CS 70 Discrete Mathematics for CS
Fall 2003 Wagner

Lecture 19

Two Killer Applications

In this lecture, we will see two "killer apps" of elementary probability in Computer Science.

1. Suppose a hash function distributes keys evenly over a table of size n. How many (randomly chosen) keys can we hash before the probability of a collision exceeds (say) $\frac{1}{2}$?
2. Consider the following simple load balancing scenario. We are given m jobs and n machines; we allocate each job to a machine uniformly at random and independently of all other jobs. What is a likely value for the maximum load on any machine?

As we shall see, both of these questions can be tackled by an analysis of the balls-and-bins probability space which we have already encountered.

Application 1: Hash functions

As you may recall, a hash table is a data structure that supports the storage of sets of keys from a (large) universe U (say, the names of all 250m people in the US). The operations supported are ADDing a key to the set, DELETEing a key from the set, and testing MEMBERship of a key in the set. The hash function h maps U to a table T of modest size. To ADD a key x to our set, we evaluate $h(x)$ (i.e., apply the hash function to the key) and store x at the location $h(x)$ in the table T. All keys in our set that are mapped to the same table location are stored in a simple linked list. The operations DELETE and MEMBER are implemented in similar fashion, by evaluating $h(x)$ and searching the linked list at $h(x)$. Note that the efficiency of a hash function depends on having only few collisions — i.e., keys that map to the same location. This is because the search time for DELETE and MEMBER operations is proportional to the length of the corresponding linked list.

The question we are interested in here is the following: suppose our hash table T has size n, and that our hash function h distributes U evenly over T.[1] Assume that the keys we want to store are chosen uniformly at random and independently from the universe U. What is the largest number, m, of keys we can store before the probability of a collision reaches $\frac{1}{2}$?

Let's begin by seeing how this problem can be put into the balls and bins framework. The balls will be the m keys to be stored, and the bins will be the n locations in the hash table T. Since the keys are chosen uniformly and independently from U, and since the hash function distributes keys evenly over the table, we can see each key (ball) as choosing a hash table location (bin) uniformly and independently from T. Thus the probability space corresponding to this hashing experiment is exactly the same as the balls and bins space.

We are interested in the event A that there is no collision, or equivalently, that all m balls land in different bins. Clearly $\Pr[A]$ will decrease as m increases (with n fixed). Our goal is to find the largest value of m

[1] I.e., $|U| = \alpha n$ (the size of U is an integer multiple α of the size of T), and for each $y \in T$, the number of keys $x \in U$ for which $h(x) = y$ is exactly α.

such that $\Pr[A]$ remains above $\frac{1}{2}$. [Note: Really we are looking at different sample spaces here, one for each value of m. So it would be more correct to write $\Pr_m$ rather than just Pr, to make clear which sample space we are talking about. However, we will omit this detail.]

Let's fix the value of m and try to compute $\Pr[A]$. Since our probability space is uniform (each outcome has probability $\frac{1}{n^m}$), it's enough just to count the number of outcomes in A. In how many ways can we arrange m balls in n bins so that no bin contains more than one ball? Well, there are n places to put the first ball, then $n-1$ remaining places for the second ball (since it cannot go in the same bin as the first), $n-2$ places for the third ball, and so on. Thus the total number of such arrangements is

$$n \times (n-1) \times (n-2) \times \cdots \times (n-m+2) \times (n-m+1).$$

This formula is valid as long as $m \leq n$: if $m > n$ then clearly the answer is zero. From now on, we'll assume that $m \leq n$.

Now we can calculate the probability of no collision:

$$\begin{aligned}
\Pr[A] &= \frac{n(n-1)(n-2)\ldots(n-m+1)}{n^m} \\
&= \frac{n}{n} \times \frac{n-1}{n} \times \frac{n-2}{n} \times \cdots \times \frac{n-m+1}{n} \\
&= \left(1-\frac{1}{n}\right) \times \left(1-\frac{2}{n}\right) \times \cdots \times \left(1-\frac{m-1}{n}\right). \qquad (1)
\end{aligned}$$

Before going on, let's pause to observe that we could compute $\Pr[A]$ in a different way, as follows. View the probability space as a sequence of choices, one for each ball. For $1 \leq i \leq m$, let A_i be the event that the ith ball lands in a different bin from balls $1, 2, \ldots, i-1$. Then

$$\begin{aligned}
\Pr[A] = \Pr[\textstyle\bigcap_{i=1}^{n} A_i] &= \Pr[A_1] \times \Pr[A_2|A_1] \times \Pr[A_3|A_1 \cap A_2] \times \cdots \times \Pr[A_m|\textstyle\bigcap_{i=1}^{m-1} A_i] \\
&= 1 \times \frac{n-1}{n} \times \frac{n-2}{n} \times \cdots \times \frac{n-m+1}{n} \\
&= \left(1-\frac{1}{n}\right) \times \left(1-\frac{2}{n}\right) \times \cdots \times \left(1-\frac{m-1}{n}\right).
\end{aligned}$$

