15-859(D) Randomized Algorithms, 10/6/98

```
* finish occupancy problems
* Hashing
* lower bounds for randomized algs
```

Occupancy problems, contd

Throwing n balls at random into n bins. For $k \ll n$, Pr[given bin has k balls] $\approx 1/(e \cdot k!)$.

- Whp, most-full bin has $O(\log(n)/\log\log(n))$ balls.

- $\Omega(\log(n)/\log\log(n))$ too.
  - Pick k so that prob a given bin is that full is at least $1/\sqrt{n}$.
  - If 1st bin is that full then done. If not, then event that 2nd bin has this many has only gone up.
  - Formally, analyze using notion of Martingale - Ch 4.4.

Coupon collector's problem

What is the expected number of balls to fill all bins?

- One way to analyze: Prob a given bin is empty after kn balls is $\approx 1/e^k$, so at $k = \ln(n) + c$, this is $1/n^c$. So, likely all filled after $n\ln(n)+cn$ balls.

- Or, let X_i be the time to fill a new bin when there are i empty ones remaining.
  - Total time $X = \sum_i X_i$.
  - Each new ball has i/n chance of filling a new bin, so X_i is geometrically distributed with $\mathbf{E}[X_i] = n/i$.
  - So, expected time is $\sum_i n/i = nH_n \approx n \ln n$.

Hashing

- Set up: domain M, subset S of M ($|S| \ll |M|$) we care about. Want to map M to small set N (e.g., $|N| = O(|S|)$) with few collisions in a way we can calculate.
- Function $h : M \to N$ is called a hash function.
- Lots of uses for hashing. E.g., storing a dictionary. Static versus dynamic.

Idea of hashing: maintain a small table, and use h to map keys into this table. If h behaves randomly, shouldn't get too many collisions.

Even if in practice you don't use randomness in selecting the function (as in the methods we will prove things about), the motivation/inspiration is random behavior.

Simple facts

- Simple fact: if M is sufficiently large ($|M| > |S| \cdot |N|$), then for any h, there exists S such that all elements of S collide.

- So, need to either choose h based on S, or make randomness assumptions on S, or choose h using some randomness.

- For convenience, lets abuse notation and let $N = |N|$ and $M = |M|$.

Universal hash functions

- H is a *2-universal* family of hash functions from M to N if for all $x, y \in M$, $(x \neq y)$:

$$\Pr_{h \leftarrow H}[h(x) = h(y)] \leq 1/N.$$

- H is a *strongly 2-universal* family if for all $x_1 \neq x_2 \in M$, all $y_1, y_2 \in N$:

$$\Pr_{h \leftarrow H}[h(x_1) = y_1 \wedge h(x_2) = y_2] = 1/N^2.$$

  I.e., the random variables $h(x_i)$ are uniformly distributed and pairwise independent.

Why are they useful....?

.... just need linear size table for $O(1)$ expected time per operation.

Perfect hashing

A hash function h is *perfect* for a set S if it causes *no* collisions.

Universal $\rightarrow$ perfect:

- Say $O(|S|^2)$ space is OK.

  Then, just do a universal hash function into table of size $n = |S|^2$. For any $x \neq y$ we have

  $$\Pr[h(x) = h(y)] \leq 1/n.$$

  So,

  $$\Pr[\exists x \neq y \text{ that collide}] \leq \binom{|S|}{2}/n < 1/2.$$

  So, just pick random $h \in H$ and try it. If it fails, try again.

- What if we only want to use $O(|S|)$ space?

Perfect hashing, contd

- Let $n = |S|$ and try a universal hash function.

- Let B_i = size of ith bin. Let C_{xy} be indicator RV for event that x and y collide.

- What is the expected value of $\sum_i (B_i)^2$?

$$\sum_i (B_i)^2 = \sum_{x,y} C_{xy} = n + \sum_{x \neq y} C_{xy}$$

  This is less than $2n$ in expectation.

Now what?

- For at least half of h's this is at most $4n$. Now, hash each bucket a second time using the previous approach.

- Final hash function is a tree of depth 2, into a table of size $4n$ with no collisions.

How to construct Universal Hash Functions?