Probability and Computing

Randomization and Probabilistic Techniques in Algorithms and Data Analysis

Second Edition

Michael Mitzenmacher Eli Upfal

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

4843/24, 2nd Floor, Ansari Road, Daryaganj, Delhi - 110002, India

79 Anson Road, #06-04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107154889
10.1017/9781316651124

First published 2017

Printed in the United Kingdom by Clays, St Ives plc

A catalogue record for this publication is available from the British Library.

Library of Congress Cataloging in Publication Data
Names: Mitzenmacher, Michael, 1969– author. | Upfal, Eli, 1954– author.
Title: Probability and computing / Michael Mitzenmacher Eli Upfal.
Description: Second edition. | Cambridge, United Kingdom ;
New York, NY, USA : Cambridge University Press, [2017] |
Includes bibliographical references and index.
Identifiers: LCCN 2016041654 | ISBN 9781107154889
Subjects: LCSH: Algorithms. | Probabilities. | Stochastic analysis.
Classification: LCC QA274.M574 2017 | DDC 518/.1 – dc23
LC record available at https://lccn.loc.gov/2016041654

ISBN 978-1-107-15488-9 Hardback

Additional resources for this publication at www.cambridge.org/Mitzenmacher.

CHAPTER FIVE

Balls, Bins, and Random Graphs

In this chapter, we focus on one of the most basic of random processes: m balls are thrown randomly into n bins, each ball landing in a bin chosen independently and uniformly at random. We use the techniques we have developed previously to analyze this process and develop a new approach based on what is known as the Poisson approximation. We demonstrate several applications of this model, including a more sophisticated analysis of the coupon collector's problem and an analysis of the Bloom filter data structure. After introducing a closely related model of random graphs, we show an efficient algorithm for finding a Hamiltonian cycle on a random graph with sufficiently many edges. Even though finding a Hamiltonian cycle is NP-hard in general, our result shows that, for a randomly chosen graph, the problem is solvable in polynomial time with high probability.

5.1. Example: The Birthday Paradox

Sitting in lecture, you notice that there are 30 people in the room. Is it more likely that some two people in the room share the same birthday or that no two people in the room share the same birthday?

We can model this problem by assuming that the birthday of each person is a random day from a 365-day year, chosen independently and uniformly at random for each person. This is obviously a simplification; for example, we assume that a person's birthday is equally likely to be any day of the year, we avoid the issue of leap years, and we ignore the possibility of twins! As a model, however, it has the virtue of being easy to understand and analyze.

One way to calculate this probability is to directly count the configurations where two people do not share a birthday. It is easier to think about the configurations where people do not share a birthday than about configurations where some two people do. Thirty days must be chosen from the 365; there are $\binom{365}{30}$ ways to do this. These 30 days can be assigned to the people in any of the 30! possible orders. Hence there are $\binom{365}{30}30!$ configurations where no two people share the same birthday, out of the 365^{30}

ways the birthdays could occur. Thus, the probability is

$$\frac{\binom{365}{30} 30!}{365^{30}}. \tag{5.1}$$

We can also calculate this probability by considering one person at a time. The first person in the room has a birthday. The probability that the second person has a different birthday is $(1 - 1/365)$. The probability that the third person in the room then has a birthday different from the first two, given that the first two people have different birthdays, is $(1 - 2/365)$. Continuing on, the probability that the kth person in the room has a different birthday than the first $k - 1$, assuming that the first $k - 1$ have different birthdays, is $(1 - (k - 1)/365)$. So the probability that 30 people all have different birthdays is the product of these terms, or

$$\left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdot \left(1 - \frac{3}{365}\right) \cdots \left(1 - \frac{29}{365}\right).$$

You can check that this matches the expression (5.1).

Calculations reveal that (to four decimal places) this product is 0.2937, so when 30 people are in the room there is more than a 70% chance that two share the same birthday. A similar calculation shows that only 23 people need to be in the room before it is more likely than not that two people share a birthday.

More generally, if there are m people and n possible birthdays then the probability that all m have different birthdays is

$$\left(1 - \frac{1}{n}\right) \cdot \left(1 - \frac{2}{n}\right) \cdot \left(1 - \frac{3}{n}\right) \cdots \left(1 - \frac{m-1}{n}\right) = \prod_{j=1}^{m-1} \left(1 - \frac{j}{n}\right).$$

Using that $1 - k/n \approx \mathrm{e}^{-k/n}$ when k is small compared to n, we see that if m is small compared to n then

$$\begin{aligned} \prod_{j=1}^{m-1} \left(1 - \frac{j}{n}\right) &\approx \prod_{j=1}^{m-1} \mathrm{e}^{-j/n} \\ &= \exp\left\{-\sum_{j=1}^{m-1} \frac{j}{n}\right\} \\ &= \mathrm{e}^{-m(m-1)/2n} \\ &\approx \mathrm{e}^{-m^2/2n}. \end{aligned}$$

Hence the value for m at which the probability that m people all have different birthdays is $1/2$ is approximately given by the equation

$$\frac{m^2}{2n} = \ln 2,$$

or $m = \sqrt{2n \ln 2}$. For the case $n = 365$, this approximation gives $m = 22.49$ to two decimal places, matching the exact calculation quite well.

Quite tight and formal bounds can be established using bounds in place of the approximations just derived, an option that is considered in Exercise 5.3. The following simple arguments, however, give loose bounds and good intuition. Let us consider each person one at a time, and let E_k be the event that the kth person's birthday does not match any of the birthdays of the first $k-1$ people. Then the probability that the first k people fail to have distinct birthdays is

$$\begin{aligned}\Pr(\bar{E}_1 \cup \bar{E}_2 \cup \cdots \cup \bar{E}_k) &\leq \sum_{i=1}^{k} \Pr(\bar{E}_i) \\ &\leq \sum_{i=1}^{k} \frac{i-1}{n} \\ &= \frac{k(k-1)}{2n}.\end{aligned}$$

If $k \leq \sqrt{n}$ this probability is less than $1/2$, so with $\lfloor \sqrt{n} \rfloor$ people the probability is at least $1/2$ that all birthdays will be distinct.

Now assume that the first $\lceil \sqrt{n} \rceil$ people all have distinct birthdays. Each person after that has probability at least $\sqrt{n}/n = 1/\sqrt{n}$ of having the same birthday as one of these first $\lceil \sqrt{n} \rceil$ people. Hence the probability that the next $\lceil \sqrt{n} \rceil$ people all have different birthdays than the first $\lceil \sqrt{n} \rceil$ people is at most

$$\left(1 - \frac{1}{\sqrt{n}}\right)^{\lceil \sqrt{n} \rceil} < \frac{1}{\mathrm{e}} < \frac{1}{2}.$$

Hence, once there are $2\lceil \sqrt{n} \rceil$ people, the probability is at most $1/\mathrm{e}$ that all birthdays will be distinct.

5.2. Balls into Bins

5.2.1. *The Balls-and-Bins Model*

The birthday paradox is an example of a more general mathematical framework that is often formulated in terms of balls and bins. We have m balls that are thrown into n bins, with the location of each ball chosen independently and uniformly at random from the n possibilities. What does the distribution of the balls in the bins look like? The question behind the birthday paradox is whether or not there is a bin with two balls.

There are several interesting questions that we could ask about this random process. For example, how many of the bins are empty? How many balls are in the fullest bin? Many of these questions have applications to the design and analysis of algorithms.

Our analysis of the birthday paradox showed that, if m balls are randomly placed into n bins then, for some $m = \Omega(\sqrt{n})$, at least one of the bins is likely to have more than one ball in it. Another interesting question concerns the maximum number of balls in a bin, or the maximum *load*. Let us consider the case where $m = n$, so that

the number of balls equals the number of bins and the average load is 1. Of course the maximum possible load is n, but it is very unlikely that all n balls land in the same bin. We seek an upper bound that holds with probability tending to 1 as n grows large. We can show that the maximum load is more than $3 \ln n / \ln \ln n$ with probability at most $1/n$ for sufficiently large n via a direct calculation and a union bound. This is a very loose bound; although the maximum load is in fact $\Omega(\ln n / \ln \ln n)$ with probability close to 1 (as we show later), the constant factor 3 we use here is chosen to simplify the argument and could be reduced with more care.

Lemma 5.1: *When n balls are thrown independently and uniformly at random into n bins, the probability that the maximum load is more than $3 \ln n / \ln \ln n$ is at most $1/n$ for n sufficiently large.*

Proof: The probability that bin 1 receives at least M balls is at most

$$\binom{n}{M}\left(\frac{1}{n}\right)^M.$$

This follows from a union bound; there are $\binom{n}{M}$ distinct sets of M balls, and for any set of M balls the probability that all land in bin 1 is $(1/n)^M$. We now use the inequalities

$$\binom{n}{M}\left(\frac{1}{n}\right)^M \le \frac{1}{M!} \le \left(\frac{e}{M}\right)^M.$$

Here the second inequality is a consequence of the following general bound on factorials: since

$$\frac{k^k}{k!} < \sum_{i=0}^{\infty} \frac{k^i}{i!} = e^k,$$

we have

$$k! > \left(\frac{k}{e}\right)^k.$$

Applying a union bound again allows us to find that, for $M \ge 3 \ln n / \ln \ln n$, the probability that any bin receives at least M balls is bounded above by

$$\begin{aligned} n\left(\frac{e}{M}\right)^M &\le n\left(\frac{e \ln \ln n}{3 \ln n}\right)^{3 \ln n / \ln \ln n} \\ &\le n\left(\frac{\ln \ln n}{\ln n}\right)^{3 \ln n / \ln \ln n} \\ &= e^{\ln n}\left(e^{\ln \ln \ln n - \ln \ln n}\right)^{3 \ln n / \ln \ln n} \\ &= e^{-2 \ln n + 3(\ln n)(\ln \ln \ln n)/\ln \ln n} \\ &\le \frac{1}{n} \end{aligned}$$

for n sufficiently large. ■

5.2.2. *Application: Bucket Sort*

Bucket sort is an example of a sorting algorithm that, under certain assumptions on the input, breaks the $\Omega(n \log n)$ lower bound for standard comparison-based sorting. For example, suppose that we have a set of $n = 2^m$ elements to be sorted and that each element is an integer chosen independently and uniformly at random from the range $[0, 2^k)$, where $k \geq m$. Using Bucket sort, we can sort the numbers in expected time $O(n)$. Here the expectation is over the choice of the random input, since Bucket sort is a completely deterministic algorithm.

Bucket sort works in two stages. In the first stage, we place the elements into n buckets. The jth bucket holds all elements whose first m binary digits correspond to the number j. For example, if $n = 2^{10}$, bucket 3 contains all elements whose first 10 binary digits are 0000000011. When $j < \ell$, the elements of the jth bucket all come before the elements in the ℓth bucket in the sorted order. Assuming that each element can be placed in the appropriate bucket in $O(1)$ time, this stage requires only $O(n)$ time. Because of the assumption that the elements to be sorted are chosen uniformly, the number of elements that land in a specific bucket follows a binomial distribution $B(n, 1/n)$. Buckets can be implemented using linked lists.

In the second stage, each bucket is sorted using any standard quadratic time algorithm (such as Bubblesort or Insertion sort). Concatenating the sorted lists from each bucket in order gives us the sorted order for the elements. It remains to show that the expected time spent in the second stage is only $O(n)$.

The result relies on our assumption regarding the input distribution. Under the uniform distribution, Bucket sort falls naturally into the balls and bins model: the elements are balls, buckets are bins, and each ball falls uniformly at random into a bin.

Let X_j be the number of elements that land in the jth bucket. The time to sort the jth bucket is then at most $c(X_j)^2$ for some constant c. The expected time spent sorting in the second stage is at most

$$\mathbf{E}\left[\sum_{j=1}^{n} c(X_j)^2\right] = c\sum_{j=1}^{n} \mathbf{E}[X_j^2] = cn\mathbf{E}[X_1^2],$$

where the first equality follows from the linearity of expectations and the second follows from symmetry, as $\mathbf{E}[X_j^2]$ is the same for all buckets.

Since X_1 is a binomial random variable $B(n, 1/n)$, using the results of Section 3.2.1 yields

$$\mathbf{E}[X_1^2] = \frac{n(n-1)}{n^2} + 1 = 2 - \frac{1}{n} < 2.$$

Hence the total expected time spent in the second stage is at most $2cn$, so Bucket sort runs in expected linear time.