Fortunately, we get the same answer as before! [You should make sure you see how we obtained the conditional probabilities in the second line above. For example, $\Pr[A_3|A_1 \cap A_2]$ is the probability that the third ball lands in a different bin from the first two balls, *given that* those two balls also landed in different bins. This means that the third ball has $n-2$ possible bin choices out of a total of n.]

Essentially, we are now done with our problem: equation (1) gives an exact formula for the probability of no collision when m keys are hashed. All we need to do now is plug values $m = 1, 2, 3, \ldots$ into (1) until we find that $\Pr[A]$ drops below $\frac{1}{2}$. The corresponding value of m (minus 1) is what we want.

But this is not really satisfactory: it would be much more useful to have a formula that gives the "critical" value of m directly, rather than having to compute $\Pr[A]$ for $m = 1, 2, 3, \ldots$. Note that we would have to do this computation separately for each different value of n we are interested in: i.e., whenever we change the size of our hash table.

So what remains is to "turn equation (1) around", so that it tells us the value of m at which $\Pr[A]$ drops below $\frac{1}{2}$. To do this, let's take logs: this is a good thing to do because it turns the product into a sum, which is easier to handle. We get

$$\ln(\Pr[A]) = \ln\left(1-\frac{1}{n}\right) + \ln\left(1-\frac{2}{n}\right) + \cdots + \ln\left(1-\frac{m-1}{n}\right), \qquad (2)$$

where "ln" denotes natural (base e) logarithm. Now we can make use of a standard approximation for logarithms: namely, if x is small then $\ln(1-x) \approx -x$. This comes from the Taylor series expansion

$$\ln(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \ldots.$$

So by replacing $\ln(1-x)$ by $-x$ we are making an error of at most $(\frac{x^2}{2} + \frac{x^3}{3} + \cdots)$, which is at most $2x^2$ when $x \leq \frac{1}{2}$. In other words, we have

$$-x \geq \ln(1-x) \geq -x - 2x^2.$$

And if x is small then the error term $2x^2$ will be much smaller than the main term $-x$. Rather than carry around the error term $2x^2$ everywhere, in what follows we'll just write $\ln(1-x) \approx -x$, secure in the knowledge that we could make this approximation precise if necessary.

Now let's plug this approximation into equation (2):

$$\begin{aligned} \ln(\Pr[A]) &\approx -\frac{1}{n} - \frac{2}{n} - \frac{3}{n} - \ldots - \frac{m-1}{n} \\ &= -\frac{1}{n}\sum_{i=1}^{m-1} i \\ &= -\frac{m(m-1)}{2n} \\ &\approx -\frac{m^2}{2n}. \end{aligned} \tag{3}$$

Note that we've used the approximation for $\ln(1-x)$ with $x = \frac{1}{n}, \frac{2}{n}, \frac{3}{n}, \ldots, \frac{m-1}{n}$. So our approximation should be good provided all these are small, i.e., provided n is fairly big and m is quite a bit smaller than n. Once we're done, we'll see that the approximation is actually pretty good even for modest sizes of n.

Now we can undo the logs in (3) to get our expression for $\Pr[A]$:

$$\Pr[A] \approx \mathrm{e}^{-\frac{m^2}{2n}}.$$

The final step is to figure out for what value of m this probability becomes $\frac{1}{2}$. So we want the largest m such that $\mathrm{e}^{-\frac{m^2}{2n}} \geq \frac{1}{2}$. This means we must have

$$-\frac{m^2}{2n} \quad \geq \quad \ln(\tfrac{1}{2}) = -\ln 2, \tag{4}$$

or equivalently

$$m \quad \leq \quad \sqrt{(2\ln 2)n} \approx 1.177\sqrt{n}.$$

So the bottom line is that we can hash approximately $\underline{m = \lfloor 1.177\sqrt{n} \rfloor \text{ keys}}$ before the probability of a collision reaches $\frac{1}{2}$.

