2, for each possible $(i-1)$-permutation of the n elements, the subarray $A[1:i-1]$ contains this $(i-1)$-permutation with probability $(n-i+1)!/n!$.

We need to show that this invariant is true prior to the first loop iteration, that each iteration of the loop maintains the invariant, that the loop terminates, and that the invariant provides a useful property to show correctness when the loop terminates.

Initialization: Consider the situation just before the first loop iteration, so that $i = 1$. The loop invariant says that for each possible 0-permutation, the subarray $A[1:0]$ contains this 0-permutation with probability $(n-i+1)!/n! = n!/n! = 1$. The subarray $A[1:0]$ is an empty subarray, and a 0-permutation has no elements. Thus, $A[1:0]$ contains any 0-permutation with probability 1, and the loop invariant holds prior to the first iteration.

Maintenance: By the loop invariant, we assume that just before the ith iteration, each possible $(i-1)$-permutation appears in the subarray $A[1:i-1]$ with probability $(n-i+1)!/n!$. We shall show that after the ith iteration, each possible i-permutation appears in the subarray $A[1:i]$ with probability $(n-i)!/n!$. Incrementing i for the next iteration then maintains the loop invariant.

Let us examine the ith iteration. Consider a particular i-permutation, and denote the elements in it by $\langle x_1, x_2, \ldots, x_i \rangle$. This permutation consists of an $(i-1)$-permutation $\langle x_1, \ldots, x_{i-1} \rangle$ followed by the value x_i that the algorithm places in $A[i]$. Let E_1 denote the event in which the first $i-1$ iterations have created the particular $(i-1)$-permutation $\langle x_1, \ldots, x_{i-1} \rangle$ in $A[1:i-1]$. By the loop invariant, $\Pr\{E_1\} = (n-i+1)!/n!$. Let E_2 be the event that the ith iteration puts x_i in position $A[i]$. The i-permutation $\langle x_1, \ldots, x_i \rangle$ appears in $A[1:i]$ precisely when both E_1 and E_2 occur, and so we wish to compute $\Pr\{E_2 \cap E_1\}$. Using equation (C.16) on page 1187, we have

$$\Pr\{E_2 \cap E_1\} = \Pr\{E_2 \mid E_1\} \Pr\{E_1\} \ .$$

The probability $\Pr\{E_2 \mid E_1\}$ equals $1/(n-i+1)$ because in line 2 the algorithm chooses x_i randomly from the $n-i+1$ values in positions $A[i:n]$. Thus, we have

$$\begin{aligned}\Pr\{E_2 \cap E_1\} &= \Pr\{E_2 \mid E_1\}\Pr\{E_1\} \\ &= \frac{1}{n-i+1} \cdot \frac{(n-i+1)!}{n!} \\ &= \frac{(n-i)!}{n!} .\end{aligned}$$

Termination: The loop terminates, since it is a **for** loop iterating n times. At termination, $i = n+1$, and we have that the subarray $A[1:n]$ is a given n-permutation with probability $(n-(n+1)+1)!/n! = 0!/n! = 1/n!$.

Thus, RANDOMLY-PERMUTE produces a uniform random permutation. ■

A randomized algorithm is often the simplest and most efficient way to solve a problem.

Exercises

5.3-1

Professor Marceau objects to the loop invariant used in the proof of Lemma 5.4. He questions whether it holds prior to the first iteration. He reasons that we could just as easily declare that an empty subarray contains no 0-permutations. Therefore, the probability that an empty subarray contains a 0-permutation should be 0, thus invalidating the loop invariant prior to the first iteration. Rewrite the procedure RANDOMLY-PERMUTE so that its associated loop invariant applies to a nonempty subarray prior to the first iteration, and modify the proof of Lemma 5.4 for your procedure.

5.3-2

Professor Kelp decides to write a procedure that produces at random any permutation except the ***identity permutation***, in which every element ends up where it started. He proposes the procedure PERMUTE-WITHOUT-IDENTITY. Does this procedure do what Professor Kelp intends?

```
PERMUTE-WITHOUT-IDENTITY(A, n)
1  for i = 1 to n − 1
2      swap A[i] with A[RANDOM(i + 1, n)]
```

5.3-3

Consider the PERMUTE-WITH-ALL procedure on the facing page, which instead of swapping element $A[i]$ with a random element from the subarray $A[i:n]$, swaps it with a random element from anywhere in the array. Does PERMUTE-WITH-ALL produce a uniform random permutation? Why or why not?

```
PERMUTE-WITH-ALL(A, n)
1  for i = 1 to n
2      swap A[i] with A[RANDOM(1, n)]
```

5.3-4

Professor Knievel suggests the procedure PERMUTE-BY-CYCLE to generate a uniform random permutation. Show that each element $A[i]$ has a $1/n$ probability of winding up in any particular position in B. Then show that Professor Knievel is mistaken by showing that the resulting permutation is not uniformly random.

```
PERMUTE-BY-CYCLE(A, n)
1  let B[1 : n] be a new array
2  offset = RANDOM(1, n)
3  for i = 1 to n
4      dest = i + offset
5      if dest > n
6          dest = dest − n
7      B[dest] = A[i]
8  return B
```