5.3. The Poisson Distribution

We now consider the probability that a given bin is empty in the balls and bins model with m balls and n bins as well as the expected number of empty bins. For the first bin

to be empty, it must be missed by all m balls. Since each ball hits the first bin with probability $1/n$, the probability the first bin remains empty is

$$\left(1 - \frac{1}{n}\right)^m \approx e^{-m/n};$$

of course, by symmetry this probability is the same for all bins. If X_i is a random variable that is 1 when the ith bin is empty and 0 otherwise, then $\mathbf{E}[X_i] = (1 - 1/n)^m$. Let X be a random variable that represents the number of empty bins. Then, by the linearity of expectations,

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbf{E}[X_i] = n\left(1 - \frac{1}{n}\right)^m \approx ne^{-m/n}.$$

Thus, the expected fraction of empty bins is approximately $e^{-m/n}$. This approximation is very good even for moderately size values of m and n, and we use it frequently throughout this chapter.

We can generalize the preceding argument to find the expected fraction of bins with r balls for any constant r. The probability that a given bin has r balls is

$$\binom{m}{r}\left(\frac{1}{n}\right)^r\left(1 - \frac{1}{n}\right)^{m-r} = \frac{1}{r!}\frac{m(m-1)\cdots(m-r+1)}{n^r}\left(1 - \frac{1}{n}\right)^{m-r}.$$

When m and n are large compared to r, the second factor on the right-hand side is approximately $(m/n)^r$, and the third factor is approximately $e^{-m/n}$. Hence the probability p_r that a given bin has r balls is approximately

$$p_r \approx \frac{e^{-m/n}(m/n)^r}{r!}, \tag{5.2}$$

and the expected number of bins with exactly r balls is approximately np_r. We formalize this relationship in Section 5.3.1.

The previous calculation naturally leads us to consider the following distribution.

Definition 5.1: *A discrete* Poisson random variable X *with parameter* μ *is given by the following probability distribution on* $j = 0, 1, 2, \ldots$:

$$\Pr(X = j) = \frac{e^{-\mu}\mu^j}{j!}.$$

(Note that Poisson random variables differ from Poisson trials, discussed in Section 4.2.1.)

Let us verify that the definition gives a proper distribution in that the probabilities sum to 1:

$$\begin{aligned}\sum_{j=0}^{\infty} \Pr(X = j) &= \sum_{j=0}^{\infty} \frac{e^{-\mu}\mu^j}{j!} \\ &= e^{-\mu}\sum_{j=0}^{\infty} \frac{\mu^j}{j!} \\ &= 1,\end{aligned}$$

where we have used the Taylor expansion $e^x = \sum_{j=0}^{\infty}(x^j/j!)$.

Next we show that the expectation of this random variable is μ:

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{j=0}^{\infty} j \Pr(X = j) \\
&= \sum_{j=1}^{\infty} j \frac{\mathrm{e}^{-\mu}\mu^j}{j!} \\
&= \mu \sum_{j=1}^{\infty} \frac{\mathrm{e}^{-\mu}\mu^{j-1}}{(j-1)!} \\
&= \mu \sum_{j=0}^{\infty} \frac{\mathrm{e}^{-\mu}\mu^{j}}{j!} \\
&= \mu.
\end{aligned}$$

In the context of throwing m balls into n bins, the distribution of the number of balls in a bin is approximately Poisson with $\mu = m/n$, which is exactly the average number of balls per bin, as one might expect.

An important property of Poisson distributions is given in the following lemma.

Lemma 5.2: *The sum of a finite number of independent Poisson random variables is a Poisson random variable.*

Proof: We consider two independent Poisson random variables X and Y with means μ_1 and μ_2; the case of more random variables is simply handled by induction. Now

$$\begin{aligned}
\Pr(X + Y = j) &= \sum_{k=0}^{j} \Pr((X = k) \cap (Y = j - k)) \\
&= \sum_{k=0}^{j} \frac{\mathrm{e}^{-\mu_1}\mu_1^k}{k!} \frac{\mathrm{e}^{-\mu_2}\mu_2^{(j-k)}}{(j-k)!} \\
&= \frac{\mathrm{e}^{-(\mu_1+\mu_2)}}{j!} \sum_{k=0}^{j} \frac{j!}{k!\,(j-k)!} \mu_1^k \mu_2^{(j-k)} \\
&= \frac{\mathrm{e}^{-(\mu_1+\mu_2)}}{j!} \sum_{k=0}^{j} \binom{j}{k} \mu_1^k \mu_2^{(j-k)} \\
&= \frac{\mathrm{e}^{-(\mu_1+\mu_2)}(\mu_1+\mu_2)^j}{j!}.
\end{aligned}$$

In the last equality we used the binomial theorem to simplify the summation. ■

We can also prove Lemma 5.2 using moment generating functions.

Lemma 5.3: *The* moment generating function *of a Poisson random variable with parameter μ is*

$$M_x(t) = \mathrm{e}^{\mu(\mathrm{e}^t - 1)}.$$

Proof: For any t,

$$\mathbf{E}[e^{tX}] = \sum_{k=0}^{\infty} \frac{e^{-\mu}\mu^k}{k!} e^{tk} = e^{\mu(e^t-1)} \sum_{k=0}^{\infty} \frac{e^{-\mu e^t}(\mu e^t)^k}{k!} = e^{\mu(e^t-1)}.$$

■

Given two independent Poisson random variables X and Y with means μ_1 and μ_2, we apply Theorem 4.3 to prove

$$M_{X+Y}(t) = M_X(t) \cdot M_Y(t) = e^{(\mu_1+\mu_2)(e^t-1)},$$

which is the moment generating function of a Poisson random variable with mean $\mu_1 + \mu_2$. By Theorem 4.2, the moment generating function uniquely defines the distribution, and hence the sum $X + Y$ is a Poisson random variable with mean $\mu_1 + \mu_2$.

We can also use the moment generating function of the Poisson distribution to prove that $\mathbf{E}[X^2] = \lambda(\lambda + 1)$ and $\mathbf{Var}[X] = \lambda$ (see Exercise 5.5).

Next we develop a Chernoff bound for Poisson random variables that we will use later in this chapter.

Theorem 5.4: *Let X be a Poisson random variable with parameter μ.*

1. *If $x > \mu$, then*

$$\Pr(X \geq x) \leq \frac{e^{-\mu}(e\mu)^x}{x^x};$$

2. *If $x < \mu$, then*

$$\Pr(X \leq x) \leq \frac{e^{-\mu}(e\mu)^x}{x^x}.$$

3. *For $\delta > 0$,*

$$\Pr(X \geq (1+\delta)\mu) \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu};$$

4. *For $0 < \delta < 1$,*

$$\Pr(X \leq (1-\delta)\mu) \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mu}.$$

Proof: For any $t > 0$ and $x > \mu$,

$$\Pr(X \geq x) = \Pr(e^{tX} \geq e^{tx}) \leq \frac{\mathbf{E}[e^{tX}]}{e^{tx}}.$$

Plugging in the expression for the moment generating function of the Poisson distribution, we have

$$\Pr(X \geq x) \leq e^{\mu(e^t-1)-xt}.$$

Choosing $t = \ln(x/\mu) > 0$ gives

$$\begin{aligned}\Pr(X \geq x) &\leq e^{x-\mu-x\ln(x/\mu)} \\ &= \frac{e^{-\mu}(e\mu)^x}{x^x}.\end{aligned}$$

For any $t < 0$ and $x < \mu$,

$$\Pr(X \leq x) = \Pr(e^{tX} \geq e^{tx}) \leq \frac{\mathbf{E}[e^{tX}]}{e^{tx}}.$$

Hence

$$\Pr(X \leq x) \leq e^{\mu(e^t - 1) - xt}.$$

Choosing $t = \ln(x/\mu) < 0$, it follows that

$$\begin{aligned} \Pr(X \leq x) &\leq e^{x - \mu - x\ln(x/\mu)} \\ &= \frac{e^{-\mu}(e\mu)^x}{x^x}. \end{aligned}$$

The alternate forms of the bound given in parts 3 and 4 follow immediately from parts 1 and 2. ■

5.3.1. *Limit of the Binomial Distribution*

We have shown that, when throwing m balls randomly into n bins, the probability p_r that a bin has r balls is approximately the Poisson distribution with mean m/n. In general, the Poisson distribution is the limit distribution of the binomial distribution with parameters n and p, when n is large and p is small. More precisely, we have the following limit result.

Theorem 5.5: *Let X_n be a binomial random variable with parameters n and p, where p is a function of n and $\lim_{n\to\infty} np = \lambda$ is a constant that is independent of n. Then, for any fixed k,*

$$\lim_{n\to\infty} \Pr(X_n = k) = \frac{e^{-\lambda}\lambda^k}{k!}.$$

This theorem directly applies to the balls-and-bins scenario. Consider the situation where there are m balls and n bins, where m is a function of n and $\lim_{m\to\infty} m/n = \lambda$. Let X_m be the number of balls in a specific bin. Then X_m is a binomial random variable with parameters m and $1/n$. Theorem 5.5 thus applies and says that

$$\lim_{m\to\infty} \Pr(X_m = r) = \frac{e^{-m/n}(m/n)^r}{r!},$$

matching the approximation of Eqn. (5.2).

Before proving Theorem 5.5, we describe some of its applications. Distributions of this type arise frequently and are often modeled by Poisson distributions. For example, consider the number of spelling or grammatical mistakes in a book, including this book. One model for such mistakes is that each word is likely to have an error with some very small probability p. The number of errors is then a binomial random variable with large n and small p that can therefore be treated as a Poisson random variable. As another example, consider the number of chocolate chips inside a chocolate chip cookie. One possible model is to split the volume of the cookie into a large number of small disjoint compartments, so that a chip lands in each compartment with some probability p. With this model, the number of chips in a cookie roughly follows a Poisson distribution.

We will see similar applications of the Poisson distribution in continuous settings in Chapter 8.

Proof of Theorem 5.5: We can write

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

In what follows, we make use of the bound that, for $|x| \leq 1$,

$$e^x(1-x^2) \leq 1 + x \leq e^x, \tag{5.3}$$

which follows from the Taylor series expansion of e^x. (This is left as Exercise 5.7.) Then

$$\begin{aligned}
\Pr(X_n = k) &\leq \frac{n^k}{k!} p^k \frac{(1-p)^n}{(1-p)^k} \\
&\leq \frac{(np)^k}{k!} \frac{e^{-pn}}{1-pk} \\
&= \frac{e^{-pn}(np)^k}{k!} \frac{1}{1-pk}.
\end{aligned}$$

The second line follows from the first by Eqn. (5.3) and the fact that $(1-p)^k \geq 1 - pk$ for $k \geq 0$. Also,

$$\begin{aligned}
\Pr(X_n = k) &\geq \frac{(n-k+1)^k}{k!} p^k (1-p)^n \\
&\geq \frac{((n-k+1)p)^k}{k!} e^{-pn}(1-p^2)^n \\
&\geq \frac{e^{-pn}((n-k+1)p)^k}{k!}(1-p^2 n),
\end{aligned}$$

where in the second inequality we applied Eqn. (5.3) with $x = -p$.

Combining, we have

$$\frac{e^{-pn}(np)^k}{k!} \frac{1}{1-pk} \geq \Pr(X_n = k) \geq \frac{e^{-pn}((n-k+1)p)^k}{k!}(1-p^2 n).$$

In the limit, as n approaches infinity, p approaches zero because the limiting value of pn is the constant λ. Hence $1/(1-pk)$ approaches 1, $1 - p^2 n$ approaches 1, and the difference between $(n-k+1)p$ and np approaches 0. It follows that

$$\lim_{n\to\infty} \frac{e^{-pn}(np)^k}{k!} \frac{1}{1-pk} = \frac{e^{-\lambda}\lambda^k}{k!}$$

and

$$\lim_{n\to\infty} \frac{e^{-pn}((n-k+1)p)^k}{k!}(1-p^2 n) = \frac{e^{-\lambda}\lambda^k}{k!}.$$

Since $\lim_{n\to\infty} \Pr(X_n = k)$ lies between these two values, the theorem follows. ■

5.4. The Poisson Approximation

The main difficulty in analyzing balls-and-bins problems is handling the dependencies that naturally arise in such systems. For example, if we throw m balls into n bins and find that bin 1 is empty, then it is less likely that bin 2 is empty because we know that the m balls must now be distributed among $n-1$ bins. More concretely: if we know the number of balls in the first $n-1$ bins, then the number of balls in the last bin is completely determined. The loads of the various bins are not independent, and independent random variables are generally much easier to analyze, since we can apply Chernoff bounds. It is therefore useful to have a general way to circumvent these sorts of dependencies.

