1 Topics Covered - Part 1

1.1 Algorithm Strategies

- **FFT**. For pattern matching.
- **Greedy**. Pick the best option at each step. e.g., Horn Formula Algorithm
- **Divide and Conquer**. Split the problem into smaller subproblems (often in half), solve the subproblems, and combine the results. e.g., Mergesort, Integer Multiplication, Strassen's
- **Dynamic Programming**. Useful for when there are many overlapping subproblems. e.g., Fibonacci, Edit Distance, Floyd-Warshall (all pairs shortest paths)

2 Practice Problems

2.1 Egyptian Fractions

For any rational number $p < 1$, it is possible to write $p = \sum \frac{1}{p_i}$ for a finite number of distinct p_i. For instance, for $p = \frac{6}{7}$, we can write $\frac{6}{7} = \frac{1}{2} + \frac{1}{3} + \frac{1}{42}$. This decomposition can be done in many ways (infinite ways actually). Provide an algorithm to find one such summation, given the fraction p.

Solution.
Solution: We provide a greedy solution here. Pick the fractions in the decomposition one at a time, at each step adding the fraction of the form $\frac{1}{p_i}$ which covers as much of the remaining sum as possible. For instance, when assembling the decomposition for $\frac{6}{7}$, we first choose $\frac{1}{2}$, leaving $\frac{5}{14}$ to cover. Since $\frac{1}{3} < \frac{5}{14} < \frac{1}{2}$, we choose $\frac{1}{3}$ as our next fraction in the decomposition. Finally, we see that $\frac{5}{14} - \frac{1}{3} = \frac{1}{42}$, giving the decomposition $\frac{6}{7} = \frac{1}{2} + \frac{1}{3} + \frac{1}{42}$.

To prove that this process ends, we consider the remaining sum that needs to be covered at each step. In general, when we want to show that a process terminates, we often want to find some invariant that cannot change past some point. Note that if the remaining sum to be covered is of the form $\frac{m}{n}$, we have that $\frac{1}{u} \leq \frac{m}{n} > \frac{1}{u-1}$, we have that $mu - n < m$. Also note that the remainder that we must cover through further decomposition is $\frac{m}{n} - \frac{1}{u} = \frac{mu-n}{nu}$. Even after xpressing in simplified terms, $mu - n < m$ implies that the numerator of $\frac{mu-n}{nu}$ is strictly decreasing. Thus, in every step of the greedy algorithm, the remaining sum to cover has a strictly decreasing numerator. Since this number cannot be > 1 for infinite steps, we will eventually reach a remainder of the form $\frac{1}{p_n}$, at which point we stop.

2.2 Cutting Wood

From MIT 6.006, Spring 2011

Given a log of wood of length k, Woody the woodcutter will cut it once, in any place you choose, for the price of k dollars. Suppose you have a log of length L, marked to be cut in n different locations labeled $1, 2, \ldots, n$. For simplicity, let indices 0 and $n+1$ denote the left and right endpoints of the original log of length L. Let the distance of mark i from the left end of the log be d_i, and assume that $0 = d_0 < d_1 < d_2 < \ldots < d_n < d_{n+1} = L$.

Exercise. *Determine the sequence of cuts to the log that will (1) cut the log at all the marked places, and (2) minimize your total payment to Woody.*

Solution.
Solution: DP with subproblem $f(i, j)$, the minimum cost of cutting a log with endpoints d_i, d_j at all its cut points.

2.3 Guitar Hero

From MIT 6.046, Fall 2009

You are training for the World Championship of Guitar Hero World Tour, whose first prize is a real guitar. You decide to use algorithms to find the optimal way to place your fingers on the keys of the guitar controller to maximize the ease by which you can play the 86 songs. Formally, a note is an element of $[A; B; C; D; E]$ (representing the green, red, yellow, blue, and orange keys on the guitar). A chord is a nonempty set of notes, that is, a nonempty subset of $[A; B; C; D; E]$. A song is a sequence of chords: $[c_1; c_2; \ldots; c_n]$. A pose is a function from $[1; 2; 3; 4]$ to $[A; B; C; D; E; \emptyset]$, that is, a mapping of each finger on your left hand (excluding thumb) either to a note or to the special value $\emptyset$; meaning that the finger is not on a key. A fingering for a song $[c_1; c_2; \ldots; c_n]$ is a sequence of n poses $[p_1; p_2; \ldots; p_n]$ such that pose p_i places exactly one finger on each note in c_i, for all $1 \leq i \leq n$.