5.3-5

Professor Gallup wants to create a ***random sample*** of the set $\{1, 2, 3, \ldots, n\}$, that is, an m-element subset S, where $0 \le m \le n$, such that each m-subset is equally likely to be created. One way is to set $A[i] = i$, for $i = 1, 2, 3, \ldots, n$, call RANDOMLY-PERMUTE(A), and then take just the first m array elements. This method makes n calls to the RANDOM procedure. In Professor Gallup's application, n is much larger than m, and so the professor wants to create a random sample with fewer calls to RANDOM.

```
RANDOM-SAMPLE(m, n)
1  S = ∅
2  for k = n − m + 1 to n          // iterates m times
3      i = RANDOM(1, k)
4      if i ∈ S
5          S = S ∪ {k}
6      else S = S ∪ {i}
7  return S
```

Show that the procedure RANDOM-SAMPLE on the previous page returns a random m-subset S of $\{1, 2, 3, \ldots, n\}$, in which each m-subset is equally likely, while making only m calls to RANDOM.

⋆ 5.4 Probabilistic analysis and further uses of indicator random variables

This advanced section further illustrates probabilistic analysis by way of four examples. The first determines the probability that in a room of k people, two of them share the same birthday. The second example examines what happens when randomly tossing balls into bins. The third investigates "streaks" of consecutive heads when flipping coins. The final example analyzes a variant of the hiring problem in which you have to make decisions without actually interviewing all the candidates.

5.4.1 The birthday paradox

Our first example is the ***birthday paradox***. How many people must there be in a room before there is a 50% chance that two of them were born on the same day of the year? The answer is surprisingly few. The paradox is that it is in fact far fewer than the number of days in a year, or even half the number of days in a year, as we shall see.

To answer this question, we index the people in the room with the integers $1, 2, \ldots, k$, where k is the number of people in the room. We ignore the issue of leap years and assume that all years have $n = 365$ days. For $i = 1, 2, \ldots, k$, let b_i be the day of the year on which person i's birthday falls, where $1 \le b_i \le n$. We also assume that birthdays are uniformly distributed across the n days of the year, so that $\Pr\{b_i = r\} = 1/n$ for $i = 1, 2, \ldots, k$ and $r = 1, 2, \ldots, n$.

The probability that two given people, say i and j, have matching birthdays depends on whether the random selection of birthdays is independent. We assume from now on that birthdays are independent, so that the probability that i's birthday and j's birthday both fall on day r is

$$\begin{aligned}\Pr\{b_i = r \text{ and } b_j = r\} &= \Pr\{b_i = r\}\Pr\{b_j = r\} \\ &= \frac{1}{n^2}\,.\end{aligned}$$

Thus, the probability that they both fall on the same day is

$$\Pr\{b_i = b_j\} = \sum_{r=1}^{n} \Pr\{b_i = r \text{ and } b_j = r\}$$

$$= \sum_{r=1}^{n} \frac{1}{n^2}$$
$$= \frac{1}{n} . \tag{5.7}$$

More intuitively, once b_i is chosen, the probability that b_j is chosen to be the same day is $1/n$. As long as the birthdays are independent, the probability that i and j have the same birthday is the same as the probability that the birthday of one of them falls on a given day.

We can analyze the probability of at least 2 out of k people having matching birthdays by looking at the complementary event. The probability that at least two of the birthdays match is 1 minus the probability that all the birthdays are different. The event B_k that k people have distinct birthdays is

$$B_k = \bigcap_{i=1}^{k} A_i ,$$

where A_i is the event that person i's birthday is different from person j's for all $j < i$. Since we can write $B_k = A_k \cap B_{k-1}$, we obtain from equation (C.18) on page 1189 the recurrence

$$\Pr\{B_k\} = \Pr\{B_{k-1}\} \Pr\{A_k \mid B_{k-1}\} , \tag{5.8}$$

where we take $\Pr\{B_1\} = \Pr\{A_1\} = 1$ as an initial condition. In other words, the probability that $b_1, b_2, \ldots, b_k$ are distinct birthdays equals the probability that $b_1, b_2, \ldots, b_{k-1}$ are distinct birthdays multiplied by the probability that $b_k \neq b_i$ for $i = 1, 2, \ldots, k-1$, given that $b_1, b_2, \ldots, b_{k-1}$ are distinct.

If $b_1, b_2, \ldots, b_{k-1}$ are distinct, the conditional probability that $b_k \neq b_i$ for $i = 1, 2, \ldots, k-1$ is $\Pr\{A_k \mid B_{k-1}\} = (n-k+1)/n$, since out of the n days, $n-(k-1)$ days are not taken. We iteratively apply the recurrence (5.8) to obtain

$$\begin{aligned}
\Pr\{B_k\} &= \Pr\{B_{k-1}\} \Pr\{A_k \mid B_{k-1}\} \\
&= \Pr\{B_{k-2}\} \Pr\{A_{k-1} \mid B_{k-2}\} \Pr\{A_k \mid B_{k-1}\} \\
&\vdots \\
&= \Pr\{B_1\} \Pr\{A_2 \mid B_1\} \Pr\{A_3 \mid B_2\} \cdots \Pr\{A_k \mid B_{k-1}\} \\
&= 1 \cdot \left(\frac{n-1}{n}\right)\left(\frac{n-2}{n}\right) \cdots \left(\frac{n-k+1}{n}\right) \\
&= 1 \cdot \left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right) \cdots \left(1 - \frac{k-1}{n}\right) .
\end{aligned}$$