We have already shown that, after throwing m balls independently and uniformly at random into n bins, the distribution of the number of balls in a given bin is approximately Poisson with mean m/n. We would like to say that the joint distribution of the number of balls in *all* the bins is well approximated by assuming the load at *each* bin is an *independent* Poisson random variable with mean m/n. This would allow us to treat bin loads as independent random variables. We show here that we can do this when we are concerned with sufficiently rare events. Specifically, we show in Corollary 5.9 that taking the probability of an event using this Poisson approximation for all of the bins and multiplying it by $e\sqrt{m}$ gives an upper bound for the probability of the event when m balls are thrown into n bins. For rare events, this extra $e\sqrt{m}$ factor will not be significant. To achieve this result, we now introduce some technical machinery.

Suppose that m balls are thrown into n bins independently and uniformly at random, and let $X_i^{(m)}$ be the number of balls in the ith bin, where $1 \leq i \leq n$. Let $Y_1^{(m)}, \ldots, Y_n^{(m)}$ be independent Poisson random variables with mean m/n. We derive a useful relationship between these two sets of random variables. Tighter bounds for specific problems can often be obtained with more detailed analysis, but this approach is quite general and easy to apply.

The difference between throwing m balls randomly and assigning each bin a number of balls that is Poisson distributed with mean m/n is that, in the first case, we know there are m balls in total, whereas in the second case we know only that m is the expected number of balls in all of the bins. But suppose when we use the Poisson distribution we end up with m balls. In this case, we do indeed have that the distribution is the same as if we threw m balls into n bins randomly.

Theorem 5.6: *The distribution of* $(Y_1^{(m)}, \ldots, Y_n^{(m)})$ *conditioned on* $\sum_i Y_i^{(m)} = k$ *is the same as* $(X_1^{(k)}, \ldots, X_n^{(k)})$, *regardless of the value of* m.

Proof: When throwing k balls into n bins, the probability that $(X_1^{(k)}, \ldots, X_n^{(k)}) = (k_1, \ldots, k_n)$ for any $k_1, \ldots, k_n$ satisfying $\sum_i k_i = k$ is given by

$$\frac{\binom{k}{k_1;k_2;\ldots;k_n}}{n^k} = \frac{k!}{(k_1!)(k_2!)\cdots(k_n!)n^k}.$$

Now, for any $k_1, \ldots, k_n$ with $\sum_i k_i = k$, consider the probability that

$$\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right) = (k_1, \ldots, k_n)$$

conditioned on $(Y_1^{(m)}, \ldots, Y_n^{(m)})$ satisfying $\sum_i Y_i^{(m)} = k$:

$$\Pr\left(\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right) = (k_1, \ldots, k_n) \mid \sum_{i=1}^{n} Y_i^{(m)} = k\right) = \frac{\Pr\left(\left(Y_1^{(m)} = k_1\right) \cap \left(Y_1^{(m)} = k_2\right) \cap \cdots \cap \left(Y_n^{(m)} = k_n\right)\right)}{\Pr\left(\sum_{i=1}^{n} Y_i^{(m)} = k\right)}.$$

The probability that $Y_i^{(m)} = k_i$ is $e^{-m/n}(m/n)^{k_i}/k_i!$, since the $Y_i^{(m)}$ are independent Poisson random variables with mean m/n. Also, by Lemma 5.2, the sum of the $Y_i^{(m)}$ is itself a Poisson random variable with mean m. Hence

$$\frac{\Pr\left(\left(Y_1^{(m)} = k_1\right) \cap \left(Y_1^{(m)} = k_2\right) \cap \cdots \cap \left(Y_n^{(m)} = k_n\right)\right)}{\Pr\left(\sum_{i=1}^{n} Y_i^{(m)} = k\right)} = \frac{\prod_{i=1}^{n} e^{-m/n}(m/n)^{k_i}/k_i!}{e^{-m}m^k/k!} = \frac{k!}{(k_1!)(k_2!)\cdots(k_n!)n^k},$$

proving the theorem. ∎

With this relationship between the two distributions, we can prove strong results about *any* function on the loads of the bins.

Theorem 5.7: *Let $f(x_1, \ldots, x_n)$ be a nonnegative function. Then*

$$\mathbf{E}\left[f\left(X_1^{(m)}, \ldots, X_n^{(m)}\right)\right] \leq e\sqrt{m}\,\mathbf{E}\left[f\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right)\right]. \tag{5.4}$$

Proof: We have that

$$\begin{aligned}\mathbf{E}\left[f\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right)\right] &= \sum_{k=0}^{\infty} \mathbf{E}\left[f\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right) \mid \sum_{i=1}^{n} Y_i^{(m)} = k\right] \Pr\left(\sum_{i=1}^{n} Y_i^{(m)} = k\right) \\ &\geq \mathbf{E}\left[f\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right) \mid \sum_{i=1}^{n} Y_i^{(m)} = m\right] \Pr\left(\sum_{i=1}^{n} Y_i^{(m)} = m\right) \\ &= \mathbf{E}\left[f\left(X_1^{(m)}, \ldots, X_n^{(m)}\right)\right] \Pr\left(\sum Y_i^{(m)} = m\right),\end{aligned}$$

where the last equality follows from the fact that the joint distribution of the $Y_i^{(m)}$ given $\sum_{i=1}^{n} Y_i^{(m)} = m$ is exactly that of the $X_i^{(m)}$, as shown in Theorem 5.6. Since $\sum_{i=1}^{n} Y_i^{(m)}$ is Poisson distributed with mean m, we now have

$$\mathbf{E}\left[f\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right)\right] \geq \mathbf{E}\left[f\left(X_1^{(m)}, \ldots, X_n^{(m)}\right)\right] \frac{m^m e^{-m}}{m!}.$$

We use the following loose bound on $m!$, which we prove as Lemma 5.8:

$$m! < e\sqrt{m}\left(\frac{m}{e}\right)^m.$$

This yields

$$\mathbf{E}\left[f\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right)\right] \geq \mathbf{E}\left[f\left(X_1^{(m)}, \ldots, X_n^{(m)}\right)\right] \frac{1}{e\sqrt{m}},$$

and the theorem is proven. ∎

We prove the upper bound we used for factorials, which closely matches the loose lower bound we used in Lemma 5.1.

Lemma 5.8:

$$n! \leq \mathrm{e}\sqrt{n}\left(\frac{n}{\mathrm{e}}\right)^n. \tag{5.5}$$

Proof: We use the fact that

$$\ln(n!) = \sum_{i=1}^{n} \ln i.$$

We first claim that, for $i \geq 2$,

$$\int_{i-1}^{i} \ln x\, dx \geq \frac{\ln(i-1) + \ln i}{2}.$$

This follows from the fact that $\ln x$ is concave, since its second derivative is $-1/x^2$, which is always negative. Therefore,

$$\int_{1}^{n} \ln x\, dx \geq \sum_{i=1}^{n} \ln i - \frac{\ln n}{2}$$

or, equivalently,

$$n \ln n - n + 1 \geq \ln(n!) - \frac{\ln n}{2}.$$

The result now follows simply by exponentiating. ■

Theorem 5.7 holds for any nonnegative function on the number of balls in the bins. In particular, if the function is the indicator function that is 1 if some event occurs and 0 otherwise, then the theorem gives bounds on the probability of events. Let us call the scenario in which the number of balls in the bins are taken to be independent Poisson random variables with mean m/n the *Poisson case,* and the scenario where m balls are thrown into n bins independently and uniformly at random the *exact case.*

Corollary 5.9: *Any event that takes place with probability p in the Poisson case takes place with probability at most* $p\mathrm{e}\sqrt{m}$ *in the exact case.*

Proof: Let f be the indicator function of the event. In this case, $\mathrm{E}[f]$ is just the probability that the event occurs, and the result follows immediately from Theorem 5.7. ■

This is a quite powerful result. It says that any event that happens with small probability in the Poisson case also happens with small probability in the exact case, where balls are thrown into bins. Since in the analysis of algorithms we often want to show that certain events happen with small probability, this result says that we can utilize an

analysis of the Poisson approximation to obtain a bound for the exact case. The Poisson approximation is easier to analyze because the numbers of balls in each bin are independent random variables.[1]

We can actually do even a little bit better in many natural cases. Part of the proof of the following theorem is outlined in Exercises 5.14 and 5.15.

Theorem 5.10: *Let $f(x_1, \ldots, x_n)$ be a nonnegative function such that $\mathbf{E}[f(X_1^{(m)}, \ldots, X_n^{(m)})]$ is either monotonically increasing or monotonically decreasing in m. Then*

$$\mathbf{E}\left[f\left(X_1^{(m)}, \ldots, X_n^{(m)}\right)\right] \leq 2\mathbf{E}\left[f\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right)\right]. \tag{5.6}$$

The following corollary is immediate.

Corollary 5.11: *Let $\mathcal{E}$ be an event whose probability is either monotonically increasing or monotonically decreasing in the number of balls. If $\mathcal{E}$ has probability p in the Poisson case, then $\mathcal{E}$ has probability at most $2p$ in the exact case.*

To demonstrate the utility of this corollary, we again consider the maximum load problem for the case $m = n$. We have shown via a union bound argument that the maximum load is at most $3 \ln n / \ln \ln n$ with high probability. Using the Poisson approximation, we prove the following almost-matching lower bound on the maximum load.

Lemma 5.12: *When n balls are thrown independently and uniformly at random into n bins, the maximum load is at least $\ln n / \ln \ln n$ with probability at least $1 - 1/n$ for n sufficiently large.*

Proof: In the Poisson case, the probability that bin 1 has load at least $M = \ln n / \ln \ln n$ is at least $1/\mathrm{e}M!$, which is the probability it has load exactly M. In the Poisson case, all bins are independent, so the probability that no bin has load at least M is at most

$$\left(1 - \frac{1}{\mathrm{e}M!}\right)^n \leq \mathrm{e}^{-n/(\mathrm{e}M!)}.$$

We now need to choose M so that $\mathrm{e}^{-n/(\mathrm{e}M!)} \leq n^{-2}$, for then (by Theorem 5.7) we will have that the probability that the maximum load is not at least M in the exact case is at most $\mathrm{e}\sqrt{n}/n^2 < 1/n$. This will give the lemma. Because the maximum load is clearly monotonically increasing in the number of balls, we could also apply the slightly better Theorem 5.10, but this would not affect the argument substantially.

It therefore suffices to show that $M! \leq n/2\mathrm{e} \ln n$, or equivalently that $\ln M! \leq \ln n - \ln \ln n - \ln(2\mathrm{e})$. From our bound of Eqn. (5.5), it follows that

$$M! \leq \mathrm{e}\sqrt{M}\left(\frac{M}{\mathrm{e}}\right)^M \leq M\left(\frac{M}{\mathrm{e}}\right)^M$$

[1] There are other ways to handle the dependencies in the balls-and-bins model. In Chapter 13 we describe a more general way to deal with dependencies (using martingales) that applies here. Also, there is a theory of negative dependence that applies to balls-and-bins problems that also allows these dependencies to be dealt with nicely.

when n (and hence $M = \ln n / \ln\ln n$) are suitably large. Hence, for n suitably large,

$$\begin{aligned}
\ln M! &\le M\ln M - M + \ln M \\
&= \frac{\ln n}{\ln\ln n}(\ln\ln n - \ln\ln\ln n) - \frac{\ln n}{\ln\ln n} + (\ln\ln n - \ln\ln\ln n) \\
&\le \ln n - \frac{\ln n}{\ln\ln n} \\
&\le \ln n - \ln\ln n - \ln(2e),
\end{aligned}$$

where in the last two inequalities we have used the fact that $\ln\ln n = o(\ln n / \ln\ln n)$. ■

5.4.1.* *Example: Coupon Collector's Problem, Revisited*

The coupon collector's problem introduced in Section 2.4.1 can be thought of as a balls-and-bins problem. Recall that in this problem there are n different types of coupons, each cereal box yields a coupon chosen independently and uniformly at random from the n types, and you need to buy cereal boxes until you collect one of each coupon. If we think of coupons as bins and cereal boxes as balls, the question becomes: If balls are thrown independently and uniformly at random into bins, how many balls are thrown until all bins have at least one ball? We showed in Section 2.4.1 that the expected number of cereal boxes necessary is $nH(n) \approx n\ln n$; in Section 3.3.1 we showed that, if there are $n\ln n + cn$ cereal boxes, then the probability that not all coupons are collected is at most e^{-c}. These results translate immediately to the balls-and-bins setting. The expected number of balls that must be thrown before each bin has at least one ball is $nH(n)$, and when $n\ln n + cn$ balls are thrown the probability that not all bins have at least one ball is e^{-c}.

We have seen in Chapter 4 that Chernoff bounds yield concentration results for sums of independent 0–1 random variables. We will use here a Chernoff bound for the Poisson distribution to obtain much stronger results for the coupon collector's problem.

