Lecture 16: Markov's inequality, balls & bins

Plan/outline

Last class, we saw some examples of how to find the expected running time of randomized algorithms. Today's goal is to understand: is this good enough? Also, we start looking at *hashing*, using a simple abstraction of throwing balls into bins.

More on expected running time

Recall that the expected running time is, by definition, the *average* running time over all the "random choices" of the algorithm. Now, if the average running time is small, does it mean that when we run the algorithm, the time taken will be small with "high" probability? A priori, this is not very clear, but we now see a simple theorem that formalizes it.

Markov's inequality. Suppose X is a random variable that only takes *non-negative* values, and suppose its expected value is $\mathbb{E}[X]$. Then for any $t \geq 1$, we have $\Pr[X \geq t \cdot \mathbb{E}[X]] \leq \frac{1}{t}$.

The proof of Markov's inequality is very straightforward, in fact just one line, using the definition of the expectation; we skip it here as one can find it easily online.

Intuitively, the inequality says that *most of the time*, a random variable is not much more than its expectation. For instance, setting $t = 4$, we get that the probability that $X > 4\mathbb{E}[X]$ is smaller than $1/4$, or equivalently, $\Pr[X \leq 4\mathbb{E}[X]] \geq 3/4$.

So let us go back to the Quicksort example. We showed that the expected running time is $4n \log n$. So Markov's inequality tells us that with probability at least $3/4$, the running time is at most $16n \log n$.

Boosting probability. While the guarantee above is reasonable, ideally we would wish to have a much larger success probability. If we were to use Markov's inequality, then if we want to say that the running time is $\leq Q$ with probability 0.999, we will have to pick $Q = 4000n \log n$, which is not great. But it turns out that by slightly changing the procedure, we can get a much higher success probability; the idea is, suppose we run Quicksort, and if it finishes in time $8n \log n$, we return the answer. Otherwise, we stop the process and retry. Now, suppose we were to repeat this process.

```
procedure ImpatientQuickSort(A):
  repeat forever:
    run QuickSort(A), killing the process after 8n log n steps
    if procedure finished, return output and exit
```

Now, the probability of "success" in each iteration of the loop is at least $1/2$ (by Markov's inequality). Thus the probability that the running time exceeds $8kn \log n$ is at most $1/2^k$, which drops exponentially with k.

Thus for this new algorithm, if we want a success probability of 0.999, we simply have to pick $k = 10$, which translates to $Q = 80n \log n$, which is much better than the earlier bound.

Hashing

The next topic we'll study is hashing. It is one of the most important tools in algorithm design, used often for tasks like finding if an object exists in a collection, or for obtaining "unique identifiers" for complex objects, and so on.

Abstractly, hashing is defined as follows: we have a universe U of "objects" (these could be strings of arbitrary sizes, and so on), and the goal is to map them to a domain V which we call "hash values" which is typically integers in some range (say $[1, 2^{32}]$). The mapping is called a *hash function* that maps U to V.

Hash functions in practice must have some nice properties; let h be the function. Then we would like:

1. Given an object x, the hash value $h(x)$ must be computable pretty quickly.
2. The set of hash values must be relatively small (compared to $|U|$)
3. If we hash a small subset U' of elements of the universe using h, we must not see too many "collisions" (two elements $x, y \in U'$ such that $h(x) = h(y)$).

Designing good hash functions is tricky. The most difficult condition to satisfy is (3) above, because ideally, we would like to have (3) for all U'. But this is impossible because once we fix a hash function and it satisfies (2), i.e., $|V| \ll |U|$, then there must be elements of V to which $|U|/|V| \gg 1$ elements map to. If a subset of this set is chosen as U', then all we have are collisions! Thus, in practice, we design hash functions that are "random enough" that for "typical" U', we do not have too many collisions with high probability.

Balls and bins

In view of the technicalities above, it turns out to be nicer to analyze randomized hashing via a more *ideal* process. This setting is often referred to as "balls and bins". Suppose we have n balls that are thrown randomly and independently into m bins. We would like to understand different properties about the distribution of the balls into bins. For example,

- What is the expected size of each bin?
- Suppose $n = m$. Then what is the expected number of bins with say precisely 4 balls?
- Again assuming $n = m$, what is the probability that *some* bin gets $\log n$ balls?

By answering these questions, we will illustrate some of the basic tools in probabilistic analysis. The main thing in these analyses is to define appropriate random variables and expressing the quantity of interest in terms of these variables.

One simple yet powerful tool in our analysis is the following, known as the *linearity of expectation*.

Theorem. (Linearity of expectation). Let X and Y be any real-valued random variables. Then for any real numbers a, b, we have $\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$.

The proof of this is quite simple (and follows from the definitions), but this is a powerful tool: in many cases, by expressing a random variable as a sum of other random variables, we can compute the expectation with much less effort!

The key thing about the linearity of expectation is that it does not depend on X, Y being independent, or anything like that. It holds for any two real-valued random variables (and thus also for complex valued, vector valued, etc.). In contrast, if we had the product of random variables, we have $\mathbb{E}[XY] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$ only if X and Y are independent.

Expected size

The entire process is symmetric (no bin or ball is special), so let us answer the question for bin 1. Let B be the random variable denoting the number of balls in bin 1. Say we want to compute it using the definition of the expectation. Then we have $\mathbb{E}[B] = \sum_{k=0}^{n} k \cdot \Pr[B = k]$.

Thus we have to compute the probability that $\Pr[B = k]$. In other words, what is the probability that exactly k of the balls fall into bin 1?