Inequality (3.14) on page 66, $1 + x \leq e^x$, gives us

$$\begin{aligned}
\Pr\{B_k\} &\le e^{-1/n} e^{-2/n} \cdots e^{-(k-1)/n} \\
&= e^{-\sum_{i=1}^{k-1} i/n} \\
&= e^{-k(k-1)/2n} \\
&\le \frac{1}{2}
\end{aligned}$$

when $-k(k-1)/2n \le \ln(1/2)$. The probability that all k birthdays are distinct is at most $1/2$ when $k(k-1) \ge 2n \ln 2$ or, solving the quadratic equation, when $k \ge (1 + \sqrt{1 + (8 \ln 2)n})/2$. For $n = 365$, we must have $k \ge 23$. Thus, if at least 23 people are in a room, the probability is at least $1/2$ that at least two people have the same birthday. Since a year on Mars is 669 Martian days long, it takes 31 Martians to get the same effect.

An analysis using indicator random variables

Indicator random variables afford a simpler but approximate analysis of the birthday paradox. For each pair (i, j) of the k people in the room, define the indicator random variable X_{ij}, for $1 \le i < j \le k$, by

$$\begin{aligned}
X_{ij} &= \mathrm{I}\{\text{person } i \text{ and person } j \text{ have the same birthday}\} \\
&= \begin{cases} 1 & \text{if person } i \text{ and person } j \text{ have the same birthday}, \\ 0 & \text{otherwise}. \end{cases}
\end{aligned}$$

By equation (5.7), the probability that two people have matching birthdays is $1/n$, and thus by Lemma 5.1 on page 130, we have

$$\begin{aligned}
\mathrm{E}[X_{ij}] &= \Pr\{\text{person } i \text{ and person } j \text{ have the same birthday}\} \\
&= 1/n .
\end{aligned}$$

Letting X be the random variable that counts the number of pairs of individuals having the same birthday, we have

$$X = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} X_{ij} .$$

Taking expectations of both sides and applying linearity of expectation, we obtain

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} X_{ij}\right] \\
&= \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \mathrm{E}[X_{ij}]
\end{aligned}$$

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

When $k(k-1) \geq 2n$, therefore, the expected number of pairs of people with the same birthday is at least 1. Thus, if we have at least $\sqrt{2n}+1$ individuals in a room, we can expect at least two to have the same birthday. For $n = 365$, if $k = 28$, the expected number of pairs with the same birthday is $(28 \cdot 27)/(2 \cdot 365) \approx 1.0356$. Thus, with at least 28 people, we expect to find at least one matching pair of birthdays. On Mars, with 669 days per year, we need at least 38 Martians.

The first analysis, which used only probabilities, determined the number of people required for the probability to exceed $1/2$ that a matching pair of birthdays exists, and the second analysis, which used indicator random variables, determined the number such that the expected number of matching birthdays is 1. Although the exact numbers of people differ for the two situations, they are the same asymptotically: $\Theta(\sqrt{n})$.

5.4.2 Balls and bins

Consider a process in which you randomly toss identical balls into b bins, numbered $1, 2, \ldots, b$. The tosses are independent, and on each toss the ball is equally likely to end up in any bin. The probability that a tossed ball lands in any given bin is $1/b$. If we view the ball-tossing process as a sequence of Bernoulli trials (see Appendix C.4), where success means that the ball falls in the given bin, then each trial has a probability $1/b$ of success. This model is particularly useful for analyzing hashing (see Chapter 11), and we can answer a variety of interesting questions about the ball-tossing process. (Problem C-2 asks additional questions about balls and bins.)

- *How many balls fall in a given bin?* The number of balls that fall in a given bin follows the binomial distribution $b(k; n, 1/b)$. If you toss n balls, equation (C.41) on page 1199 tells us that the expected number of balls that fall in the given bin is n/b.
- *How many balls must you toss, on the average, until a given bin contains a ball?* The number of tosses until the given bin receives a ball follows the geometric distribution with probability $1/b$ and, by equation (C.36) on page 1197, the expected number of tosses until success is $1/(1/b) = b$.
- *How many balls must you toss until every bin contains at least one ball?* Let us call a toss in which a ball falls into an empty bin a "hit." We want to know the expected number n of tosses required to get b hits.

Using the hits, we can partition the n tosses into stages. The ith stage consists of the tosses after the $(i-1)$st hit up to and including the ith hit. The first stage consists of the first toss, since you are guaranteed to have a hit when all bins are empty. For each toss during the ith stage, $i-1$ bins contain balls and $b-i+1$ bins are empty. Thus, for each toss in the ith stage, the probability of obtaining a hit is $(b-i+1)/b$.