You have carefully defined a real number $D[p; q]$ measuring the difficulty of transitioning your fingers from pose p to q, for all poses p and q. The difficulty of a fingering $[p_1; p_2; \ldots; p_n]$ is the sum $\sum_{i=2}^{n} D[p_{i-1}, p_i]$.

Exercise. *Give an $O(n)$-time algorithm that, given a song $[c_1; c_2; \ldots; c_n]$, finds a fingering of the song with minimum possible difficulty.*

Solution.
Solution: DP with subproblem $f(i, p_i)$, the minimum possible difficulty for the song $[c_1; \ldots; c_i]$ when the i-th pose is p_i. This is a classic Markov Chain DP.

2.4 Inversions and Permutations

Exercise. *How many permutations of $\{1, 2, ..., N\}$ have exactly K inversions? For example, for $N = 3$ and $K = 1$, the answer is 2: $\{2, 1, 3\}$ and $\{1, 3, 2\}$.*

Solution.
Solution: DP with subproblem $f(i, j)$, the number of permutations of $i, i+1, ..., n$ with exactly j inversions.

Recurrence is:

$$f(i,j) = \sum_{l=j-(N-i)}^{min(j,K)} f(i+1,l)$$

Essentially, at each step, we decide where to place the smallest number in the set. With a set of size $N-i$, i.e. the set $\{i+1,...,N\}$, we can add between 0 and $N-i$ new inversions depending on where we insert i into the set. The base cases are $f(N,0) = 1$ and $f(N,j) = 0$ for $j = 1,2,...,K$. Moreover, we define $f(N,i) = 0$ for $i < 0$. This takes $O(NK)$ space and $O(N^2K)$ time.

3 Topics Covered - Part 2

3.1 Math Facts

- (Markov's Inequality) $\mathbb{P}(X > \lambda \mathbb{E}\, X) < \frac{1}{\lambda}$ for any nonnegative r.v. X and $\lambda > 0$.
- $\mathbb{E}\, X = \sum_{k=0}^{\infty} \mathbb{P}(X > k)$ for any nonnegative r.v. X.
- If a coin lands on heads with probability p, the average number of flips needed for the first head is $\frac{1}{p}$.

3.2 Randomized Algorithms

- Monte Carlo and Las Vegas Algorithms
- Frievald's Algorithm
- Quicksort and Quickselect
- Karger/Karger-Stein algorithms
- 2SAT

3.3 Data Structures

- Skip Lists
- Universal Hash Families
- Two Level Perfect Hashing

4 Practice Problems

4.1 True/False

(a) The maximum possible number of min-cuts a graph on n vertices can have is $\binom{n}{2}$.

Solution.
True. Recall that Karger's algorithm returns any particular min-cut with probability at least $\frac{1}{\binom{n}{2}}$, hence there can be at most $\binom{n}{2}$ mincuts. The cycle on n vertices has exactly $\binom{n}{2}$ cuts, so this is the maximum.

(b) Suppose we try to solve 3-SAT the same way we solved 2-SAT - that is, find some unsatisfied disjunction and flip one literal in the disjunction chosen uniformly at random. Then we can achieve an error probability of at most $\frac{1}{100}$ in time $O(n^{100})$.

Solution.
False. For any satisfying assignment S, and our assignment A, at each reassignment we can now only be sure that we are making A agree with an additional variable S with probability at least $\frac{1}{3}$ (it was $\frac{1}{2}$ for 2-SAT). Now, if t_i is the expected number of reassignments necessary until S and A agree, in the worst case we have equations $t_0 = t_1 + 1$, $t_n = 0$ and $t_i = \frac{1}{3}t_{i+1} + \frac{2}{3}t_{i-1} + 1$ otherwise. Intuitively, this is much worse than with 2-SAT - we are on average drifting away from our satisfying assignment. It's not hard to show that the exact solution is $t_i = 2^{n+2} - 2^{i+2} - 3(n-i)$, which is exponential.

(c) Quickselect has an asymtotically faster worst case running time than quicksort.

Solution.
False, they are both $O(n^2)$ in the worst case.

4.2 Short Answer

Exercise. *Describe how to construct a Monte Carlo algorithm from a Las Vegas algorithm. When can we construct a Las Vegas algorithm from a Monte Carlo algorithm?*

Solution.
Suppose our Las Vegas algorithm has expected runtime T. Then by Markov's inequality, the probability that it runs for more than λT steps is less than $\frac{1}{\lambda}$. Choosing $\lambda = \frac{1}{p}$, it suffices to run the Las Vegas algorithm for $\frac{1}{p}T$ steps. If it hasn't yet terminated, output something arbitrary.

