1 Probability

First, recall a couple useful facts from last time about probability:

- Linearity of expectation: $\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$ even when X and Y are not independent.
- Geometric sums: $\sum_{i=0}^{\infty} p^i = \frac{1}{1-p}$ (for $0 < |p| < 1$) and $\sum_{i=0}^{n} p^i = \frac{1-p^{n+1}}{1-p}$ (for all p).
- For random numbers X which only take on nonnegative integer values, $\mathbb{E}(X) = \sum_{j=0}^{\infty} \mathbb{P}(X > j)$.
- Geometric random variables: If some event happens with probability p, the expected number of tries we need in order to get that even to occur is $1/p$.
- Markov's inequality: For any nonnegative random variable X and $\lambda > 0$, $\mathbb{P}(X > \lambda \cdot \mathbb{E}X) < \frac{1}{\lambda}$.

Exercise (Fixed points in permutations)**.** *Suppose we pick a uniformly random permutation on n elements. First, how would we do it? And then, what is the expected number of fixed points it has?*

Exercise (Special Dice)**.** *Consider the following game: there are three dice, each with its faces labeled by integers. You are allowed to choose what you think is the "best" die, after which your opponent will choose another die. Each of you roll yours, and the person who rolls the larger number receives* \$1 *from the other player. Should you always play?*

Exercise (Coupon Collector)**.** *There are n coupons $C_1, \ldots, C_n$. Every time you buy a magazine, a uniformly random coupon C_i is inside. How many magazines do you have to buy, in expectation, to get all n coupons? How many magazines do you have to buy to get all n coupons with a failure probability of at most P?*

Exercise (Election Problem (Challenge))**.** *America is having its next presidential election between two candidates, and for each of the n states you know the probability that candidate 1 wins that state is P_i. For simplicity we'll assume that independent candidates don't exist, so the probability candidate 2 wins is $1 - P_i$.*

Because of the electoral college, state i is worth S_i points: when a candidate wins a state, he/she gets all S_i points. To win, a candidate must get a strict majority of the electoral votes. Assuming a tie isn't possible, what's the probability that candidate 1 wins?

2 Probabilistic Algorithms

We have looked at probailistic algorithsm with bounded running time, but that may be incorrect with a small probability. More specifically:

- **RP** is the class of languages for which there exists a polynomial-time randomized algorithm which always returns correctly for inputs in the language, and which returns correctly at least half the time for inputs not in the language.
- **BPP** is the class of languages for which there exists a polynomial-time ranomized algorithm which returns correctly at least 2/3 of the time on any input.

Note that the randomness is over the coins of the algorithm, not over the input.

Oftentimes, we can decrease the failure probability by running the algorithm multiple times. For example, if we use our algorithm to check some computation but can be fooled with some probability, we can just run it multiple times. Indeed, this can be used to show that it is sufficient for a **BPP** algorithm to be correct at least $\frac{1}{2} + \frac{1}{p(n)}$ of the time for any polynomial $p(n)$ in the length of the input.

2.1 Hashing

Remember from class that a hash function is a mapping $h : \{0, \ldots, n-1\} \to \{0, \ldots, m-1\}$. In general, because one of the goals of a hash function is to save space, we will see that $m << n$.

Hashing-based data structures are useful since they ideally allow for constant time operations (lookup, adding, and deletion), although collisions can change that if, for example, n is too large or you use a poorly chosen hash function. (Why?)

In practice, there are several ways to deal with collisions while hashing, such as:

- Chaining: Each bucket holds a set of all items that are hashed to it. Simply add x to this set.
- Linear probing: If $f(x)$ already has an item, try $f(x) + 1, f(x) + 2$, etc. until you find an empty location (all taken mod m).
- Perfect hashing: If the set is known and fixed, generate a hash function for that specific set. However, this does not support insertions and deletions.
- Automatic resizing: Once the number of elements exceeds cm for some constant c, make the hash table larger by doubling m. In practice, a load factor of $c = 0.75$ is fairly common.