Let n_i denote the number of tosses in the ith stage. The number of tosses required to get b hits is $n = \sum_{i=1}^{b} n_i$. Each random variable n_i has a geometric distribution with probability of success $(b-i+1)/b$ and thus, by equation (C.36), we have

$$\mathrm{E}[n_i] = \frac{b}{b-i+1} .$$

By linearity of expectation, we have

$$\begin{aligned}
\mathrm{E}[n] &= \mathrm{E}\left[\sum_{i=1}^{b} n_i\right] \\
&= \sum_{i=1}^{b} \mathrm{E}[n_i] \\
&= \sum_{i=1}^{b} \frac{b}{b-i+1} \\
&= b\sum_{i=1}^{b} \frac{1}{i} && \text{(by equation (A.14) on page 1144)} \\
&= b(\ln b + O(1)) && \text{(by equation (A.9) on page 1142) .}
\end{aligned}$$

It therefore takes approximately $b \ln b$ tosses before we can expect that every bin has a ball. This problem is also known as the ***coupon collector's problem***, which says that if you are trying to collect each of b different coupons, then you should expect to acquire approximately $b \ln b$ randomly obtained coupons in order to succeed.

5.4.3 Streaks

Suppose that you flip a fair coin n times. What is the longest streak of consecutive heads that you expect to see? We'll prove upper and lower bounds separately to show that the answer is $\Theta(\lg n)$.

We first prove that the expected length of the longest streak of heads is $O(\lg n)$. The probability that each coin flip is a head is $1/2$. Let A_{ik} be the event that a streak of heads of length at least k begins with the ith coin flip or, more precisely, the event that the k consecutive coin flips $i, i+1, \ldots, i+k-1$ yield only heads, where $1 \le k \le n$ and $1 \le i \le n-k+1$. Since coin flips are mutually independent, for any given event A_{ik}, the probability that all k flips are heads is

$$\Pr\{A_{ik}\} = \frac{1}{2^k} . \tag{5.9}$$

For $k = 2\lceil \lg n \rceil$,

$$\begin{aligned}
\Pr\{A_{i,2\lceil \lg n \rceil}\} &= \frac{1}{2^{2\lceil \lg n \rceil}} \\
&\le \frac{1}{2^{2 \lg n}} \\
&= \frac{1}{n^2} ,
\end{aligned}$$

and thus the probability that a streak of heads of length at least $2\lceil \lg n \rceil$ begins in position i is quite small. There are at most $n - 2\lceil \lg n \rceil + 1$ positions where such a streak can begin. The probability that a streak of heads of length at least $2\lceil \lg n \rceil$ begins anywhere is therefore

$$\begin{aligned}
&\Pr\left\{ \bigcup_{i=1}^{n-2\lceil \lg n \rceil+1} A_{i,2\lceil \lg n \rceil} \right\} \\
&\quad \le \sum_{i=1}^{n-2\lceil \lg n \rceil+1} \Pr\{A_{i,2\lceil \lg n \rceil}\} \quad \text{(by Boole's inequality (C.21) on page 1190)} \\
&\quad \le \sum_{i=1}^{n-2\lceil \lg n \rceil+1} \frac{1}{n^2} \\
&\quad < \sum_{i=1}^{n} \frac{1}{n^2} \\
&\quad = \frac{1}{n} .
\end{aligned} \tag{5.10}$$

We can use inequality (5.10) to bound the length of the longest streak. For $j = 0, 1, 2, \ldots, n$, let L_j be the event that the longest streak of heads has length exactly j, and let L be the length of the longest streak. By the definition of expected value, we have

$$\mathrm{E}[L] = \sum_{j=0}^{n} j \Pr\{L_j\} . \tag{5.11}$$

We could try to evaluate this sum using upper bounds on each $\Pr\{L_j\}$ similar to those computed in inequality (5.10). Unfortunately, this method yields weak bounds. We can use some intuition gained by the above analysis to obtain a good bound, however. For no individual term in the summation in equation (5.11) are both the factors j and $\Pr\{L_j\}$ large. Why? When $j \geq 2\lceil \lg n \rceil$, then $\Pr\{L_j\}$ is very small, and when $j < 2\lceil \lg n \rceil$, then j is fairly small. More precisely, since the events L_j for $j = 0, 1, \ldots, n$ are disjoint, the probability that a streak of heads of length at least $2\lceil \lg n \rceil$ begins anywhere is $\sum_{j=2\lceil \lg n \rceil}^{n} \Pr\{L_j\}$. Inequality (5.10) tells us that the probability that a streak of heads of length at least $2\lceil \lg n \rceil$ begins anywhere is less than $1/n$, which means that $\sum_{j=2\lceil \lg n \rceil}^{n} \Pr\{L_j\} < 1/n$. Also, noting that $\sum_{j=0}^{n} \Pr\{L_j\} = 1$, we have that $\sum_{j=0}^{2\lceil \lg n \rceil - 1} \Pr\{L_j\} \leq 1$. Thus, we obtain

$$\begin{aligned}
\mathrm{E}[L] &= \sum_{j=0}^{n} j \Pr\{L_j\} \\
&= \sum_{j=0}^{2\lceil \lg n \rceil - 1} j \Pr\{L_j\} + \sum_{j=2\lceil \lg n \rceil}^{n} j \Pr\{L_j\} \\
&< \sum_{j=0}^{2\lceil \lg n \rceil - 1} (2\lceil \lg n \rceil) \Pr\{L_j\} + \sum_{j=2\lceil \lg n \rceil}^{n} n \Pr\{L_j\} \\
&= 2\lceil \lg n \rceil \sum_{j=0}^{2\lceil \lg n \rceil - 1} \Pr\{L_j\} + n \sum_{j=2\lceil \lg n \rceil}^{n} \Pr\{L_j\} \\
&< 2\lceil \lg n \rceil \cdot 1 + n \cdot \frac{1}{n} \\
&= O(\lg n) \, .
\end{aligned}$$