Theorem 5.13: *Let X be the number of coupons observed before obtaining one of each of n types of coupons. Then, for any constant c,*

$$\lim_{n\to\infty} \Pr[X > n\ln n + cn] = 1 - e^{-e^{-c}}.$$

This theorem states that, for large n, the number of coupons required should be very close to $n\ln n$. For example, over 98% of the time the number of coupons required lies between $n\ln n - 4n$ and $n\ln n + 4n$. This is an example of a *sharp threshold,* where the random variable is closely concentrated around its mean.

Proof: We look at the problem as a balls-and-bins problem. We begin by considering the Poisson approximation, and then demonstrate that the Poisson approximation gives the correct answer in the limit. For the Poisson approximation, we suppose that the number of balls in each bin is a Poisson random variable with mean $\ln n + c$, so that the expected total number of balls is $m = n\ln n + cn$. The probability that a specific bin is empty is then

$$e^{-(\ln n + c)} = \frac{e^{-c}}{n}.$$

Since all bins are independent under the Poisson approximation, the probability that no bin is empty is

$$\left(1 - \frac{e^{-c}}{n}\right)^n \approx e^{-e^{-c}}.$$

The last approximation is appropriate in the limit as n grows large, so we apply it here.

To show the Poisson approximation is accurate, we undertake the following steps. Consider the experiment where each bin has a Poisson number of balls, each with mean $\ln n + c$. Let $\mathcal{E}$ be the event that no bin is empty, and let X be the number of balls thrown. We have seen that

$$\lim_{n\to\infty} \Pr(\mathcal{E}) = e^{-e^{-c}}.$$

We use $\Pr(\mathcal{E})$ by splitting it as follows:

$$\begin{aligned}\Pr(\mathcal{E}) &= \Pr\left(\mathcal{E} \cap \left(|X-m| \le \sqrt{2m\ln m}\right)\right) + \Pr\left(\mathcal{E} \cap \left(|X-m| > \sqrt{2m\ln m}\right)\right) \\ &= \Pr\left(\mathcal{E} \mid |X-m| \le \sqrt{2m\ln m}\right) \cdot \Pr\left(|X-m| \le \sqrt{2m\ln m}\right) \\ &\quad + \Pr\left(\mathcal{E} \mid |X-m| > \sqrt{2m\ln m}\right) \cdot \Pr\left(|X-m| > \sqrt{2m\ln m}\right). \qquad (5.7)\end{aligned}$$

This representation proves helpful once we establish two facts. First, we show that $\Pr\left(|X-m| > \sqrt{2m\ln m}\right)$ is $o(1)$; that is, the probability that in the Poisson case the number of balls thrown deviates significantly from its mean m is $o(1)$. This guarantees that the second term in the summation on the right of Eqn. (5.7) is $o(1)$. Second, we show that

$$\left|\Pr\left(\mathcal{E} \mid |X-m| \le \sqrt{2m\ln m}\right) - \Pr(\mathcal{E} \mid X = m)\right| = o(1).$$

That is, the difference between our experiment coming up with exactly m balls or just almost m balls makes an asymptotically negligible difference in the probability that every bin has a ball. With these two facts, Eqn. (5.7) becomes

$$\begin{aligned}\Pr(\mathcal{E}) &= \Pr\left(\mathcal{E} \mid |X-m| \le \sqrt{2m\ln m}\right) \cdot \Pr\left(|X-m| \le \sqrt{2m\ln m}\right) \\ &\quad + \Pr\left(\mathcal{E} \mid |X-m| > \sqrt{2m\ln m}\right) \cdot \Pr\left(|X-m| > \sqrt{2m\ln m}\right) \\ &= \Pr\left(\mathcal{E} \mid |X-m| \le \sqrt{2m\ln m}\right) \cdot (1 - o(1)) + o(1) \\ &= \Pr(\mathcal{E} \mid X = m)(1 - o(1)) + o(1),\end{aligned}$$

and hence

$$\lim_{n\to\infty} \Pr(\mathcal{E}) = \lim_{n\to\infty} \Pr(\mathcal{E} \mid X = m).$$

But from Theorem 5.6, the quantity on the right is equal to the probability that every bin has at least one ball when m balls are thrown randomly, since conditioning on m total balls with the Poisson approximation is equivalent to throwing m balls randomly into the n bins. As a result, the theorem follows once we have shown these two facts.

To show that $\Pr\left(|X-m| > \sqrt{2m\ln m}\right)$ is $o(1)$, consider that X is a Poisson random variable with mean m, since it is a sum of independent Poisson random variables. We use the Chernoff bound for the Poisson distribution (Theorem 5.4) to bound this

probability, writing the bound as

$$\Pr(X \geq x) \leq e^{x-m-x\ln(x/m)}.$$

For $x = m + \sqrt{2m\ln m}$, we use that $\ln(1+z) \geq z - z^2/2$ for $z \geq 0$ to show

$$\begin{aligned}\Pr\left(X > m + \sqrt{2m\ln m}\right) &\leq e^{\sqrt{2m\ln m}-(m+\sqrt{2m\ln m})\ln(1+\sqrt{2\ln m/m})} \\ &\leq e^{\sqrt{2m\ln m}-(m+\sqrt{2m\ln m})(\sqrt{2\ln m/m}-\ln m/m)} \\ &= e^{-\ln m+\sqrt{2m\ln m}(\ln m/m)} = o(1).\end{aligned}$$

A similar argument holds if $x < m$, so $\Pr\left(|X - m| > \sqrt{2m\ln m}\right) = o(1)$.

We now show the second fact, that

$$\left|\Pr\left(\mathcal{E} \mid |X - m| \leq \sqrt{2m\ln m}\right) - \Pr(\mathcal{E} \mid X = m)\right| = o(1).$$

Note that $\Pr(\mathcal{E} \mid X = k)$ is increasing in k, since this probability corresponds to the probability that all bins are nonempty when k balls are thrown independently and uniformly at random. The more balls that are thrown, the more likely all bins are nonempty. It follows that

$$\begin{aligned}\Pr\left(\mathcal{E} \mid X = m - \sqrt{2m\ln m}\right) &\leq \Pr\left(\mathcal{E} \mid |X - m| \leq \sqrt{2m\ln m}\right) \\ &\leq \Pr\left(\mathcal{E} \mid X = m + \sqrt{2m\ln m}\right).\end{aligned}$$

Hence we have the bound

$$\begin{aligned}&\left|\Pr(\mathcal{E} \mid |X - m| \leq \sqrt{2m\ln m}) - \Pr(\mathcal{E} \mid X = m)\right| \\ &\qquad \leq \Pr\left(\mathcal{E} \mid X = m + \sqrt{2m\ln m}\right) - \Pr\left(\mathcal{E} \mid X = m - \sqrt{2m\ln m}\right),\end{aligned}$$

and we show the right-hand side is $o(1)$. This is the difference between the probability that all bins receive at least one ball when $m - \sqrt{2m\ln m}$ balls are thrown and when $m + \sqrt{2m\ln m}$ balls are thrown. This difference is equivalent to the probability of the following experiment: we throw $m - \sqrt{2m\ln m}$ balls and there is still at least one empty bin, but after throwing an additional $2\sqrt{2m\ln m}$ balls, all bins are nonempty. In order for this to happen, there must be at least one empty bin after $m - \sqrt{2m\ln m}$ balls; the probability that one of the next $2\sqrt{2m\ln m}$ balls covers this bin is at most $\left(2\sqrt{2m\ln m}\right)/n = o(1)$ by the union bound. Hence this difference is $o(1)$ as well. ■

5.5. Application: Hashing

5.5.1. *Chain Hashing*

The balls-and-bins-model is also useful for modeling *hashing*. For example, consider the application of a password checker, which prevents people from using common, easily cracked passwords by keeping a dictionary of unacceptable passwords. When a user tries to set up a password, the application would like to check if the requested password is part of the unacceptable set. One possible approach for a password checker would be to store the unacceptable passwords alphabetically and do a binary search on the dictionary to check if a proposed password is unacceptable. A binary search would require $\Theta(\log m)$ time for m words.

Another possibility is to place the words into bins and then search the appropriate bin for the word. The words in a bin would be represented by a linked list. The placement of words into bins is accomplished by using a *hash function.* A hash function f from a universe U into a range $[0, n-1]$ can be thought of as a way of placing items from the universe into n bins. Here the universe U would consist of possible password strings. The collection of bins is called a *hash table.* This approach to hashing is called *chain hashing,* since items that fall in the same bin are chained together in a linked list.

Using a hash table turns the dictionary problem into a balls-and-bins problem. If our dictionary of unacceptable passwords consists of m words and the range of the hash function is $[0, n-1]$, then we can model the distribution of words in bins with the same distribution as m balls placed randomly in n bins. We are making a rather strong assumption by presuming that our hash function maps words into bins in a fashion that appears random, so that the location of each word is independent and identically distributed. There is a great deal of theory behind designing hash functions that appear random, and we will not delve into that theory here. We simply model the problem by assuming that hash functions are random. In other words, we assume that (a) for each $x \in U$, the probability that $f(x) = j$ is $1/n$ (for $0 \leq j \leq n-1$) and that (b) the values of $f(x)$ for each x are independent of each other. Notice that this does not mean that every evaluation of $f(x)$ yields a different random answer! The value of $f(x)$ is fixed for all time; it is just equally likely to take on any value in the range.

Let us consider the search time when there are n bins and m words. To search for an item, we first hash it to find the bin that it lies in and then search sequentially through the linked list for it. If we search for a word that is not in our dictionary, the expected number of words in the bin the word hashes to is m/n. If we search for a word that is in our dictionary, the expected number of other words in that word's bin is $(m-1)/n$, so the expected number of words in the bin is $1 + (m-1)/n$. If we choose $n = m$ bins for our hash table, then the expected number of words we must search through in a bin is constant. If the hashing takes constant time, then the total expected time for the search is constant.

The maximum time to search for a word, however, is proportional to the maximum number of words in a bin. We have shown that when $n = m$ this maximum load is $\Theta(\ln n / \ln \ln n)$ with probability close to 1, and hence with high probability this is the maximum search time in such a hash table. While this is still faster than the required time for standard binary search, it is much slower than the average, which can be a drawback for many applications.

Another drawback of chain hashing can be wasted space. If we use n bins for n items, several of the bins will be empty, potentially leading to wasted space. The space wasted can be traded off against the search time by making the average number of words per bin larger than 1.

5.5.2. *Hashing: Bit Strings*

If we want to save space instead of time, we can use hashing in another way. Again, we consider the problem of keeping a dictionary of unsuitable passwords. Assume that a password is restricted to be eight ASCII characters, which requires 64 bits (8 bytes) to

represent. Suppose we use a hash function to map each word into a 32-bit string. This string will serve as a short *fingerprint* for the word; just as a fingerprint is a succinct way of identifying people, the fingerprint string is a succinct way of identifying a word. We keep the fingerprints in a sorted list. To check if a proposed password is unacceptable, we calculate its fingerprint and look for it on the list, say by a binary search.[2] If the fingerprint is on the list, we declare the password unacceptable.

In this case, our password checker may not give the correct answer! It is possible for a user to input an acceptable password, only to have it rejected because its fingerprint matches the fingerprint of an unacceptable password. Hence there is some chance that hashing will yield a *false positive*: it may falsely declare a match when there is not an actual match. The problem is that – unlike fingerprints for human beings – our fingerprints do not uniquely identify the associated word. This is the only type of mistake this algorithm can make; it does *not* allow a password that is in the dictionary of unsuitable passwords. In the password application, allowing false positives means our algorithm is overly conservative, which is probably acceptable. Letting easily cracked passwords through, however, would probably not be acceptable.

To place the problem in a more general context, we describe it as an *approximate set membership* problem. Suppose we have a set $S = \{s_1, s_2, \ldots, s_m\}$ of m elements from a large universe U. We would like to represent the elements in such a way that we can quickly answer queries of the form "is x an element of S?" We would also like the representation to take as little space as possible. In order to save space, we would be willing to allow occasional mistakes in the form of false positives. Here the unallowable passwords correspond to our set S.