2.2 Document Similarity

Consider two sets of numbers, A and B. For concreteness, we will assume that A and B are subsets of 64 bit numbers. We may define the *resemblance* of A and B as

$$\text{resemblance}(A, B) = R(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

The resemblance is a real number between 0 and 1. Intuitively, the resemblance accurately captures how close the two sets are. To calculate this, we could sort the sets and compare, but this is still rather slow, so we'd like to do better. We will do this by computing an approximation to the set resemblance, rather than the exact value.

Let π_1 be some random permutations of the numbers $1, 2, \ldots, 2^{64}$ (that is, it is a bijection $\{1, \ldots, 2^{64}\} \to \{1, \ldots, 2^{64}\}$). Let $\pi_1(A) = \{\pi_1(x) : x \in A\}$.

Exercise. *When does* $\min\{\pi_1(A)\} = \min\{\pi_1(B)\}$*?*

Exercise. *What is* $Pr[\min\{\pi_1(A)\} = \min\{\pi_1(B)\}]$*?*

This gives us a way to estimate the resemblance. Instead of taking just one permutation, we take many– say 100. For each set A, we preprocess by computing $\min\{\pi_j(A)\}$ for $j = 1$ to 100, and store these values. To estimate the resemblance of two sets A and B, we count how often the minima are the same, and divide by 100. It is like each permutation gives us a coin flip, where the probability of a heads (a match) is exactly the resemblance $R(A, B)$ of the two sets.

If you have some more experience with this type of probability, you might think about how we could bound the accuracy of this method – that is, you want to make some statement of the form "the algorithm is within ϵ of the correct set resemblence with probability at least $1 - \delta$" for some $\epsilon, \delta \in [0, 1]$.

Now, we can use this to think about similarity between documents via a method called *shingling*. That is, divide the document into pieces of consecutive words and hash the results into a set S_D. Once we have the shingles for the document, we associate a document *sketch* with each document. The sketch of a document S_D is a list of say 100 numbers: $(\min\{\pi_1(S_D)\}, \min\{\pi_2(S_D)\}, \min\{\pi_3(S_D)\}, \ldots, \min\{\pi_{100}(S_D)\})$.

Exercise. *How well does this method do?*

More concretely, suppose that we mark two documents as "similar" if their sketches match on at least 90 values. What is the probability that this happens if the set resemblence of the two sets of shingles is r*?*

3 Random Walks

A random walk is an iterative process on a set of vertices V. In each step, you move from the current vertex v_0 to each $v \in V$ with some probability. The simplest version of a random walk is a one-dimensional

random walk in which the vertices are the integers, you start at 0, and at each step you either move up one (with probability 1/2) or down one (with probability 1/2).

2-SAT: In lecture, we gave the following randomized algorithm for solving 2-SAT. Start with some truth assignment, say by setting all the variables to false. Find some clause that is not yet satisfied. Randomly choose one the variables in that clause, say by flipping a coin, and change its value. Continue this process, until either all clauses are satisfied or you get tired of flipping coins.

We used a random walk with a completely reflecting boundary at 0 to model our randomized solution to 2-SAT. Fix some solution S and keep track of the number of variables k consistent with the solution S. In each step, we either increase or decrease k by one. Using this model, we showed that the expected running time of our algorithm is $O(n^2)$.

Exercise. *This weekend, you decide to go to a casino and gamble. You start with k dollars, and you decide that if you ever have $n \geq k$ dollars, you will take your winnings and go home. Assuming that at each step you either win* \$1 *or lose* \$1 *(with equal probability), what is the probability that you instead lose all your money?*

Exercise. *Now suppose that there are $n+1$ people in a circle numbered $0, 1, \ldots, n$. Person 0 starts with a bag of candy. At each step, the person with the bag of candy passes it either left or right with equal probability. The* last *person to receive the bag wins (and gets to keep all the candy). So if you were playing, then you would want to receive the bag only after everyone else has received the bag at least once. What is the probability that person i wins?*