The probability that a streak of heads exceeds $r\lceil \lg n \rceil$ flips diminishes quickly with r. Let's get a rough bound on the probability that a streak of at least $r\lceil \lg n \rceil$ heads occurs, for $r \geq 1$. The probability that a streak of at least $r\lceil \lg n \rceil$ heads starts in position i is

$$\begin{aligned}
\Pr\{A_{i, r\lceil \lg n \rceil}\} &= \frac{1}{2^{r\lceil \lg n \rceil}} \\
&\leq \frac{1}{n^r} \, .
\end{aligned}$$

A streak of at least $r\lceil \lg n \rceil$ heads cannot start in the last $n - r\lceil \lg n \rceil + 1$ flips, but let's overestimate the probability of such a streak by allowing it to start anywhere within the n coin flips. Then the probability that a streak of at least $r\lceil \lg n \rceil$ heads

occurs is at most

$$\begin{aligned}
\Pr\left\{\bigcup_{i=1}^{n} A_{i,r\lceil \lg n\rceil}\right\} &\le \sum_{i=1}^{n} \Pr\{A_{i,r\lceil \lg n\rceil}\} \qquad \text{(by Boole's inequality (C.21))} \\
&\le \sum_{i=1}^{n} \frac{1}{n^r} \\
&= \frac{1}{n^{r-1}} .
\end{aligned}$$

Equivalently, the probability is at least $1-1/n^{r-1}$ that the longest streak has length less than $r\lceil \lg n\rceil$.

As an example, during $n = 1000$ coin flips, the probability of encountering a streak of at least $2\lceil \lg n\rceil = 20$ heads is at most $1/n = 1/1000$. The chance of a streak of at least $3\lceil \lg n\rceil = 30$ heads is at most $1/n^2 = 1/1{,}000{,}000$.

Let's now prove a complementary lower bound: the expected length of the longest streak of heads in n coin flips is $\Omega(\lg n)$. To prove this bound, we look for streaks of length s by partitioning the n flips into approximately n/s groups of s flips each. If we choose $s = \lfloor (\lg n)/2\rfloor$, we'll see that it is likely that at least one of these groups comes up all heads, which means that it's likely that the longest streak has length at least $s = \Omega(\lg n)$. We'll then show that the longest streak has expected length $\Omega(\lg n)$.

Let's partition the n coin flips into at least $\lfloor n/\lfloor (\lg n)/2\rfloor\rfloor$ groups of $\lfloor (\lg n)/2\rfloor$ consecutive flips and bound the probability that no group comes up all heads. By equation (5.9), the probability that the group starting in position i comes up all heads is

$$\begin{aligned}
\Pr\{A_{i,\lfloor (\lg n)/2\rfloor}\} &= \frac{1}{2^{\lfloor (\lg n)/2\rfloor}} \\
&\ge \frac{1}{\sqrt{n}} .
\end{aligned}$$

The probability that a streak of heads of length at least $\lfloor (\lg n)/2\rfloor$ does not begin in position i is therefore at most $1-1/\sqrt{n}$. Since the $\lfloor n/\lfloor (\lg n)/2\rfloor\rfloor$ groups are formed from mutually exclusive, independent coin flips, the probability that every one of these groups *fails* to be a streak of length $\lfloor (\lg n)/2\rfloor$ is at most

$$\begin{aligned}
\left(1-1/\sqrt{n}\right)^{\lfloor n/\lfloor (\lg n)/2\rfloor\rfloor} &\le \left(1-1/\sqrt{n}\right)^{n/\lfloor (\lg n)/2\rfloor - 1} \\
&\le \left(1-1/\sqrt{n}\right)^{2n/\lg n - 1} \\
&\le e^{-(2n/\lg n - 1)/\sqrt{n}} \\
&= O(e^{-\ln n}) \\
&= O(1/n) . \qquad (5.12)
\end{aligned}$$

For this argument, we used inequality (3.14), $1 + x \le e^x$, on page 66 and the fact, which you may verify, that $(2n/\lg n - 1)/\sqrt{n} \ge \ln n$ for sufficiently large n.

We want to bound the probability that the longest streak equals or exceeds $\lfloor (\lg n)/2 \rfloor$. To do so, let L be the event that the longest streak of heads equals or exceeds $s = \lfloor (\lg n)/2 \rfloor$. Let $\overline{L}$ be the complementary event, that the longest streak of heads is strictly less than s, so that $\Pr\{L\} + \Pr\{\overline{L}\} = 1$. Let F be the event that every group of s flips fails to be a streak of s heads. By inequality (5.12), we have $\Pr\{F\} = O(1/n)$. If the longest streak of heads is less than s, then certainly every group of s flips fails to be a streak of s heads, which means that event $\overline{L}$ implies event F. Of course, event F could occur even if event $\overline{L}$ does not (for example, if a streak of s or more heads crosses over the boundary between two groups), and so we have $\Pr\{\overline{L}\} \le \Pr\{F\} = O(1/n)$. Since $\Pr\{L\} + \Pr\{\overline{L}\} = 1$, we have that

$$\begin{aligned}
\Pr\{L\} &= 1 - \Pr\{\overline{L}\} \\
&\ge 1 - \Pr\{F\} \\
&= 1 - O(1/n) \, .
\end{aligned}$$