How large should the range of the hash function used to create the fingerprints be? Specifically, if we are working with bits, how many bits should we use to create a fingerprint? Obviously, we want to choose the number of bits that gives an acceptable probability for a false positive match. The probability that an acceptable password has a fingerprint that is different from any specific unallowable password in S is $(1 - 1/2^b)$. It follows that if the set S has size m and if we use b bits for the fingerprint, then the probability of a false positive for an acceptable password is $1 - (1 - 1/2^b)^m \geq 1 - e^{-m/2^b}$. If we want this probability of a false positive to be less than a constant c, we need

$$e^{-m/2^b} \geq 1 - c,$$

which implies that

$$b \geq \log_2 \frac{m}{\ln(1/(1-c))}.$$

That is, we need $b = \Omega(\log_2 m)$ bits. On the other hand, if we use $b = 2\log_2 m$ bits, then the probability of a false positive falls to

$$1 - \left(1 - \frac{1}{m^2}\right)^m < \frac{1}{m}.$$

[2] In this case the fingerprints will be uniformly distributed over all 32-bit strings. There are faster algorithms for searching over sets of numbers with this distribution, just as Bucket sort allows faster sorting than standard comparison-based sorting when the elements to be sorted are from a uniform distribution, but we will not concern ourselves with this point here.

In our example, if our dictionary has $2^{16} = 65{,}536$ words, then using 32 bits when hashing yields a false positive probability of just less than $1/65{,}536$.

5.5.3. *Bloom Filters*

We can generalize the hashing ideas of Sections 5.5.1 and 5.5.2 to achieve more interesting trade-offs between the space required and the false positive probability. The resulting data structure for the approximate set membership problem is called a *Bloom filter.*

A Bloom filter consists of an array of n bits, $A[0]$ to $A[n-1]$, initially all set to 0. A Bloom filter uses k independent random hash functions $h_1, \ldots, h_k$ with range $\{0, \ldots, n-1\}$. We make the usual assumption for analysis that these hash functions map each element in the universe to a random number independently and uniformly over the range $\{0, \ldots, n-1\}$. Suppose that we use a Bloom filter to represent a set $S = \{s_1, s_2, \ldots, s_m\}$ of m elements from a large universe U. For each element $s \in S$, the bits $A[h_i(s)]$ are set to 1 for $1 \leq i \leq k$. A bit location can be set to 1 multiple times, but only the first change has an effect. To check if an element x is in S, we check whether all array locations $A[h_i(x)]$ for $1 \leq i \leq k$ are set to 1. If not, then clearly x is not a member of S, because if x were in S then all locations $A[h_i(x)]$ for $1 \leq i \leq k$ would be set to 1 by construction. If all $A[h_i(x)]$ are set to 1, we assume that x is in S, although we could be wrong. We would be wrong if all of the positions $A[h_i(x)]$ were set to 1 by elements of S even though x is not in the set. Hence Bloom filters may yield false positives. Figure 5.1 shows an example.

The probability of a false positive for an element not in the set – the *false positive probability* – can be calculated in a straightforward fashion, given our assumption that the hash functions are random. After all the elements of S are hashed into the Bloom filter, the probability that a specific bit is still 0 is

$$\left(1 - \frac{1}{n}\right)^{km} \approx \mathrm{e}^{-km/n}.$$

We let $p = \mathrm{e}^{-km/n}$. To simplify the analysis, let us temporarily assume that a fraction p of the entries are still 0 after all of the elements of S are hashed into the Bloom filter.

The probability of a false positive is then

$$\left(1 - \left(1 - \frac{1}{n}\right)^{km}\right)^{k} \approx (1 - \mathrm{e}^{-km/n})^k = (1-p)^k.$$

We let $f = (1 - \mathrm{e}^{-km/n})^k = (1-p)^k$. From now on, for convenience we use the asymptotic approximations p and f to represent (respectively) the probability that a bit in the Bloom filter is 0 and the probability of a false positive.

Suppose that we are given m and n and wish to optimize the number of hash functions k in order to minimize the false positive probability f. There are two competing forces: using more hash functions gives us more chances to find a 0-bit for an element that is not a member of S, but using fewer hash functions increases the fraction of 0-bits

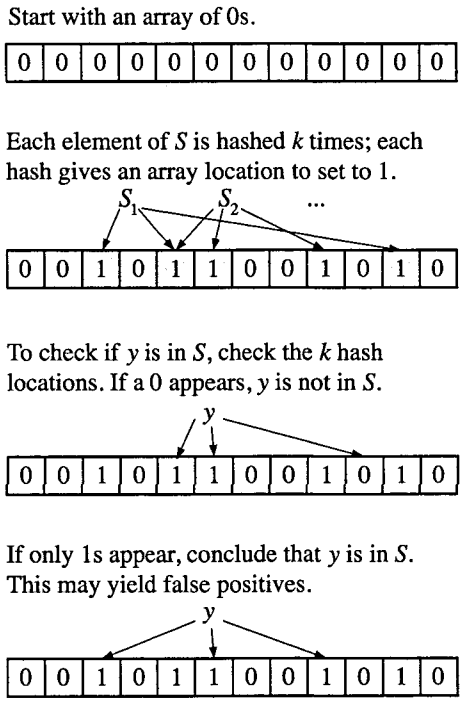

Figure 5.1: Example of how a Bloom filter functions.

in the array. The optimal number of hash functions that minimizes f as a function of k is easily found taking the derivative. Let $g = k\ln(1 - e^{-km/n})$, so that $f = e^g$ and minimizing the false positive probability f is equivalent to minimizing g with respect to k. We find

$$\frac{dg}{dk} = \ln(1 - e^{-km/n}) + \frac{km}{n}\frac{e^{-km/n}}{1 - e^{-km/n}}.$$

It is easy to check that the derivative is zero when $k = (\ln 2) \cdot (n/m)$ and that this point is a global minimum. In this case the false positive probability f is $(1/2)^k \approx (0.6185)^{n/m}$. The false positive probability falls exponentially in n/m, the number of bits used per item. In practice, of course, k must be an integer, so the best possible choice of k may lead to a slightly higher false positive rate.

A Bloom filter is like a hash table, but instead of storing set items we simply use one bit to keep track of whether or not an item hashed to that location. If $k = 1$, we have just one hash function and the Bloom filter is equivalent to a hashing-based fingerprint system, where the list of the fingerprints is stored in a 0–1 bit array. Thus Bloom filters can be seen as a generalization of the idea of hashing-based fingerprints. As we saw when using fingerprints, to get even a small constant probability of a false positive required $\Omega(\log m)$ fingerprint bits per item. In many practical applications, $\Omega(\log m)$ bits per item can be too many. Bloom filters allow a constant probability of a false positive while keeping n/m, the number of bits of storage required per item, constant. For many applications, the small space requirements make a constant probability of

error acceptable. For example, in the password application, we may be willing to accept false positive rates of 1% or 2%.

Bloom filters are highly effective even if $n = cm$ for a small constant c, such as $c = 8$. In this case, when $k = 5$ or $k = 6$ the false positive probability is just over 0.02. This contrasts with the approach of hashing each element into $\Theta(\log m)$ bits. Bloom filters require significantly fewer bits while still achieving a very good false positive probability.

It is also interesting to frame the optimization another way. Consider f, the probability of a false positive, as a function of p. We find

$$\begin{aligned} f &= (1-p)^k \\ &= (1-p)^{(-\ln p)(n/m)} \\ &= (e^{-\ln(p)\ln(1-p)})^{n/m}. \end{aligned} \tag{5.8}$$

From the symmetry of this expression, it is easy to check that $p = 1/2$ minimizes the false positive probability f. Hence the optimal results are achieved when each bit of the Bloom filter is 0 with probability $1/2$. An optimized Bloom filter looks like a random bit string.

To conclude, we reconsider our assumption that the fraction of entries that are still 0 after all of the elements of S are hashed into the Bloom filter is p. Each bit in the array can be thought of as a bin, and hashing an item is like throwing a ball. The fraction of entries that are still 0 after all of the elements of S are hashed is therefore equivalent to the fraction of empty bins after mk balls are thrown into n bins. Let X be the number of such bins when mk balls are thrown. The expected fraction of such bins is

$$p' = \left(1 - \frac{1}{n}\right)^{km}.$$

The events of different bins being empty are not independent, but we can apply Corollary 5.9, along with the Chernoff bound of Eqn. (4.6), to obtain

$$\Pr(|X - np'| \geq \varepsilon n) \leq 2e\sqrt{n}e^{-n\varepsilon^2/3p'}.$$

Actually, Corollary 5.11 applies as well, since the number of 0-entries – which corresponds to the number of empty bins – is monotonically decreasing in the number of balls thrown. The bound tells us that the fraction of empty bins is close to p' (when n is reasonably large) and that p' is very close to p. Our assumption that the fraction of 0-entries in the Bloom filter is p is therefore quite accurate for predicting actual performance.

5.5.4. *Breaking Symmetry*

As our last application of hashing, we consider how hashing provides a simple way to break symmetry. Suppose that n users want to utilize a resource, such as time on a supercomputer. They must use the resource sequentially, one at a time. Of course, each user wants to be scheduled as early as possible. How can we decide a permutation of the users quickly and fairly?

If each user has an identifying name or number, hashing provides one possible solution. Hash each user's identifier into 2^b bits, and then take the permutation given by the sorted order of the resulting numbers. That is, the user whose identifier gives the smallest number when hashed comes first, and so on. For this approach to work, we do not want two users to hash to the same value, since then we must decide again how to order these users.

If b is sufficiently large, then with high probability the users will all obtain distinct hash values. One can analyze the probability that two hash values collide by using the analysis from Section 5.1 for the birthday paradox; hash values correspond to birthdays. We here use a simpler analysis based just on using a union bound. There are $\binom{n}{2}$ pairs of users. The probability that any specific pair has the same hash value is $1/2^b$. Hence the probability that any pair has the same hash value is at most

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

Choosing $b = 3\log_2 n - 1$ guarantees success with probability at least $1 - 1/n$.

This solution is extremely flexible, making it useful for many situations in distributed computing. For example, new users can easily be added into the schedule at any time, as long as they do not hash to the same number as another scheduled user.

A related problem is *leader election.* Suppose that instead of trying to order all of the users, we simply want to fairly choose a leader from them. Again, if we have a suitably random hash function then we can simply take the user whose hash value is the smallest. An analysis of this scheme is left as Exercise 5.26.

5.6. Random Graphs

5.6.1. *Random Graph Models*

There are many NP-hard computational problems defined on graphs: Hamiltonian cycle, independent set, vertex cover, and so forth. One question worth asking is whether these problems are hard for most inputs or just for a relatively small fraction of all graphs. *Random graph* models provide a probabilistic setting for studying such questions.

Most of the work on random graphs has focused on two closely related models, $G_{n,p}$ and $G_{n,N}$. In $G_{n,p}$ we consider all undirected graphs on n distinct vertices $v_1, v_2, \ldots, v_n$. A graph with a given set of m edges has probability

$$p^m(1-p)^{\binom{n}{2}-m}.$$

One way to generate a random graph in $G_{n,p}$ is to consider each of the $\binom{n}{2}$ possible edges in some order and then independently add each edge to the graph with probability p.

The expected number of edges in the graph is therefore $\binom{n}{2}p$, and each vertex has expected degree $(n-1)p$.

In the $G_{n,N}$ model, we consider all undirected graphs on n vertices with exactly N edges. There are $\binom{\binom{n}{2}}{N}$ possible graphs, each selected with equal probability. One way to generate a graph uniformly from the graphs in $G_{n,N}$ is to start with a graph with no edges. Choose one of the $\binom{n}{2}$ possible edges uniformly at random and add it to the edges in the graph. Now choose one of the remaining $\binom{n}{2}-1$ possible edges independently and uniformly at random and add it to the graph. Similarly, continue choosing one of the remaining unchosen edges independently and uniformly at random until there are N edges.

The $G_{n,p}$ and $G_{n,N}$ models are related; when $p = N/\binom{n}{2}$, the number of edges in a random graph in $G_{n,p}$ is concentrated around N, and conditioned on a graph from $G_{n,p}$ having N edges, that graph is uniform over all the graphs from $G_{n,N}$. The relationship is similar to the relationship between throwing m balls into n bins and having each bin have a Poisson distributed number of balls with mean m/n.