It's possible to do better. Again by Markov's inequality, the probability that the Las Vegas algorithm runs for more than $2T$ steps is at most $\frac{1}{2}$. Then if we run the Las Vegas algorithm k times for $2T$ steps each time, the probability that it will not terminate on any run is at most $\frac{1}{2^k}$. For this to be smaller than p, it's enough to have $k \geq \log_2 \frac{1}{p}$. Then the total runtime is $T \log_2 \frac{1}{p}$.

We can construct a Las Vegas algorithm from a Monte Carlo algorithm when there is an efficient procedure for verifying the output of the Monte Carlo algorithm - just run the Monte Carlo algorithm until we verify that the output is correct. If the Monte Carlo algorithm has error probability p, we will run it $\frac{1}{p}$ times on average.

Exercise. *Suppose I have n fair coins. I toss each coin until I flip heads. Let M be the greatest number of times I flip any one coin. Show that $\mathbb{E}\, M \leq \log_2 n + 1$.*

Solution.
Suppose we use the n coins to insert n elements into a skip list. Then M is the height of the list. We showed in class that the expected height was at most $\log_2 n + 1$.

4.3 Long Answer

Exercise. *Suppose it is possible to colour the vertices of an undirected graph G blue, yellow and red such that no edge connects two vertices of the same color. Such graphs are called* 3-colorable*.*

(a) Show that it is possible to partition the vertices of G into sets S and T such that no three vertices in S and no three vertices in T form a triangle.

(b) Give an efficient Las Vegas algorithm to achieve this (you are told G is 3-colorable, but you aren't given the coloring).

Solution.
(a) Any triangle has distinct colours at its vertices. Then it suffices to choose S to be any set containing all blue vertices but no yellow vertices.

(b) Fix some 3-coloring of G. Randomly assign the vertices to S and T. The idea is similar to the 2-SAT algorithm - while at least one of them has a triangle, choose one vertex of a triangle uniformly at random and move it to the other set. We aim to end up with all blue vertices in S and all yellow vertices in T (or vice versa).

With probability $\frac{1}{3}$, we move a blue/yellow vertex to the correct set. With probability $\frac{1}{3}$, we move a blue/yellow vertex away from the correct set. With probability $\frac{1}{3}$, we move a red vertex.

Suppose that there are i blue vertices in S/yellow vertices in T, and k yellow/blue vertices in total. If i is 0 or k, we are done. Recall from the 2-SAT analysis that the expected number of steps in a random walk before we hit a boundary is $O(n^2)$. Here we make a step with probability $\frac{2}{3}$, so the expected number of steps is still $O(n^2)$.

Finally, we need an efficient way to check for triangles. First, for every pair (u, v) of vertices, construct two lists - one of all vertices w in T such that $\{u, v, w\}$ is a triangle, and one of all such vertices in S. This takes time $O(n^3)$. When we are looking for a triangle in S and T, it suffices to find some u, v in the same set with a nonempty list corresponding to that set. This takes time $O(n^2)$. Updating the lists after reassignment is also $O(n^2)$. Then total time is $O(n^3 + n^4) = O(n^4)$.

Exercise. *Suppose we try to insert n elements into a hash table in the following way: begin with a table of size one. Whenever two consecutive elements hash to the same location, we start over and attempt to insert all the elements into a table of twice the size. Assume we use independently chosen universal hash functions for each attempt. Show that the expected size of the final table is $O(n)$.*

Solution.
Suppose our table has size m. Since the hash is universal, the probability two consecutive elements hash to the same location is at most $\frac{1}{m}$. Then the proability that this ever happens is at most $\frac{n-1}{m} < \frac{n}{m}$ by a union bound.

Let X be the final size. Choose $k = \lceil \log n \rceil$. Note that

$$\mathbb{P}(X \geq 2^{k+i}) \leq \prod_{j=0}^{k+i-1} \min\left\{1, \frac{n}{2^j}\right\} \leq \prod_{j=k}^{k+i-1} \frac{n}{2^j} \leq \prod_{j=0}^{i-1} \frac{1}{2^j} = 2^{-i(i-1)/2}$$

Then

$$\begin{aligned}
\mathbb{E}\, X &= \sum_{i=0}^{\infty} 2^i\, \mathbb{P}(X = 2^i) \\
&\leq 2^k\, \mathbb{P}(X \leq k) + \sum_{i=1}^{\infty} 2^{k+i}\, \mathbb{P}(X = k+i) \\
&\leq 2^k + \sum_{i=1}^{\infty} 2^{k+i} 2^{-i(i-1)/2} \\
&= 2^k + 2^k \sum_{i=1}^{\infty} 2^{-(i^2-2i)/2} \\
&= O(2^k) \\
&= O(n)
\end{aligned}$$