That is, the probability that the longest streak equals or exceeds $\lfloor (\lg n)/2 \rfloor$ is

$$\sum_{j=\lfloor (\lg n)/2 \rfloor}^{n} \Pr\{L_j\} \ge 1 - O(1/n) \, . \tag{5.13}$$

We can now calculate a lower bound on the expected length of the longest streak, beginning with equation (5.11) and proceeding in a manner similar to our analysis of the upper bound:

$$\begin{aligned}
\mathrm{E}[L] &= \sum_{j=0}^{n} j \Pr\{L_j\} \\
&= \sum_{j=0}^{\lfloor (\lg n)/2 \rfloor - 1} j \Pr\{L_j\} + \sum_{j=\lfloor (\lg n)/2 \rfloor}^{n} j \Pr\{L_j\} \\
&\ge \sum_{j=0}^{\lfloor (\lg n)/2 \rfloor - 1} 0 \cdot \Pr\{L_j\} + \sum_{j=\lfloor (\lg n)/2 \rfloor}^{n} \lfloor (\lg n)/2 \rfloor \Pr\{L_j\} \\
&= 0 \cdot \sum_{j=0}^{\lfloor (\lg n)/2 \rfloor - 1} \Pr\{L_j\} + \lfloor (\lg n)/2 \rfloor \sum_{j=\lfloor (\lg n)/2 \rfloor}^{n} \Pr\{L_j\} \\
&\ge 0 + \lfloor (\lg n)/2 \rfloor \, (1 - O(1/n)) \qquad \text{(by inequality (5.13))} \\
&= \Omega(\lg n) \, .
\end{aligned}$$

As with the birthday paradox, we can obtain a simpler, but approximate, analysis using indicator random variables. Instead of determining the expected length of the longest streak, we'll find the expected number of streaks with at least a given length. Let $X_{ik} = \mathrm{I}\{A_{ik}\}$ be the indicator random variable associated with a streak of heads of length at least k beginning with the ith coin flip. To count the total number of such streaks, define

$$X_k = \sum_{i=1}^{n-k+1} X_{ik} \; .$$

Taking expectations and using linearity of expectation, we have

$$\begin{aligned}
\mathrm{E}[X_k] &= \mathrm{E}\left[\sum_{i=1}^{n-k+1} X_{ik}\right] \\
&= \sum_{i=1}^{n-k+1} \mathrm{E}[X_{ik}] \\
&= \sum_{i=1}^{n-k+1} \Pr\{A_{ik}\} \\
&= \sum_{i=1}^{n-k+1} \frac{1}{2^k} \\
&= \frac{n-k+1}{2^k} \; .
\end{aligned}$$

By plugging in various values for k, we can calculate the expected number of streaks of length at least k. If this expected number is large (much greater than 1), then we expect many streaks of length k to occur, and the probability that one occurs is high. If this expected number is small (much less than 1), then we expect to see few streaks of length k, and the probability that one occurs is low. If $k = c \lg n$, for some positive constant c, we obtain

$$\begin{aligned}
\mathrm{E}[X_{c \lg n}] &= \frac{n - c \lg n + 1}{2^{c \lg n}} \\
&= \frac{n - c \lg n + 1}{n^c} \\
&= \frac{1}{n^{c-1}} - \frac{(c \lg n - 1)/n}{n^{c-1}} \\
&= \Theta(1/n^{c-1}) \; .
\end{aligned}$$

If c is large, the expected number of streaks of length $c \lg n$ is small, and we conclude that they are unlikely to occur. On the other hand, if $c = 1/2$, then we

obtain $\mathrm{E}[X_{(1/2)\lg n}] = \Theta(1/n^{1/2-1}) = \Theta(n^{1/2})$, and we expect there to be numerous streaks of length $(1/2)\lg n$. Therefore, one streak of such a length is likely to occur. We can conclude that the expected length of the longest streak is $\Theta(\lg n)$.

5.4.4 The online hiring problem

As a final example, let's consider a variant of the hiring problem. Suppose now that you do not wish to interview all the candidates in order to find the best one. You also want to avoid hiring and firing as you find better and better applicants. Instead, you are willing to settle for a candidate who is close to the best, in exchange for hiring exactly once. You must obey one company requirement: after each interview you must either immediately offer the position to the applicant or immediately reject the applicant. What is the trade-off between minimizing the amount of interviewing and maximizing the quality of the candidate hired?