Recall that our calculation was only approximate; so we should go back and get a feel for how much error we made. We can do this by using equation (1) to compute the exact value $m = m_0$ at which $\Pr[A]$ drops below $\frac{1}{2}$, for a few sample values of n. Then we can compare these values with our estimate $m = 1.177\sqrt{n}$.

n	10	20	50	100	200	365	500	1000	10^4	10^5	10^6
$1.177\sqrt{n}$	3.7	5.3	8.3	11.8	16.6	22.5	26.3	37.3	118	372	1177
exact m_0	4	5	8	12	16	22	26	37	118	372	1177

From the table, we see that our approximation is very good even for small values of n. When n is large, the error in the approximation becomes negligible.

Why $\frac{1}{2}$?

Our hashing question asked when the probability of a collision rises to $\frac{1}{2}$. Is there anything special about $\frac{1}{2}$? Not at all. What we did was to (approximately) compute $\Pr[A]$ (the probability of no collision) as a function of m, and then find the largest value of m for which our estimate is smaller than $\frac{1}{2}$. If instead we were interested in when the collision probability reaches (say) 0.95 ($=95\%$), we would just replace $\frac{1}{2}$ by 0.05 in equation (4). If you work through the last piece of algebra again, you'll see that this gives us the critical value $m=\sqrt{(2\ln 20)n}\approx 2.45\sqrt{n}$. So no matter what "confidence" probability we specify, our critical value of m will always be $c\sqrt{n}$ for some constant c (which depends on the confidence probability).

Birthdays

Here is a famous problem, often referred to as the "birthday paradox" (though it is not really a paradox at all). You are having some friends over for a party. How many guests do you need to invite in order to have a good chance (say, at least $\frac{1}{2}$) that two of them have the same birthday? This is exactly our hashing collision problem, with $n=365$ (the number of days in the year).[2] From the above table, we see that 23 people are enough (why?). If you wanted to play it safer and have a higher probability (say, 95%), you would need 47 people (you should check this).

Application 2: Load balancing

One of the most pressing practical issues in distributed computing is how to spread the workload in a distributed system among its processors. This is a huge question, still very much unsolved in general. Here we investigate an extremely simple scenario that is both fundamental in its own right and also establishes a baseline against which more sophisticated methods should be judged.

Suppose we have m identical jobs and n identical processors. Our task is to assign the jobs to the processors in such a way that no processor is too heavily loaded. Of course, there is a simple optimal solution here: just divide up the jobs as evenly as possible, so that each processor receives either $\lceil \frac{m}{n} \rceil$ or $\lfloor \frac{m}{n} \rfloor$ jobs. However, this solution requires a lot of centralized control, and/or a lot of communication: the workload has to be balanced evenly either by a powerful centralized scheduler that talks to all the processors, or by the exchange of many messages between jobs and processors. This kind of operation is very costly in most distributed systems. The question therefore is: What can we do with little or no overhead in scheduling and communication cost?

The first idea that comes to mind here is... balls and bins! I.e., each job simply selects a processor uniformly at random and independently of all others, and goes to that processor. (Make sure you believe that the probability space for this experiment is the same as the one for balls and bins.) This scheme requires no communication. However, presumably it won't in general achieve an optimal balancing of the load. Let X be the maximum loading of any processor under our randomized scheme. Note that X isn't a fixed number: its value depends on the outcome of our balls and bins experiment.[3] So, as designers or users of this load balancing scheme, what should be our question?

[2] Well, not *exactly*: We are ignoring leap years, and more seriously, we are assuming that birthdays are independently and uniformly distributed throughout the year. In fact, the birth rate fluctuates through the year, with peaks (for example) about nine months after winter holidays.

[3] In fact, X is called a random variable: we'll define this properly in the next lecture.

Q: Find the smallest value k such that

$$\Pr[X \geq k] \leq \tfrac{1}{2}.$$

If we have such a value k, then we'll know that, with good probability (at least $\frac{1}{2}$), the maximum load on any processor in our system won't exceed k. This will give us a good idea about the performance of the system. Of course, as with our hashing application, there's nothing special about the value $\frac{1}{2}$: we're just using this for illustration. As you can check later, essentially the same analysis can be used to find k such that $\Pr[X \geq k] \leq 0.05$ (i.e., 95% confidence), or any other value we like. Indeed, we can even find the k's for several different confidence levels and thus build up a more detailed picture of the behavior of the scheme. To simplify our problem, we'll also assume from now on that $m = n$ (i.e., the number of jobs is the same as the number of processors). With a bit more work, we could generalize our analysis to other values of m.