Here, for example is one way of formalizing the relationship between the $G_{n,p}$ and $G_{n,N}$ models. A graph property is a property that holds for a graph regardless of how the vertices are labeled, so it holds for all possible isomorphisms of the graph. We say that a graph property is monotone increasing if whenever the property holds for $G = (V, E)$ it holds also for any graph $G' = (V, E')$ with $E \subseteq E'$; monotone decreasing graph properties are defined similarly. For example, the property that a graph is connected is a monotone increasing graph property, as is the property that a graph contains a connected component of at least k vertices for any particular value of k. The property that a graph is a tree, however, is not a monotone graph property, although the property that the graph contains no cycles is a monotone decreasing graph property. We have the following lemma:

Lemma 5.14: *For a given monotone increasing graph property let $P(n, N)$ be the probability that the property holds for a graph in $G_{n,N}$ and $P(n, p)$ the probability that it holds for a graph in $G_{n,p}$. Let $p^+ = (1+\epsilon)N/\binom{n}{2}$ and $p^- = (1-\epsilon)N/\binom{n}{2}$ for a constant $1 > \epsilon > 0$. Then*

$$P(n, p^-) - e^{-O(N)} \leq P(n, N) \leq P(n, p^+) + e^{-O(N)}.$$

Proof: Let X be a random variable giving the number of edges that occur when a graph is chosen from G_{n,p^-}. Conditioned on $X = k$, a random graph from G_{n,p^-} is equivalent to a graph from $G_{n,k}$, since the k edges chosen are equally likely to be any subset of k edges. Hence

$$P(n, p^-) = \sum_{k=0}^{\binom{n}{2}} P(n, k) \Pr(X = k).$$

In particular,

$$P(n, p^-) = \sum_{k \leq N} P(n, k) \Pr(X = k) + \sum_{k > N} P(n, k) \Pr(X = k).$$

Also, for a monotone increasing graph property, $P(n,k) \leq P(n,N)$ for $k \leq N$. Hence

$$P(n,p^-) \leq \Pr(X \leq N)P(n,N) + \Pr(X > N) \leq P(n,N) + \Pr(X > N).$$

However, $\Pr(X > N)$ can be bounded by a standard Chernoff bound; X is the sum of $\binom{n}{2}$ independent Bernoulli random variables, and hence by Theorem 4.4

$$\Pr(X > N) = \Pr\left(X > \frac{1}{1-\epsilon}\mathrm{E}[X]\right) \leq \Pr(X > (1+\epsilon)\mathrm{E}[X]) \leq \mathrm{e}^{-(1-\epsilon)\epsilon^2 N/3}.$$

Here we have used that $\frac{1}{1-\epsilon} > 1+\epsilon$ for $0 < \epsilon < 1$.

Similarly,

$$P(n,p^+) = \sum_{k<N} P(n,k)\Pr(X=k) + \sum_{k\geq N} P(n,k)\Pr(X=k),$$

so

$$P(n,p^+) \geq \Pr(X \geq N)P(n,N) \geq P(n,N) - \Pr(X < N).$$

By Theorem 4.5

$$\Pr(X > N) = \Pr\left(X < \frac{1}{1+\epsilon}\mathrm{E}[X]\right) \leq \Pr\left(X < \left(1+\frac{\epsilon}{2}\right)\mathrm{E}[X]\right) \leq \mathrm{e}^{-(1+\epsilon)\epsilon^2 N/8},$$

where here we have used that $\frac{1}{1+\epsilon} < 1 - \epsilon/2$ for $0 < \epsilon < 1$. ■

A similar result holds for monotone decreasing properties. Another formalization of the relationship between the graph models is given as Exercise 5.18.

There are many additional similarities between random graphs and the balls-and-bins model. Throwing edges into the graph as in the $G_{n,N}$ model is like throwing balls into bins. However, since each edge has two endpoints, each edge is like throwing two balls at once into two different bins. The pairing defined by the edges adds a rich structure that does not exist in the balls-and-bins model. Yet we can often utilize the relation between the two models to simplify analysis in random graph models. For example, in the coupon collector's problem we found that when we throw $n\ln n + cn$ balls, the probability that there are no empty bins converges to $\mathrm{e}^{-\mathrm{e}^{-c}}$ as n grows to infinity. Similarly, we have the following theorem for random graphs, which is left as Exercise 5.20.

Theorem 5.15: *Let $N = \frac{1}{2}(n\ln n + cn)$. Then the probability that there are no isolated vertices (vertices with degree 0) in $G_{n,N}$ converges to $\mathrm{e}^{-\mathrm{e}^{-c}}$ as n grows to infinity.*

5.6.2. *Application: Hamiltonian Cycles in Random Graphs*

A Hamiltonian path in a graph is a path that traverses each vertex exactly once. A Hamiltonian cycle is a cycle that traverses each vertex exactly once. We show an interesting connection between random graphs and balls-and-bins problems by analyzing a simple and efficient algorithm for finding Hamiltonian cycles in random graphs. The algorithm is randomized, and its probabilistic analysis is over both the input distribution and the random choices of the algorithm. Finding a Hamiltonian cycle in a graph

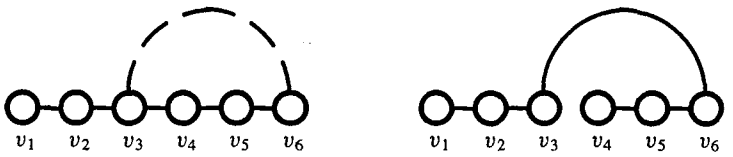

Figure 5.2: The rotation of the path $v_1, v_2, v_3, v_4, v_5, v_6$ with the edge (v_6, v_3) yields a new path $v_1, v_2, v_3, v_6, v_5, v_4$.

is an NP-hard problem. However, our analysis of this algorithm shows that finding a Hamiltonian cycle is not hard for suitably randomly selected graphs, even though it may be hard to solve in general.

Our algorithm will make use of a simple operation called a *rotation.* Let G be an undirected graph. Suppose that

$$P = v_1, v_2, \ldots, v_k$$

is a simple path in G and that (v_k, v_i) is an edge of G. Then

$$P' = v_1, v_2, \ldots, v_i, v_k, v_{k-1}, \ldots, v_{i+2}, v_{i+1}$$

is also a simple path, which we refer to as the rotation of P with the rotation edge (v_k, v_i); see Figure 5.2.

We first consider a simple, natural algorithm that proves challenging to analyze. We assume that our input is presented as a list of adjacent edges for each vertex in the graph, with the edges of each list being given in a random order according to independent and uniform random permutations. Initially, the algorithm chooses an arbitrary vertex to start the path; this is the initial *head* of the path. The head is always one of the endpoints of the path. From this point on, the algorithm either "grows" the path deterministically from the head, or rotates the path – as long as there is an adjacent edge remaining on the head's list. See Algorithm 5.1.

The difficulty in analyzing this algorithm is that, once the algorithm views some edges in the edge lists, the distribution of the remaining edges is conditioned on the edges the algorithm has already seen. We circumvent this difficulty by considering a modified algorithm that, though less efficient, avoids this conditioning issue and so is easier to analyze for the random graphs we consider. See Algorithm 5.2. Each vertex v keeps two lists. The list used-edges(v) contains edges adjacent to v that have been used in the course of the algorithm while v was the head; initially this list is empty. The list unused-edges(v) contains other edges adjacent to v that have not been used.

We initially analyze the algorithm assuming a specific model for the initial unused-edges lists. We subsequently relate this model to the $G_{n,p}$ model for random graphs. Assume that each of the $n - 1$ possible edges connected to a vertex v is initially on the unused-edges list for vertex v independently with some probability q. We also assume these edges are in a random order. One way to think of this is that, before beginning the algorithm, we create the unused-edges list for each vertex v by inserting each possible edge (v, u) with probability q; we think of the corresponding graph G as being the graph including all edges that were inserted on some unused-edges list. Notice that

Hamiltonian Cycle Algorithm:

Input: A graph $G = (V, E)$ with n vertices.

Output: A Hamiltonian cycle, or failure.

1. Start with a random vertex as the head of the path.
2. Repeat the following steps until the rotation edge closes a Hamiltonian cycle or the unused-edges list of the head of the path is empty:
   (a) Let the current path be $P = v_1, v_2, \ldots, v_k$, where v_k is the head, and let (v_k, u) be the first edge in the head's list.
   (b) Remove (v_k, u) from the head's list and u's list.
   (c) If $u \neq v_i$ for $1 \leq i \leq k$, add $u = v_{k+1}$ to the end of the path and make it the head.
   (d) Otherwise, if $u = v_i$, rotate the current path with (v_k, v_i) and set v_{i+1} to be the head. (This step closes the Hamiltonian path if $k = n$ and the chosen edge is (v_n, v_1).)
3. Return a Hamiltonian cycle if one was found or failure if no cycle was found.

Algorithm 5.1: Hamiltonian cycle algorithm.

Modified Hamiltonian Cycle Algorithm:

Input: A graph $G = (V, E)$ with n vertices and associated edge lists.

Output: A Hamiltonian cycle, or failure.

1. Start with a random vertex as the head of the path.
2. Repeat the following steps until the rotation edge closes a Hamiltonian cycle or the unused-edges list of the head of the path is empty:
   (a) Let the current path be $P = v_1, v_2, \ldots, v_k$, with v_k being the head.
   (b) Execute i, ii, or iii with probabilities $1/n$, $|\text{used-edges}(v_k)|/n$, and $1 - 1/n - |\text{used-edges}(v_k)|/n$, respectively:
      i. Reverse the path, and make v_1 the head.
      ii. Choose uniformly at random an edge from used-edges(v_k); if the edge is (v_k, v_i), rotate the current path with (v_k, v_i) and set v_{i+1} to be the head. (If the edge is (v_k, v_{k-1}), then no change is made.)
      iii. Select the first edge from unused-edges(v_k), call it (v_k, u). If $u \neq v_i$ for $1 \leq i \leq k$, add $u = v_{k+1}$ to the end of the path and make it the head. Otherwise, if $u = v_i$, rotate the current path with (v_k, v_i) and set v_{i+1} to be the head. (This step closes the Hamiltonian path if $k = n$ and the chosen edge is (v_n, v_1).)
   (c) Update the used-edges and unused-edges lists appropriately.
3. Return a Hamiltonian cycle if one was found or failure if no cycle was found.

Algorithm 5.2: Modified Hamiltonian cycle algorithm.

this means an edge (v, u) could initially be on the unused-edges list for v but not for u. Also, when an edge (v, u) is first used in the algorithm, if v is the head then it is removed just from the unused-edges list of v; if the edge is on the unused-edges list for u, it remains on this list.

By choosing the rotation edge from either the used-edges list or the unused-edges list with appropriate probabilities and then reversing the path with some small probability in each step, we modify the rotation process so that the next head of the list is chosen uniformly at random from among all vertices of the graph. Once we establish this property, the progress of the algorithm can be analyzed through a straightforward application of our analysis of the coupon collector's problem.

The modified algorithm appears wasteful; reversing the path or rotating with one of the used edges cannot increase the path length. Also, we may not be taking advantage of all the possible edges of G at each step. The advantage of the modified algorithm is that it proves easier to analyze, owing to the following lemma.

Lemma 5.16: *Suppose the modified Hamiltonian cycle algorithm is run on a graph chosen using the described model. Let V_t be the head vertex after the tth step. Then, for any vertex u, as long as at the tth step there is at least one unused edge available at the head vertex,*

$$\Pr(V_{t+1} = u \mid V_t = u_t, V_{t-1} = u_{t-1}, \ldots, V_0 = u_0) = 1/n.$$

That is, the head vertex can be thought of as being chosen uniformly at random from all vertices at each step, regardless of the history of the process.

Proof: Consider the possible cases when the path is $P = v_1, v_2, \ldots, v_k$.

The only way v_1 can become the head is if the path is reversed, so $V_{t+1} = v_1$ with probability $1/n$.

If $u = v_{i+1}$ is a vertex that lies on the path and (v_k, v_i) is in used-edges(v_k), then the probability that $V_{t+1} = u$ is

$$\frac{|\text{used-edges}(v_k)|}{n} \frac{1}{|\text{used-edges}(v_k)|} = \frac{1}{n}.$$

If u is not covered by one of the first two cases then we use the fact that, when an edge is chosen from unused-edges(v_k), the adjacent vertex is uniform over all the $n - |\text{used-edges}(v_k)| - 1$ remaining vertices. This follows from the principle of deferred decisions. Our initial setup required the unused-edges list for v_k to be constructed by including each possible edge with probability q and randomizing the order of the list. This is equivalent to choosing X neighboring vertices for v_k, where X is a $B(n-1, q)$ random variable and the X vertices are chosen uniformly at random without replacement. Because v_k's list was determined independently from the lists of the other vertices, the history of the algorithm tells us nothing about the remaining edges in unused-edges(v_k), and the principle of deferred decisions applies. Hence any edge in v_k's unused-edges list that we have not seen is by construction equally likely to connect to any of the $n - |\text{used-edges}(v_k)| - 1$ remaining possible neighboring vertices.