We can model this problem in the following way. After meeting an applicant, you are able to give each one a score. Let $score(i)$ denote the score you give to the ith applicant, and assume that no two applicants receive the same score. After you have seen j applicants, you know which of the j has the highest score, but you do not know whether any of the remaining $n - j$ applicants will receive a higher score. You decide to adopt the strategy of selecting a positive integer $k < n$, interviewing and then rejecting the first k applicants, and hiring the first applicant thereafter who has a higher score than all preceding applicants. If it turns out that the best-qualified applicant was among the first k interviewed, then you hire the nth applicant—the last one interviewed. We formalize this strategy in the procedure ONLINE-MAXIMUM(k, n), which returns the index of the candidate you wish to hire.

```
ONLINE-MAXIMUM(k, n)
1  best-score = −∞
2  for i = 1 to k
3      if score(i) > best-score
4          best-score = score(i)
5  for i = k + 1 to n
6      if score(i) > best-score
7          return i
8  return n
```

If we determine, for each possible value of k, the probability that you hire the most qualified applicant, then you can choose the best possible k and implement the strategy with that value. For the moment, assume that k is fixed. Let

$M(j) = \max\{score(i) : 1 \le i \le j\}$ denote the maximum score among applicants 1 through j. Let S be the event that you succeed in choosing the best-qualified applicant, and let S_i be the event that you succeed when the best-qualified applicant is the ith one interviewed. Since the various S_i are disjoint, we have that $\Pr\{S\} = \sum_{i=1}^{n} \Pr\{S_i\}$. Noting that you never succeed when the best-qualified applicant is one of the first k, we have that $\Pr\{S_i\} = 0$ for $i = 1, 2, \ldots, k$. Thus, we obtain

$$\Pr\{S\} = \sum_{i=k+1}^{n} \Pr\{S_i\} \ . \tag{5.14}$$

We now compute $\Pr\{S_i\}$. In order to succeed when the best-qualified applicant is the ith one, two things must happen. First, the best-qualified applicant must be in position i, an event which we denote by B_i. Second, the algorithm must not select any of the applicants in positions $k+1$ through $i-1$, which happens only if, for each j such that $k+1 \le j \le i-1$, line 6 finds that $score(j) < best\text{-}score$. (Because scores are unique, we can ignore the possibility of $score(j) = best\text{-}score$.) In other words, all of the values $score(k+1)$ through $score(i-1)$ must be less than $M(k)$. If any are greater than $M(k)$, the algorithm instead returns the index of the first one that is greater. We use O_i to denote the event that none of the applicants in position $k+1$ through $i-1$ are chosen. Fortunately, the two events B_i and O_i are independent. The event O_i depends only on the relative ordering of the values in positions 1 through $i-1$, whereas B_i depends only on whether the value in position i is greater than the values in all other positions. The ordering of the values in positions 1 through $i-1$ does not affect whether the value in position i is greater than all of them, and the value in position i does not affect the ordering of the values in positions 1 through $i-1$. Thus, we can apply equation (C.17) on page 1188 to obtain

$$\Pr\{S_i\} = \Pr\{B_i \cap O_i\} = \Pr\{B_i\}\Pr\{O_i\} \ .$$

We have $\Pr\{B_i\} = 1/n$ since the maximum is equally likely to be in any one of the n positions. For event O_i to occur, the maximum value in positions 1 through $i-1$, which is equally likely to be in any of these $i-1$ positions, must be in one of the first k positions. Consequently, $\Pr\{O_i\} = k/(i-1)$ and $\Pr\{S_i\} = k/(n(i-1))$. Using equation (5.14), we have

$$\begin{aligned}
\Pr\{S\} &= \sum_{i=k+1}^{n} \Pr\{S_i\} \\
&= \sum_{i=k+1}^{n} \frac{k}{n(i-1)}
\end{aligned}$$

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

We approximate by integrals to bound this summation from above and below. By the inequalities (A.19) on page 1150, we have

$$\int_k^n \frac{1}{x}\,dx \le \sum_{i=k}^{n-1} \frac{1}{i} \le \int_{k-1}^{n-1} \frac{1}{x}\,dx .$$

Evaluating these definite integrals gives us the bounds

$$\frac{k}{n}(\ln n - \ln k) \le \Pr\{S\} \le \frac{k}{n}(\ln(n-1) - \ln(k-1)) ,$$

which provide a rather tight bound for $\Pr\{S\}$. Because you wish to maximize your probability of success, let us focus on choosing the value of k that maximizes the lower bound on $\Pr\{S\}$. (Besides, the lower-bound expression is easier to maximize than the upper-bound expression.) Differentiating the expression $(k/n)(\ln n - \ln k)$ with respect to k, we obtain

$$\frac{1}{n}(\ln n - \ln k - 1) .$$

Setting this derivative equal to 0, we see that you maximize the lower bound on the probability when $\ln k = \ln n - 1 = \ln(n/e)$ or, equivalently, when $k = n/e$. Thus, if you implement our strategy with $k = n/e$, you succeed in hiring the best-qualified applicant with probability at least $1/e$.

Exercises

5.4-1

How many people must there be in a room before the probability that someone has the same birthday as you do is at least $1/2$? How many people must there be before the probability that at least two people have a birthday on July 4 is greater than $1/2$?

5.4-2

How many people must there be in a room before the probability that two people have the same birthday is at least 0.99? For that many people, what is the expected number of pairs of people who have the same birthday?

5.4-3
You toss balls into b bins until some bin contains two balls. Each toss is independent, and each ball is equally likely to end up in any bin. What is the expected number of ball tosses?

★ ***5.4-4***
For the analysis of the birthday paradox, is it important that the birthdays be mutually independent, or is pairwise independence sufficient? Justify your answer.

★ ***5.4-5***
How many people should be invited to a party in order to make it likely that there are *three* people with the same birthday?