From Application 1 we know that we get collisions already when $m \approx 1.177\sqrt{n}$. So when $m = n$ the maximum load will certainly be larger than 1 (with good probability). But how large will it be? If we try to analyze the maximum load directly, we run into the problem that it depends on the number of jobs at *every* processor (or equivalently, the number of balls in every bin). Since the load in one bin depends on those in the others, this becomes very tricky. Instead, what we'll do is analyze the load in any *one* bin, say bin 1; this will be fairly easy. Call the load in bin 1 X_1 (another random variable). What we'll do is find k such that

$$\Pr[X_1 \geq k] \leq \tfrac{1}{2n}. \tag{5}$$

Since all the bins are identical, we will then know that, for the same k,

$$\Pr[X_i \geq k] \leq \tfrac{1}{2n} \qquad \text{for } i = 1, 2, \ldots, n,$$

where X_i is the load in bin i. But now, since the event $X \geq k$ is exactly the union of the events $X_i \geq k$ (why?), we can use the "Union Bound" from the previous lecture:

$$\begin{aligned}
\Pr[X \geq k] &= \Pr[\textstyle\bigcup_{i=1}^{n} (X_i \geq k)] \\
&\leq \textstyle\sum_{i=1}^{n} \Pr[X_i \geq k] \\
&= n \times \tfrac{1}{2n} \\
&= \tfrac{1}{2}.
\end{aligned}$$

It's worth standing back to notice what we did here: we wanted to conclude that $\Pr[A] \leq \frac{1}{2}$, where A is the event that $X \geq k$. We couldn't analyze A directly, but we knew that $A = \bigcup_{i=1}^{n} A_i$, for much simpler events A_i (namely, A_i is the event that $X_i \geq k$). Since there are n events A_i, and all have the same probability, it is enough for us to show that $\Pr[A_i] \leq \frac{1}{2n}$; the union bound then guarantees that $\Pr[A] \leq \frac{1}{2}$. This kind of reasoning is very common in applications of probability in Computer Science.

Now let's get back to our problem. Recall that we've reduced our task to finding k such that

$$\Pr[X_1 \geq k] \leq \tfrac{1}{2n},$$

where X_1 is the load in bin 1. It's not hard to write down an exact expression for $\Pr[X_1 = j]$, the probability that the load in bin 1 is precisely j:

$$\Pr[X_1 = j] = \binom{n}{j} \left(\tfrac{1}{n}\right)^j \left(1 - \tfrac{1}{n}\right)^{n-j}. \tag{6}$$

This can be seen by viewing each ball as a biased coin toss: Heads corresponds to the ball landing in bin 1, Tails to all other outcomes. So the Heads probability is $\frac{1}{n}$; and all coin tosses are (mutually) independent. As we saw in earlier lectures, (6) gives the probability of exactly j Heads in n tosses.

Thus we have

$$\Pr[X_1 \geq k] = \sum_{j=k}^{n} \Pr[X_1 = j] = \sum_{j=k}^{n} \binom{n}{j} \left(\tfrac{1}{n}\right)^j \left(1 - \tfrac{1}{n}\right)^{n-j}. \tag{7}$$

Now in some sense we are done: we could try plugging values $k = 1, 2, \ldots$ into (7) until the probability drops below $\frac{1}{2n}$. However, as in the hashing example, it will be much more useful if we can massage equation (7) into a cleaner form from which we can read off the value of k more directly. To do this, we'll replace the exact equation (7) with an approximation obtained from the union bound: it will turn out that this approximation is pretty good in practice.

Let B denote the event "$X_1 \geq k$", and for each subset $S \subseteq \{1, 2, \ldots, n\}$ of exactly k balls, let B_S denote the event that all balls in S land in bin 1. Clearly $B = \bigcup_S B_S$, because B happens if and only if at least one of the B_S happens. So, using the union bound again, we can write

$$\Pr[X_1 \geq k] \equiv \Pr[B] = \Pr[\textstyle\bigcup_S B_S] \leq \displaystyle\sum_S \Pr[B_S]. \tag{8}$$

What is $\Pr[B_S]$? Well, for any S, this is just the probability that some particular set of k balls land in bin 1, which is just $(\frac{1}{n})^k$. And the number of such sets S is $\binom{n}{k}$. So the sum in (8) can be written as

$$\Pr[X_1 \geq k] \leq \sum_S \Pr[B_S] = \tbinom{n}{k}(\tfrac{1}{n})^k. \tag{9}$$

This is much simpler to work with than the sum in equation (7).