If $u = v_{i+1}$ is a vertex on the path but (v_k, v_i) is not in used-edges(v_k), then the probability that $V_{t+1} = u$ is the probability that the edge (v_k, v_i) is chosen from unused-edges(v_k) as the next rotation edge, which is

$$\left(1 - \frac{1}{n} - \frac{|\text{used-edges}(v_k)|}{n}\right)\left(\frac{1}{n - |\text{used-edges}(v_k)| - 1}\right) = \frac{1}{n}. \tag{5.9}$$

Finally, if u is not on the path, then the probability that $V_{t+1} = u$ is the probability that the edge (v_{k+1}, u) is chosen from unused-edges(v_k). But this has the same probability as in Eqn. (5.9). ■

For Algorithm 5.2, the problem of finding a Hamiltonian path looks exactly like the coupon collector's problem; the probability of finding a new vertex to add to the path when there are k vertices left to be added is k/n. Once all the vertices are on the path, the probability that a cycle is closed in each rotation is $1/n$. Hence, if no list of unused-edges is exhausted then we can expect a Hamiltonian path to be formed in about $O(n \ln n)$ rotations, with about another $O(n \ln n)$ rotations to close the path to form a Hamiltonian cycle. More concretely, we can prove the following theorem.

Theorem 5.17: *Suppose the input to the modified Hamiltonian cycle algorithm initially has unused-edge lists where each edge (v, u) with $u \neq v$ is placed on v's list independently with probability $q \geq 20 \ln n/n$. Then the algorithm successfully finds a Hamiltonian cycle in $O(n \ln n)$ iterations of the repeat loop (step 2) with probability $1 - O(n^{-1})$.*

Note that we did not assume that the input random graph has a Hamiltonian cycle. A corollary of the theorem is that, with high probability, a random graph chosen in this way has a Hamiltonian cycle.

Proof of Theorem 5.17: Consider the following two events.

$\mathcal{E}_1$: The algorithm ran for $3n \ln n$ steps with no unused-edges list becoming empty, but it failed to construct a Hamiltonian cycle.

$\mathcal{E}_2$: At least one unused-edges list became empty during the first $3n \ln n$ iterations of the loop.

For the algorithm to fail, either event $\mathcal{E}_1$ or $\mathcal{E}_2$ must occur. We first bound the probability of $\mathcal{E}_1$. Lemma 5.16 implies that, as long as there is no empty unused-edges list in the first $3n \ln n$ iterations of step 2 of Algorithm 5.2, in each iteration the next head of the path is uniform among the n vertices of the graph. To bound $\mathcal{E}_1$, we therefore consider the probability that more than $3n \ln n$ iterations are required to find a Hamiltonian cycle when the head is chosen uniformly at random each iteration.

The probability that the algorithm takes more than $2n \ln n$ iterations to find a Hamiltonian path is exactly the probability that a coupon collector's problem on n types requires more than $2n \ln n$ coupons. The probability that any specific coupon type has not been found among $2n \ln n$ random coupons is

$$\left(1 - \frac{1}{n}\right)^{2n \ln n} \leq e^{-2 \ln n} = \frac{1}{n^2}.$$

By the union bound, the probability that any coupon type is not found is at most $1/n$.

In order to complete a Hamiltonian path to a cycle the path must close, which it does at each step with probability $1/n$. Hence the probability that the path does not become a cycle within the next $n \ln n$ iterations is

$$\left(1 - \frac{1}{n}\right)^{n \ln n} \leq e^{-\ln n} = \frac{1}{n}.$$

Thus we have shown that

$$\Pr(\mathcal{E}_1) \leq \frac{2}{n}.$$

Next we bound $\Pr(\mathcal{E}_2)$, the probability that an unused-edges list is empty in the first $3n \ln n$ iterations. We consider two subevents as follows.

$\mathcal{E}_{2a}$: At least $9 \ln n$ edges were removed from the unused-edges list of at least one vertex in the first $3n \ln n$ iterations of the loop.

$\mathcal{E}_{2b}$: At least one vertex had fewer than $10 \ln n$ edges initially in its unused-edges list.

For $\mathcal{E}_2$ to occur, either $\mathcal{E}_{2a}$ or $\mathcal{E}_{2b}$ must occur. Hence

$$\Pr(\mathcal{E}_2) \leq \Pr(\mathcal{E}_{2a}) + \Pr(\mathcal{E}_{2b}).$$

Let us first bound $\Pr(\mathcal{E}_{2a})$. Exactly one edge is used in each iteration of the loop. From the proof of Lemma 5.16 we have that, at each iteration, the probability that a given vertex v is the head of the path is $1/n$, independently at each step. Hence the number of times X that v is the head during the first $3n \ln n$ steps is a binomial random variable $B(3n \ln n, 1/n)$, and this dominates the number of edges taken from v's unused-edges list.

Using the Chernoff bound of Eqn. (4.1) with $\delta = 2$ and $\mu = 3 \ln n$ for the binomial random variable $B(3n \ln n, 1/n)$, we have

$$\Pr(X \geq 9 \ln n) \leq \left(\frac{e^2}{27}\right)^{3 \ln n} \leq \frac{1}{n^2}.$$

By taking a union bound over all vertices, we find $\Pr(\mathcal{E}_{2a}) \leq 1/n$.

Next we bound $\Pr(\mathcal{E}_{2b})$. The expected number of edges Y initially in a vertex's unused-edges list is at least $(n-1)q \geq (20(n-1) \ln n)/n \geq 19 \ln n$ for sufficiently large n. Using Chernoff bounds again (Eqn. (4.5)), the probability that any vertex initially has $10 \ln n$ edges or fewer on its list is at most

$$\Pr(Y \leq 10 \ln n) \leq e^{-(19 \ln n)(9/19)^2/2} \leq \frac{1}{n^2},$$

and by the union bound the probability that any vertex has too few adjacent edges is at most $1/n$. Thus,

$$\Pr(\mathcal{E}_{2b}) \leq \frac{1}{n}$$

and hence

$$\Pr(\mathcal{E}_2) \leq \frac{2}{n}.$$

In total, the probability that the algorithm fails to find a Hamiltonian cycle in $3n \ln n$ iterations is bounded by

$$\Pr(\mathcal{E}_1) + \Pr(\mathcal{E}_2) \leq \frac{4}{n}.$$ ■

We did not make an effort to optimize the constants in the proof. There is, however, a clear trade-off; with more edges, one could achieve a lower probability of failure.

We are left with showing how our algorithm can be applied to graphs in $G_{n,p}$. We show that, as long as p is known, we can partition the edges of the graph into edge lists that satisfy the requirements of Theorem 5.17.

Corollary 5.18: *By initializing edges on the unused-edges lists appropriately, Algorithm 5.2 will find a Hamiltonian cycle on a graph chosen randomly from $G_{n,p}$ with probability $1 - O(1/n)$ whenever $p \geq (40 \ln n)/n$.*

Proof: We partition the edges of our input graph from $G_{n,p}$ as follows. Let $q \in [0, 1]$ be such that $p = 2q - q^2$. Consider any edge (u, v) in the input graph. We execute exactly one of the following three possibilities: with probability $q(1 - q)/(2q - q^2)$ we place the edge on u's unused-edges list but not on v's; with probability $q(1 - q)/(2q - q^2)$ we initially place the edge on v's unused-edges list but not on u's; and with the remaining probability $q^2/(2q - q^2)$ the edge is placed on both unused-edges lists.

Now, for any possible edge (u, v), the probability that it is initially placed in the unused-edges list for v is

$$p\left(\frac{q(1-q)}{2q-q^2} + \frac{q^2}{2q-q^2}\right) = q.$$

Moreover, the probability that an edge (u, v) is initially placed on the unused-edges list for both u and v is $pq^2/(2q - q^2) = q^2$, so these two placements are independent events. Since each edge (u, v) is treated independently, this partitioning fulfills the requirements of Theorem 5.17 provided the resulting q is at least $20 \ln n/n$. When $p \geq (40 \ln n)/n$ we have $q \geq p/2 \geq (20 \ln n)/n$, and the result follows. ■

In Exercise 5.27, we consider how to use Algorithm 5.2 even in the case where p is not known in advance, so that the edge lists must be initialized without knowledge of p.

5.7. Exercises

Exercise 5.1: For what values of n is $(1 + 1/n)^n$ within 1% of e? Within 0.0001% of e? Similarly, for what values of n is $(1 - 1/n)^n$ within 1% of $1/e$? Within 0.0001%?

Exercise 5.2: Suppose that Social Security numbers were issued uniformly at random, with replacement. That is, your Social Security number would consist of just nine randomly generated digits, and no check would be made to ensure that the same number was not issued twice. Sometimes, the last four digits of a Social Security number are used as a password. How many people would you need to have in a room before it was more likely than not that two had the same last four digits? How many numbers could be issued before it would be more likely than not that there is a duplicate number? How would you answer these two questions if Social Security numbers had 13 digits? Try to give exact numerical answers.

Exercise 5.3: Suppose that balls are thrown randomly into n bins. Show, for some constant c_1, that if there are $c_1\sqrt{n}$ balls then the probability that no two land in the same bin is at most $1/e$. Similarly, show for some constant c_2 (and sufficiently large n) that, if there are $c_2\sqrt{n}$ balls, then the probability that no two land in the same bin is at least $1/2$. Make these constants as close to optimal as possible. *Hint:* You may want to use the facts that

$$e^{-x} \geq 1 - x$$

and

$$e^{-x-x^2} \leq 1 - x \quad \text{for} \quad x \leq \frac{1}{2}.$$

Exercise 5.4: In a lecture hall containing 100 people, you consider whether or not there are three people in the room who share the same birthday. Explain how to calculate this probability exactly, using the same assumptions as in our previous analysis.

Exercise 5.5: Use the moment generating function of the Poisson distribution to compute the second moment and the variance of the distribution.

Exercise 5.6: Let X be a Poisson random variable with mean μ, representing the number of errors on a page of this book. Each error is independently a grammatical error with probability p and a spelling error with probability $1 - p$. If Y and Z are random variables representing the number of grammatical and spelling errors (respectively) on a page of this book, prove that Y and Z are Poisson random variables with means μp and $\mu(1 - p)$, respectively. Also, prove that Y and Z are independent.

Exercise 5.7: Use the Taylor expansion

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$$

to prove that, for any x with $|x| \leq 1$,

$$e^x(1 - x^2) \leq 1 + x \leq e^x.$$

Exercise 5.8: Suppose that n balls are thrown independently and uniformly at random into n bins.

(a) Find the conditional probability that bin 1 has one ball given that exactly one ball fell into the first three bins.
(b) Find the conditional expectation of the number of balls in bin 1 under the condition that bin 2 received no balls.
(c) Write an expression for the probability that bin 1 receives more balls than bin 2.

Exercise 5.9: Our analysis of Bucket sort in Section 5.2.2 assumed that n elements were chosen independently and uniformly at random from the range $[0, 2^k)$. Suppose instead that n elements are chosen independently from the range $[0, 2^k)$ according to a distribution with the property that any number $x \in [0, 2^k)$ is chosen with probability at most $a/2^k$ for some fixed constant $a > 0$. Show that, under these conditions, Bucket sort still requires linear expected time.

Exercise 5.10: Consider the probability that every bin receives exactly one ball when n balls are thrown randomly into n bins.

(a) Give an upper bound on this probability using the Poisson approximation.
(b) Determine the *exact* probability of this event.
(c) Show that these two probabilities differ by a multiplicative factor that equals the probability that a Poisson random variable with parameter n takes on the value n. Explain why this is implied by Theorem 5.6.

Exercise 5.11: Consider throwing m balls into n bins, and for convenience let the bins be numbered from 0 to $n-1$. We say there is a *k-gap* starting at bin i if bins $i, i+1, \ldots, i+k-1$ are all empty.

(a) Determine the expected number of k-gaps.
(b) Prove a Chernoff-like bound for the number of k-gaps. (*Hint:* If you let $X_i = 1$ when there is a k-gap starting at bin i, then there are dependencies between X_i and X_{i+1}; to avoid these dependencies, you might consider X_i and X_{i+k}.)

Exercise 5.12: The following problem models a simple distributed system wherein agents contend for resources but "back off" in the face of contention. Balls represent agents, and bins represent resources.

The system evolves over rounds. Every round, balls are thrown independently and uniformly at random into n bins. Any ball that lands in a bin by itself is *served* and removed from consideration. The remaining balls are thrown again in the next round. We begin with n balls in the first round, and we finish when every ball is served.

(a) If there are b balls at the start of a round, what is the expected number of balls at the start of the next round?
(b) Suppose that every round the number of balls served was exactly the expected number of balls to be served. Show that all the balls would be served in $O(\log \log n)$ rounds. (*Hint:* If x_j is the expected number of balls left after j rounds, show and use that $x_{j+1} \leq x_j^2/n$.)