★ ***5.4-6***
What is the probability that a k-string (defined on page 1179) over a set of size n forms a k-permutation? How does this question relate to the birthday paradox?

★ ***5.4-7***
You toss n balls into n bins, where each toss is independent and the ball is equally likely to end up in any bin. What is the expected number of empty bins? What is the expected number of bins with exactly one ball?

★ ***5.4-8***
Sharpen the lower bound on streak length by showing that in n flips of a fair coin, the probability is at least $1 - 1/n$ that a streak of length $\lg n - 2 \lg \lg n$ consecutive heads occurs.

Problems

5-1 Probabilistic counting
With a b-bit counter, we can ordinarily only count up to $2^b - 1$. With R. Morris's ***probabilistic counting***, we can count up to a much larger value at the expense of some loss of precision.

We let a counter value of i represent a count of n_i for $i = 0, 1, \ldots, 2^b - 1$, where the n_i form an increasing sequence of nonnegative values. We assume that the initial value of the counter is 0, representing a count of $n_0 = 0$. The INCREMENT operation works on a counter containing the value i in a probabilistic manner. If $i = 2^b - 1$, then the operation reports an overflow error. Otherwise, the INCREMENT operation increases the counter by 1 with probability $1/(n_{i+1} - n_i)$, and it leaves the counter unchanged with probability $1 - 1/(n_{i+1} - n_i)$.

If we select $n_i = i$ for all $i \geq 0$, then the counter is an ordinary one. More interesting situations arise if we select, say, $n_i = 2^{i-1}$ for $i > 0$ or $n_i = F_i$ (the ith Fibonacci number—see equation (3.31) on page 69).

For this problem, assume that n_{2^b-1} is large enough that the probability of an overflow error is negligible.

a. Show that the expected value represented by the counter after n INCREMENT operations have been performed is exactly n.

b. The analysis of the variance of the count represented by the counter depends on the sequence of the n_i. Let us consider a simple case: $n_i = 100i$ for all $i \geq 0$. Estimate the variance in the value represented by the register after n INCREMENT operations have been performed.

5-2 Searching an unsorted array

This problem examines three algorithms for searching for a value x in an unsorted array A consisting of n elements.

Consider the following randomized strategy: pick a random index i into A. If $A[i] = x$, then terminate; otherwise, continue the search by picking a new random index into A. Continue picking random indices into A until you find an index j such that $A[j] = x$ or until every element of A has been checked. This strategy may examine a given element more than once, because it picks from the whole set of indices each time.

a. Write pseudocode for a procedure RANDOM-SEARCH to implement the strategy above. Be sure that your algorithm terminates when all indices into A have been picked.

b. Suppose that there is exactly one index i such that $A[i] = x$. What is the expected number of indices into A that must be picked before x is found and RANDOM-SEARCH terminates?

c. Generalizing your solution to part (b), suppose that there are $k \geq 1$ indices i such that $A[i] = x$. What is the expected number of indices into A that must be picked before x is found and RANDOM-SEARCH terminates? Your answer should be a function of n and k.

d. Suppose that there are no indices i such that $A[i] = x$. What is the expected number of indices into A that must be picked before all elements of A have been checked and RANDOM-SEARCH terminates?

Now consider a deterministic linear search algorithm. The algorithm, which we call DETERMINISTIC-SEARCH, searches A for x in order, considering $A[1]$, $A[2]$,

$A[3], \ldots, A[n]$ until either it finds $A[i] = x$ or it reaches the end of the array. Assume that all possible permutations of the input array are equally likely.

e. Suppose that there is exactly one index i such that $A[i] = x$. What is the average-case running time of DETERMINISTIC-SEARCH? What is the worst-case running time of DETERMINISTIC-SEARCH?

f. Generalizing your solution to part (e), suppose that there are $k \geq 1$ indices i such that $A[i] = x$. What is the average-case running time of DETERMINISTIC-SEARCH? What is the worst-case running time of DETERMINISTIC-SEARCH? Your answer should be a function of n and k.

g. Suppose that there are no indices i such that $A[i] = x$. What is the average-case running time of DETERMINISTIC-SEARCH? What is the worst-case running time of DETERMINISTIC-SEARCH?

Finally, consider a randomized algorithm SCRAMBLE-SEARCH that first randomly permutes the input array and then runs the deterministic linear search given above on the resulting permuted array.

h. Letting k be the number of indices i such that $A[i] = x$, give the worst-case and expected running times of SCRAMBLE-SEARCH for the cases in which $k = 0$ and $k = 1$. Generalize your solution to handle the case in which $k \geq 1$.

i. Which of the three searching algorithms would you use? Explain your answer.

Chapter notes

Bollobás [65], Hofri [223], and Spencer [420] contain a wealth of advanced probabilistic techniques. The advantages of randomized algorithms are discussed and surveyed by Karp [249] and Rabin [372]. The textbook by Motwani and Raghavan [336] gives an extensive treatment of randomized algorithms.

The RANDOMLY-PERMUTE procedure is by Durstenfeld [128], based on an earlier procedure by Fisher and Yates [143, p. 34].

Several variants of the hiring problem have been widely studied. These problems are more commonly referred to as "secretary problems." Examples of work in this area are the paper by Ajtai, Meggido, and Waarts [11] and another by Kleinberg [258], which ties the secretary problem to online ad auctions.