Recitation 8 — Probability and Treaps

Parallel and Sequential Data Structures and Algorithms, 15-210 (Spring 2012)

March 7, 2012

Today's Agenda:
- Announcements
- Conventions for Writing Tests
- Probability
- Treaps

1 Announcements

- Assignment 5 is due **tomorrow**, Thursday the 8th.
- Assignment 6 will be released at the end of this week
- Bill wrote an MLton driver for assignment 5, which should run much faster than SML/NJ. See Piazza for details
- Questions about homework, class, life, universe?

2 Test Conventions - 3 mins

Many of you have been writing tests that assign to constants, as you learned in 150 (e.g. val true = 1+1=2). We prefer that you follow our convention introduced in HW 1 of a test structure that exports a boolean `all` of whether all the tests passed. We will provide stubs in future HWs to make this easier for you.

On HWs 5 and later, you will lose points for tests that assign to constants
On HWs 6 and later, you will lose points for not putting your tests in a separate file and structure

3 Probability

3.1 Max Cut - 15 mins

Given a graph, we want to split its vertices into two groups A and B such that the number of edges between A and B is as large as possible. MAX-CUT is NP-hard (but this is hard to show).

3.1.1 Decision Problems - *** OPTIONAL ***

Sometimes, we state the max cut problem as: Given a graph G and an integer k, is there a cut of size at least k? This is a decision problem (the output is "yes" or "no"); the original problem is an optimization problem (the output is an integer).

I claim these two problems are "almost the same". Why, and what do I mean by "almost"?

Obviously, the original problem is harder (if you know the real value, you can just compare it to k). But if you can answer whether you can get k, you can just binary search for the answer, so the optimization is at most $\log(n)$ times harder than the decision problem. In practice, factors of $\log(n)$ never matter.

This is obviously a very general reduction (it applies to any problem where it makes sense to ask "is the answer at least k?"). Often, the decision problem is easier to think about than the optimization problem, so it is often useful.

3.1.2 A Random Algorithm

Idea: randomly assign each vertex to A or B. We define an indicator variable X_e for each edge e to be 1 if e goes between A and B and 0 otherwise. Note $P(X_e = 1) = 0.5$ (fix one endpoint; the other is in the other side with probability 0.5). By linearity of expectation, this random cut gets $m/2$ edges.

This implies that there actually is a cut of size $m/2$. Why? (In a set of numbers, one of them is at least the average) More generally:

Theorem 3.1. $P(X \geq E[X]) > 0$

Proof. Suppose not. Then $E[X] = \sum_{i=1}^{n} x_i P(X = x_i) < \sum_{i=1}^{n} E[X]P(X = x_i) = E[X] \sum P(X = x_i) = E[X]$, which is a contradiction. □

Can we say anything about the probabilty we actually get a good cut? How about $P(X \geq m/2)$? (No) Why not? (they could all be just below the average)

We *can* say something about $P(X > m(1/2 - A))$. What? Let $p = P(X \leq m(1/2 - A))$.

$$\begin{aligned}
&E[X] = m/2 \leq pm(1/2 - A) + (1 - p)m \\
&\Rightarrow 1/2 \leq p/2 - pA + 1 - p \\
&\Rightarrow -1/2 \leq -p/2 - pA \\
&\Rightarrow 1 \geq p + 2pA \\
&\Rightarrow p \leq \frac{1}{1 + 2A}
\end{aligned}$$

So $P(X > m(1/2 - A)) > 1 - \frac{1}{1+2A}$.

We can adapt this into a non-random algorithm to guarantee a cut of size $m/2$.

We will examine the vertices in some order deciding which side to put them in. Let v be the vertex. We will call the "edges determined by v" the ones whose other endpoint has already been assigned a side. We put v in the side which maximizes the number of edges determined by v which make it into the cut (at least half).

Every edge gets determined eventually, and in each group that is determined, at least half of them make it into the cut, so at least half the edges make it into the cut.

3.2 Weak Law of Large Numbers - 15 mins

Recall Markov's inequality: If X is non-negative, then $P(X \geq A) \leq \frac{E[X]}{A}$

The quantity $E[(X - E[X])^2]$ is called the variance of X, written $Var(X)$ or σ_X^2.

Theorem 3.2. *(Chebyshev's Inequality)* $P(|X - E[X]| \geq d) \leq \frac{Var(X)}{d^2}$

Proof. Let $Y = (X - E[X])^2$.
Note that Y is non-negative.
Note $P(|X - E[X]| \geq d) = P((X - E[X]^2) \geq d^2)$.
Note $E[Y] = Var(X)$.
Apply Markov's inequality to get $P(Y \geq d^2) \leq \frac{E[Y]}{d^2} = \frac{Var(X)}{d^2}$. □

Note $Var(aX) = E[(aX - E[ax])^2] = E[(aX - aE[X])^2] = a^2E[(X - E[X])^2] = a^2Var(X)$.