Exercise 5.13: Suppose that we vary the balls-and-bins process as follows. For convenience let the bins be numbered from 0 to $n-1$. There are $\log_2 n$ players. Each player randomly chooses a starting location ℓ uniformly from $[0, n-1]$ and then places one ball in each of the bins numbered $\ell \bmod n, \ell+1 \bmod n, \ldots, \ell + n/\log_2 n - 1 \bmod n$. Argue that the maximum load in this case is only $O(\log\log n/\log\log\log n)$ with probability that approaches 1 as $n \to \infty$.

Exercise 5.14: We prove that if Z is a Poisson random variable of mean μ, where $\mu \geq 1$ is an integer, then $\Pr(Z \geq \mu) \geq 1/2$.

(a) Show that $\Pr(Z = \mu + h) \geq \Pr(Z = \mu - h - 1)$ for $0 \leq h \leq \mu - 1$.
(b) Using part (a), argue that $\Pr(Z \geq \mu) \geq 1/2$.
(c) Show that $\Pr(Z \leq \mu) \leq 1/2$ for all integers μ from 1 to 10 by explicitly performing the calculation. (This is in fact true for all integers $\mu \geq 1$, but it is more difficult to prove.)

Exercise 5.15: **(a)** In Theorem 5.7 we showed that, for any nonnegative functions f,

$$\mathbf{E}\left[f\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right)\right] \geq \mathbf{E}\left[f\left(X_1^{(m)}, \ldots, X_n^{(m)}\right)\right] \Pr\left(\sum Y_i^{(m)} = m\right).$$

Prove that if $\mathbf{E}[f(X_1^{(m)}, \ldots, X_n^{(m)})]$ is monotonically increasing in m, then

$$\mathbf{E}\left[f\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right)\right] \geq \mathbf{E}\left[f\left(X_1^{(m)}, \ldots, X_n^{(m)}\right)\right] \Pr\left(\sum Y_i^{(m)} \geq m\right),$$

again under the condition that f is nonnegative. Make a similar statement for the case when $\mathbf{E}[f(X_1^{(m)}, \ldots, X_n^{(m)})]$ is monotonically decreasing in m.

(b) Using part (a) and Exercise 5.14, prove Theorem 5.10 for the case that $\mathbf{E}[f(X_1^{(m)}, \ldots, X_n^{(m)})]$ is monotonically increasing.

Exercise 5.16: We consider another way to obtain Chernoff-like bounds in the setting of balls and bins without using Theorem 5.7. Consider n balls thrown randomly into n bins. Let $X_i = 1$ if the ith bin is empty and 0 otherwise. Let $X = \sum_{i=1}^{n} X_i$. Let Y_i, $i = 1, \ldots, n$, be independent Bernoulli random variables that are 1 with probability $p = (1 - 1/n)^n$. Let $Y = \sum_{i=1}^{n} Y_i$.

(a) Show that $\mathbf{E}[X_1 X_2 \cdots X_k] \leq \mathbf{E}[Y_1 Y_2 \cdots Y_k]$ for any $k \geq 1$.
(b) Show that $\mathbf{E}[e^{tX}] \leq \mathbf{E}[e^{tY}]$ for all $t \geq 0$. (*Hint:* Use the expansion for e^x and compare $\mathbf{E}[X^k]$ to $\mathbf{E}[Y^k]$.)
(c) Derive a Chernoff bound for $\Pr(X \geq (1+\delta)\mathbf{E}[X])$.

Exercise 5.17: Let G be a random graph generated using the $G_{n,p}$ model.

(a) A *clique* of k vertices in a graph is a subset of k vertices such that all $\binom{k}{2}$ edges between these vertices lie in the graph. For what value of p, as a function of n, is the expected number of cliques of five vertices in G equal to 1?
(b) A $K_{3,3}$ graph is a complete bipartite graph with three vertices on each side. In other words, it is a graph with six vertices and nine edges; the six distinct vertices are arranged in two groups of three, and the nine edges connect each of the nine pairs

of vertices with one vertex in each group. For what value of p, as a function of n, is the expected number of $K_{3,3}$ subgraphs of G equal to 1?

(c) For what value of p, as a function of n, is the expected number of Hamiltonian cycles in the graph equal to 1?

Exercise 5.18: Theorem 5.7 shows that any event that occurs with small probability in the balls-and-bins setting where the number of balls in each bin is an independent Poisson random variable also occurs with small probability in the standard balls-and-bins model. Prove a similar statement for random graphs: Every event that happens with small probability in the $G_{n,p}$ model also happens with small probability in the $G_{n,N}$ model for $N = \binom{n}{2} p$.

Exercise 5.19: An undirected graph on n vertices is *disconnected* if there exists a set of $k < n$ vertices such that there is no edge between this set and the rest of the graph. Otherwise, the graph is said to be *connected.* Show that there exists a constant c such that if $N \geq cn \log n$ then, with probability $1 - o(1)$, a graph randomly chosen from $G_{n,N}$ is connected.

Exercise 5.20: Prove Theorem 5.15.

Exercise 5.21: (a) Let $f(n)$ be the expected number of random edges that must be added before an empty undirected graph with n vertices becomes connected. (Connectedness is defined in Exercise 5.19.) That is, suppose that we start with a graph on n vertices with zero edges and then repeatedly add an edge, chosen uniformly at random from all edges not currently in the graph, until the graph becomes connected. If X_n represents the number of edges added, then $f(n) = \mathbf{E}[X_n]$.

Write a program to estimate $f(n)$ for a given value of n. Your program should track the connected components of the graph as you add edges until the graph becomes connected. You will probably want to use a disjoint set data structure, a topic covered in standard undergraduate algorithms texts. You should try $n = 100, 200, 300, 400, 500, 600, 700, 800, 900$, and 1000. Repeat each experiment 100 times, and for each value of n compute the average number of edges needed. Based on your experiments, suggest a function $h(n)$ that you think is a good estimate for $f(n)$.

(b) Modify your program for the problem in part (a) so that it also keeps track of isolated vertices. Let $g(n)$ be the expected number of edges added before there are no more isolated vertices. What seems to be the relationship between $f(n)$ and $g(n)$?

Exercise 5.22: In hashing with *open addressing,* the hash table is implemented as an array and there are no linked lists or chaining. Each entry in the array either contains one hashed item or is empty. The hash function defines, for each key k, a *probe sequence* $h(k, 0), h(k, 1), \ldots$ of table locations. To insert the key k, we first examine the sequence of table locations in the order defined by the key's probe sequence until we find an empty location; then we insert the item at that position. When searching for an item in the hash table, we examine the sequence of table locations in the order defined by the key's probe sequence until either the item is found or we have found an empty location

in the sequence. If an empty location is found, this means the item is not present in the table.

An open-address hash table with $2n$ entries is used to store n items. Assume that the table location $h(k, j)$ is uniform over the $2n$ possible table locations and that all $h(k, j)$ are independent.

(a) Show that, under these conditions, the probability of an insertion requiring more than k probes is at most 2^{-k}.

(b) Show that, for $i = 1, 2, \ldots, n$, the probability that the ith insertion requires more than $2 \log n$ probes is at most $1/n^2$.

Let the random variable X_i denote the number of probes required by the ith insertion. You have shown in part (b) that $\Pr(X_i > 2 \log n) \leq 1/n^2$. Let the random variable $X = \max_{1 \leq i \leq n} X_i$ denote the maximum number of probes required by any of the n insertions.

(c) Show that $\Pr(X > 2 \log n) \leq 1/n$.

(d) Show that the expected length of the longest probe sequence is $\mathbf{E}[X] = O(\log n)$.

Exercise 5.23: Bloom filters can be used to estimate set differences. Suppose you have a set X and I have a set Y, both with n elements. For example, the sets might represent our 100 favorite songs. We both create Bloom filters of our sets, using the same number of bits m and the same k hash functions. Determine the expected number of bits where our Bloom filters differ as a function of m, n, k, and $|X \cap Y|$. Explain how this could be used as a tool to find people with the same taste in music more easily than comparing lists of songs directly.

Exercise 5.24: Suppose that we wanted to extend Bloom filters to allow deletions as well as insertions of items into the underlying set. We could modify the Bloom filter to be an array of counters instead of an array of bits. Each time an item is inserted into a Bloom filter, the counters given by the hashes of the item are increased by one. To delete an item, one can simply decrement the counters. To keep space small, the counters should be a fixed length, such as 4 bits.

Explain how errors can arise when using fixed-length counters. Assuming a setting where one has at most n elements in the set at any time, m counters, k hash functions, and counters with b bits, explain how to bound the probability that an error occurs over the course of t insertions or deletions.

Exercise 5.25: Suppose that you built a Bloom filter for a dictionary of words with $m = 2^b$ bits. A co-worker building an application wants to use your Bloom filter but has only 2^{b-1} bits available. Explain how your colleague can use your Bloom filter to avoid rebuilding a new Bloom filter using the original dictionary of words.

Exercise 5.26: For the leader election problem alluded to in Section 5.5.4, we have n users, each with an identifier. The hash function takes as input the identifier and outputs a b-bit hash value, and we assume that these values are independent and uniformly distributed. Each user hashes its identifier, and the leader is the user with the smallest

hash value. Give lower and upper bounds on the number of bits b necessary to ensure that a unique leader is successfully chosen with probability p. Make your bounds as tight as possible.

Exercise 5.27: Consider Algorithm 5.2, the modified algorithm for finding Hamiltonian cycles. We have shown that the algorithm can be applied to find a Hamiltonian cycle with high probability in a graph chosen randomly from $G_{n,p}$, when p is known and sufficiently large, by initially placing edges in the edge lists appropriately. Argue that the algorithm can similarly be applied to find a Hamiltonian cycle with high probability on a graph chosen randomly from $G_{n,N}$ when $N = c_1 n \ln n$ for a suitably large constant c_1. Argue also that the modified algorithm can be applied even when p is not known in advance as long as p is at least $c_2 \ln n/n$ for a suitably large constant c_2.

5.8. An Exploratory Assignment

Part of the research process in random processes is first to understand what is going on at a high level and then to use this understanding in order to develop formal mathematical proofs. In this assignment, you will be given several variations on a basic random process. To gain insight, you should perform experiments based on writing code to simulate the processes. (The code should be very short, a few pages at most.) After the experiments, you should use the results of the simulations to guide you to make conjectures and prove statements about the processes. You can apply what you have learned up to this point, including probabilistic bounds and analysis of balls-and-bins problems.

Consider a complete binary tree with $N = 2^n - 1$ nodes. Here n is the depth of the tree. Initially, all nodes are *unmarked.* Over time, via processes that we shall describe, nodes becomes *marked.*

All of the processes share the same basic form. We can think of the nodes as having unique identifying numbers in the range of $[1, N]$. Each unit of time, I *send* you the identifier of a node. When you receive a sent node, you mark it. Also, you invoke the following *marking rule, which takes effect before I send out the next node.*

- If a node and its sibling are marked, its parent is marked.
- If a node and its parent are marked, the other sibling is marked.

The marking rule is applied *recursively as much as possible* before the next node is sent. For example, in Figure 5.3, the marked nodes are filled in. The arrival of the node labeled by an X will allow you to mark the remainder of the nodes, as you apply the marking rule first up and then down the tree. Keep in mind that you always apply the marking rule as much as possible.

Now let us consider the different ways in which I might be sending you the nodes.

Process 1: Each unit of time, I send the identifier of a node chosen *independently and uniformly at random from all of the N nodes.* Note that I might send you a node that is already marked, and in fact I may send a useless node that I have already sent.

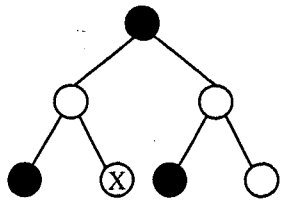

Figure 5.3: The arrival of X causes all other nodes to be marked.

Process 2: Each unit of time I send the identifier of a node chosen uniformly at random from those nodes *that I have not yet sent.* Again, a node that has already been marked might arrive, but each node will be sent at most once.

Process 3: Each unit of time I send the identifier of a node chosen uniformly at random from those nodes *that you have not yet marked.*

We want to determine how many time steps are needed before all the nodes are marked for each of these processes. Begin by writing programs to simulate the sending processes and the marking rule. Run each process ten times for each value of n in the range [10, 20]. Present the data from your experiments in a clear, easy-to-read fashion and explain your data suitably. A tip: You may find it useful to have your program print out the last node that was sent before the tree became completely marked.

1. For the first process, prove that the expected number of nodes sent is $\Omega(N \log N)$. How well does this match your simulations?
2. For the second process, you should find that almost all N nodes must be sent before the tree is marked. Show that, with constant probability, at least $N - 2\sqrt{N}$ nodes must be sent.
3. The behavior of the third process might seem a bit unusual. Explain it with a proof.

After answering these questions, you may wish to consider other facts you could prove about these processes.