The one remaining messy thing in equation (9) is the binomial coefficient $\binom{n}{k}$. This we can deal with using the standard approximation[4] $(\frac{n}{k})^k \leq \binom{n}{k} \leq (\frac{ne}{k})^k$, which gives us

$$\Pr[X_1 \geq k] \leq \left(\tfrac{ne}{k}\right)^k \left(\tfrac{1}{n}\right)^k = \left(\tfrac{e}{k}\right)^k \tag{10}$$

Note that, even though we have made a few approximations, inequality (10) is completely valid: all our approximations were "$\leq$", so we always have an *upper* bound on $\Pr[X_1 \geq k]$. [You should go back through all the steps and check this.]

Recall from (5) that our goal is to make the probability in (10) less than $\frac{1}{2n}$. We can ensure this by choosing k so that

$$\left(\tfrac{e}{k}\right)^k \leq \tfrac{1}{2n}. \tag{11}$$

Now we are in good shape: given any value n for the number of jobs/processors, we just need to find the smallest value $k = k_0$ that satisfies inequality (11). We will then know that, with probability at least $\frac{1}{2}$, the maximum load on any processor is at most k_0. The table below shows the values of k_0 for some sample values of n. As an exercise, you are invited to perform the experiment and compare these values of k_0 with what happens in practice.

n	10	20	50	100	500	1000	10^4	10^5	10^6	10^7	10^8	10^{15}
exact k_0	5	6	6	7	8	8	9	10	11	12	13	19
$\ln(2n)$	3.0	3.7	4.6	5.3	6.9	7.6	9.9	12.2	14.5	16.8	19.1	35.2
$\frac{2\ln n}{\ln\ln n}$	5.6	5.4	5.8	6.0	6.8	7.2	8.2	9.4	10.6	11.6	12.6	20

[4] Computer scientists and mathematicians carry around a little bag of tricks for replacing complicated expressions like $\binom{n}{k}$ with simpler approximations. This is just one of these. It isn't too hard to prove the lower bound, i.e., that $\binom{n}{k} \geq (\frac{n}{k})^k$. The upper bound is a bit trickier, and makes use of another approximation for $n!$ known as *Stirling's approximation*, which implies that $k! \geq (\frac{k}{e})^k$. We won't discuss the details here.

Can we come up with a formula for k_0 as a function of n (as we did for the hashing problem)? Well, let's take logs in (11):

$$k(\ln k - 1) \geq \ln(2n). \tag{12}$$

From this, we might guess that $k = \ln(2n)$ is a good value for k_0. Plugging in this value of k makes the left-hand side of (12) equal to $\ln(2n)(\ln\ln(2n) - 1)$, which is certainly bigger than $\ln(2n)$ provided $\ln\ln(2n) \geq 2$, i.e., $n \geq \frac{1}{2}e^{e^2} \approx 810$. So for $n \geq 810$ we can claim that the maximum load is (with probability at least $\frac{1}{2}$) no larger than $\ln(2n)$. The table above plots the values of $\ln(2n)$ for comparison with k_0. As expected, the estimate is quite good for small n, but becomes rather pessimistic when n is large.

For large n we can do better as follows. If we plug the value $k = \frac{\ln n}{\ln\ln n}$ into the left-hand side of (12), it becomes

$$\frac{\ln n}{\ln\ln n}(\ln\ln n - \ln\ln\ln n - 1) = \ln n\left(1 - \frac{\ln\ln\ln n + 1}{\ln\ln n}\right). \tag{13}$$

Now when n is large this is *just barely* smaller than the right-hand side, $\ln(2n)$. Why? — Because the second term inside the parentheses goes to zero as $n \to \infty$,[5] and because $\ln(2n) = \ln n + \ln 2$, which is very close to $\ln n$ when n is large (since $\ln 2$ is a fixed small constant). So we can conclude that, for large values of n, the quantity $\frac{\ln n}{\ln\ln n}$ should be a pretty good estimate of k_0. Actually for this estimate to become good n has to be (literally) astronomically large. For more civilized values of n, we get a better estimate by taking $k = \frac{2\ln n}{\ln\ln n}$. The extra factor of 2 helps to wipe out the lower order terms (i.e., the second term in the parenthesis in (13) and the $\ln 2$) more quickly. The table above also shows the behavior of this estimate for various values of n.

Finally, here is one punchline from Application 2. Let's say the total US population is about 250 million. Suppose we mail 250 million items of junk mail, each one with a random US address. Then (see the above table) with probability at least $\frac{1}{2}$, no one person anywhere will receive more than about a dozen items!

[5] To see this, note that it is of the form $\frac{\ln z + 1}{z}$ where $z = \ln\ln n$, and of course $\frac{\ln z}{z} \to 0$ as $z \to \infty$.