We will use without proof that if X and Y are independent, then $Var(X + Y) = Var(X) + Var(Y)$ (the proof is very similar to the proof of linearity of expectations, although a bit messier)

Suppose we have a program, and we want to measure how long (in seconds) it takes to run. Fine, that's what `time` (a unix program) is for. But `time` gives different results when I run it again. What should I do?

Well, I guess I want the average time. Then I'll take several measurements and average them together. Why does this work?

Theorem 3.3. *(Weak Law of Large Numbers) Let $X_1, X_2, \ldots$ be an infinite sequence of independant and identically distributed randomly variables. Defines $\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$ (the sample mean of the first n observations). Then $\forall \varepsilon > 0 : \lim_{n\to\infty} P(\overline{X}_n - E[X] > \varepsilon) = 0$.*

Proof. $Var(\overline{X}_n) = \frac{1}{n^2}Var(\sum_{i=1}^{n} X_i) = \frac{1}{n^2}nVar(X_i) = \frac{Var(X_i)}{n}$
So $Var(\overline{X}_n) \to 0$
Apply Chebyshev's to show $P(\overline{X}_n - E[X] > \varepsilon) \leq \frac{Var(\overline{X}_n)}{\varepsilon^2} \to 0$. □

Fine, but I don't want to take infinitely many measurements. Suppose I took 10 measurements? How close am I to the real average, probably? This is a somewhat complicated question that requires more detailed information about the distribution, like the variance. How can you estimate the variance? From your measurements! How good is *that* estimate? Hmm... Confidence intervals answer this kind of question in science and statistics; we won't review the math on them today.

3.3 Simulating a Fair Coin - OPTIONAL

Suppose you have a biased coin which lands heads with probability p, and you want to generate a fair sequence of random bits.

Q: How can we do this?
A: Flip the coin twice. If you see "HH" or "TT", do nothing. If you see "HT", output 1. If you see "TH", output 0.

Note $P[HT] = p(1 - p) = P[TH]$, so in expectation there will be as many 1s as 0s.

Q: How many bits do we lose, on average
A: For every 2 bits, we generate $2pq$ bits.

Exercise: Can you do better? How much better?

How about the other way: given a fair coin, can you give me bits biased with probability p?

Lemma 3.4. *Only if $p = x/2^k$ for some k*

Proof. After k flips, all sequences have probability $\frac{1}{2^k}$, so any combination of these probabilities will have denominator $\frac{1}{2^k}$. □

Of course, every real number can be approximated arbitrarily closely as $x/2^k$. In fact, this is infinite binary expansion!

Let $p = p_1 ... p_n$ be given in binary. How can we generate a bit with probability p? Flip up to n coins, stopping when we get a head. If we get a head at position i, return 1 if $p_i = 1$ and 0 otherwise. Why does this work? We get the sequence $0^k 1$ with probability $2^{-(k+1)}$, and $p = \sum_{i=1}^{n} 2^{-i} p_i$.

4 Treaps

4.1 Uniqueness - 3 mins

```
datatype treap = Leaf | Node of treap * treap * key * value * priority
```

Theorem 4.1. *Given a list of key-priority pairs $(k_1, p_1), ..., (k_n, p_n)$ with all the k_i distinct and all the p_is distinct, there is a unique treap that can be built from them*

Proof. By strong induction on n. The root is uniquely determined (as the element of lowest priority). The elements in the left and right subtrees are determined by the key in the root. Apply IH on the left and right subtrees. □

4.2 Split - 10 mins

Three *important* questions:
What does it mean for this to maintain balance? (It's still a treap) Why does it work? (work through the cases; observe that BST and heap invariants are preserved) Why does being a treap suffice to maintain balance? (Any "subtreap" is "just as random". Insertions and deletions are still "just as random")

```
fun split(T,k) =
    case T of
        Leaf => (Leaf, NONE, Leaf)
      | Node(L,R,k',v', p') =>
            case compare(k,k') of
                LESS =>
                let val (L', r, R') = split(L,k)
                in (L', r, Node(R', R, k, v, p))
              | EQUAL => (L,SOME(v),R)
              | GREATER =>
                let val (L', l, R') = split(R,k)
                in (Node(L, L', k', v', p'), l, R')
```

4.3 Join - 3 mins

We assume the existence of `get_priority : key -> priority` (a hash function), and that all the priorities and keys are distinct.

Why does join preserve treapness? (work through the cases; observe that BST and heap invariants are preserved)

```
fun join'(Leaf, R) = R
|   join'(L, Leaf) = L
|   join'(Node(LL,LR,lk,lv,lp) as L, Node(RL,RR,rk,rv,rp) as R) =
    if lp < rp
    then Node(join'(LL,LR), R, lk, lv, lp)
    else Node(L, join'(RL,RR), rk, rv, rp))

fun join(L,NONE,R) = join'(L,R)
|   join(L,SOME(k,v),R) = join'(L, join'(Node(Leaf,Leaf,k,v,get_priority(k)),R))
```