Now, if we are given k balls and we want all of them to go into bin 1, then the probability is precisely $\frac{1}{m^k}$. If we want precisely these k balls (and no others), then the probability is $\frac{1}{m^k}\left(1 - \frac{1}{m}\right)^{n-k}$. (Because the given k balls go into bin 1 and the rest should *not* go into bin 1).

The probability that exactly k balls fall into bin 1 is thus the sum over the term above over all possible choices of the k balls. Given that there are n balls total, this number is $\binom{n}{k}$. Thus, overall, we have

$\Pr[B = k] = \binom{n}{k} \frac{1}{m^k}\left(1 - \frac{1}{m}\right)^{n-k}$.

And now to compute $\mathbb{E}[B]$, we must simply do the summation $\sum_k k \cdot \Pr[B = k]$ and plug in the expression above. Simplicity, of course, is in the eye of the beholder. At this point, one can't be blamed for asking if there's an easier way.

Sums of random variables. let us try to solve the problem by simplifying B. Let X_1 be a binary (0/1 valued) random variable that "indicates" if ball 1 went into bin 1, i.e., define X_1 to be 1 if ball 1 went to bin 1 and is 0 otherwise. Similarly, define X_2 as the indicator for ball 2 going into bin 1, and so on. So we have a random variable for each of the n balls. By definition, we have $B = X_1 + X_2 + \cdots + X_n$. Note that this is an equality between random variables, meaning that it always holds no matter what values the variables attain.

Now, taking the expectation and using linearity, we get $\mathbb{E}[B] = \sum_{i=1}^{n} \mathbb{E}[X_i]$ (call this (*)). The key observation is now that since X_i is binary, computing the expectation is easy! In fact, it is exactly $0 \cdot \Pr[X_i = 0] + 1 \cdot \Pr[X_i = 1] = \Pr[X_i = 1]$. This is a useful thing to remember for 0/1 random variables: the expectation is exactly the probability that the variable equals 1.

In our case, what is $\Pr[X_i] = 1$? It is the probability that ball i goes to bin 1 (by definition), which is exactly $1/m$ (as there are m bins and the ball goes to a random bin). Plugging this into (*) above, we get $\mathbb{E}[B] = n/m$.

Alternative approach. In class, an alternate (and elegant!) approach was also proposed: let us define B_i to be the number of bins that land up in bin i. Then we always have the identity $B_1 + B_2 + \cdots + B_m = n$. Taking the expectation on both sides and using linearity, we get $\sum_{j=1}^{m} \mathbb{E}[B_j] = n$. Since all the bins are identical in our process, B_i are all identically distributed, and thus have the same expectation. Thus $\mathbb{E}[B_i] = n/m$ for all i.

Number of bins with 4 balls (when $m = n$)

Let us now consider the second question we posed: when $m = n$, what is the expected number of bins that get exactly 4 balls?

Now if $m = n$, the calculation above shows that the expected size of each bin is 1. So every bin, in expectation, has 1 ball -- this does not mean that every bin has *precisely* one ball!

In fact, in the first computation we tried (where we wrote out $\Pr[B = k]$), we found the distribution of B, the number of balls landing in bin 1. So even if k is a somewhat large value, there is some non-zero (though perhaps negligible) probability that k balls fall into bin 1.

Now, back to the question. As before, let us define some random variables. Let Y be the random variable of interest: the number of bins that get precisely 4 balls.

This time, it is even harder than before to compute $\mathbb{E}[Y]$ by direct computation (where for each r, we want to find $\Pr[Y = r]$). To see why, you are encouraged to try!

But we can use the "sum of simpler variables" approach from before. Let us write $Y = X_1 + X_2 + \cdots + X_m$, where X_i is a 0/1 random variable that indicates if bin i received precisely 4 balls. From linearity of expectation, we have $\mathbb{E}[Y] = \sum_{i=1}^{m} \mathbb{E}[X_i]$. (Here, as before, the X_i are not independent random variables; but linearity holds irrespective of that, as we discussed.)

Again, we have $\mathbb{E}[X_i] = \Pr[X_i = 1]$ (as X_i is a binary variable). Now, $X_i = 1$ means that bin i received precisely 4 balls (by definition). What is the probability of this?

Recall that we computed this quantity before; in fact, we did this not just for 4 but for any k (when attempting to compute the expected number of balls in a bin). Thus, plugging in $k = 4$ in that formula, we get $\Pr[X_i = 1] = \binom{n}{4} \frac{1}{m^4} \left(1 - \frac{1}{m}\right)^{n-4}$.

Using $m = n$ and simplifying, we get $\Pr[X_i = 1] = \frac{n(n-1)(n-2)(n-3)}{24n^4} \left(\frac{n-1}{n}\right)^{n-4} = \frac{n(n-1)(n-2)(n-3)}{24(n-1)^4} \cdot \left(\frac{n-1}{n}\right)^{n}$.

Assuming that n is large, it's a standard fact that $\left(1 - \frac{1}{n}\right)^n \to 1/e$, where e is the base of the natural logarithm. Thus the second term on the RHS above is roughly $1/e$. For large n, the first term converges to $1/24$, and thus the overall probability that $X_i = 1$ is very close to $1/(24e) \approx 1/65$. And this is the same computation for every i.

Thus we have $\mathbb{E}[Y] \approx n/65$, implying that in expectation, about $n/65$ bins receive precisely 4 balls. Note that this is a reasonably large number -- a constant fraction of the bins.