CS 124 **Midterm Review** 3/4/17

1 Format

You will have 83 minutes to complete the exam. The exam will have true/false questions, multiple choice, example/counterexample problems, run-this-algorithm problems, and problem set style present-and-prove problems.

The exam is difficult. In previous years, the median score has been around 60%.

2 Topics Covered

This is a pretty comprehensive list of topics we have gone over this first half of the semester, but anything said in lecture is fair game for the exam.

2.1 Math Fundamentals

- **Coin Flipping** and basic statistics.
- **Induction**. If $P(n)$ is a statement ("$2n$ is even"), $P(1)$ is true, and $\forall n$, $P(n) \to P(n+1)$, then $\forall n$, $P(n)$ is true.
- **Big-O Notation**. $o, O, \omega, \Omega, \Theta$ and identifying which of these relations hold for two functions.
- **Recurrence Relations**. Solve simple ones by finding a pattern and proving them with induction. More complicated recurrences can be solved using the **Master Theorem** (must memorize)

2.2 Graph Search

- **Representation.** Adjacency list versus adjacency matrix
- **Depth First Search (DFS).** Uses a stack to process nodes, Pre and post order numbers, labels for the edges (tree, forward, back, cross).
  Important Applications. Detecting cycles, topological sorting, finding strongly connected components
- **Breadth First Search (BFS).** Uses a queue to process nodes, can be used to find the shortest path when all edge weights are 1.
- **Dijkstra's Algorithm.** Single source shortest path for non-negative edge weights. Uses a heap or priority queue to process nodes. Does not work when there are negative edge weights (why?).
- **Heaps.** binary heap implementation, operations: DELETEMIN, INSERT, DECREASEKEY, how they are used in Dijkstra's algorithm
- **Bellman-Ford Algorithm.** Single source shortest path for general edge weights, edge relaxation procedure (referred to as *update* in the lecture notes), detecting negative cycles

- **Shortest Path in DAG.** Can be done in linear time via dynamic programming regardless of edge weights.

2.3 Minimum Spanning Trees

- **Basic Properties.** Connected, acyclic, $|E| = |V| - 1$.
- **Cut Property:** The basis for why our MST algorithms are correct, allows us to greedily add edges to a sub-MST.
- **Prim's Algorithm:** implemented in a similar fashion as Dijkstra's, builds MST from a single vertex.
- **Kruskal's Algorithm:** implemented using a union-find data structure, builds MST from lightest edges of the graph
- **Disjoint Forest Data Structure:** maintain a bunch of sets that can be combined efficiently
  Operations: $\textsc{makeset}(x)$, $\textsc{find}(x)$, $\textsc{union}(x, y)$
  Heuristics: Union by Rank, Path Compression

2.4 Greedy

- **Main Idea:** At each step, make a locally optimal choice in hope of reaching the globally optimal solution.
- **Horn Formula:** set all variables to false and greedily set ones to be true when forced to
- **Huffman Coding:** find the best encoding by greedily combining the two least frequently items
- **Set Cover:** greedy approximation algorithm with $O(\log n)$ performance ratio, showed on problem set 3 that we can actually achieve $\Theta(\log n)$ by example.

2.5 Divide and Conquer

- **Main Idea:** Divide the problem into smaller pieces, recursively solve those, and then combine them in the right way.
- **Mergesort and Stoogesort.**
- **Integer Multiplication.** perform 3 multiplications on $n/2$ digit numbers, and then do some additions
- **Strassen's Algorithm.** multiplies two $n \times n$ matrices by doing 7 multiplications on $n/2 \times n/2$ matrices.

2.6 Dynamic Programming

- **Main Idea:** Maintain a lookup table of correct solutions to sub-problems and build up this table in a particular order.
- **Run-time and Space Analysis.** How does one go about analyzing the time and space complexity of a DP algorithm?

- **String Reconstruction.** Tries to find where the first dictionary word ends by checking all possible breaking points.
- **Edit Distance.** Tries all possibilities of INSERT, DELETE, CHANGE on letters that are different.
- **All Pairs Shortest Paths.** Uses the idea that the shortest path to a node must have come via one of the neighboring nodes.
- **Traveling Salesman.** DP can provide better exponential-time solutions to NP-hard problems.

3 Practice Problems

Exercise 1. True or False:

(a) T F $4^{\sqrt{n}} = o(2^n)$

(b) T F $\log_5(n) = \Omega(\log_3(n))$

(c) T F $2^{\sqrt{\log n}} = \omega(n)$

(d) T F Suppose $T(n) = 2T(n/b) + n$. Then, as $b \to \infty$, we eventually get $T(n) = \Theta(1)$.

(e) T F If $T(n) = T(\sqrt[3]{n}) + 1$, then $T(n) = O(\log\log\log n)$.

(f) T F If $T(n) = T(\log n) + 1$, then $T(n) = O(\log^* n)$.

Exercise 2. (Problem Set 2) The input to 2SAT is a logical expression of a specific form: it is the conjunction (AND) of a set of clauses, where each clause is the disjunction (OR) of two literals. (A literal is either a Boolean variable or the negation of a Boolean variable.) For example, the following expression is an instance of 2SAT:

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

A solution to an instance of a 2SAT formula is an assignment of the variables to the values T (true) and F (false) so that all the clauses are satisfied that is, there is at least one true literal in each clause. For example, the assignment $x_1 = T$, $x_2 = F$, $x_3 = F$, $x_4 = T$ satisfies the 2SAT formula above.

Derive an algorithm that either finds a solution to a 2SAT formula, or returns that no solution exists.

Hint: Reduce to an appropriate problem. It may help to consider the following directed graph, given a formula I in 2SAT: the nodes of the graph are all the variables appearing in I and their negations. For each clause $(\alpha \vee \beta)$ in I, we add a directed edge from $\overline{\alpha}$ to β and a second directed edge from $\overline{\beta}$ to α. How can this be interpreted?

Exercise 3. A directed graph $G = (V, E)$ is called semiconnected if for each pair of distinct vertices $u, v \in V$, there is either a path from u to v or a path from v to u. Find an algorithm to determine whether a directed graph is semiconnected. *Hint:* Look at the SCC graph.

Exercise 4. Suppose you are given an adjacency-list representation of an n-vertex graph undirected G with non-negative edge weights in which every vertex has at most 5 incident edges. Give an algorithm that will find the K closest vertices to some vertex v in $O(K \log K)$ time.
Warning: Your solution's run-time must not involve N in any way!

Exercise 5. We are given a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has an associated value $r(u, v)$, which is a real number in the range $0 \leq r(u, v) \leq 1$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u, v)$ as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices.

Exercise 6. Modify Bellman Ford to set dist$[v] = -\infty$ for all vertices v that can be reached from the source via a negative cycle.

Exercise 7. True or False

(a) T F The heaviest edge in a graph is never part of a MST.

(b) T F The shortest-path tree computed by Dijkstra's algorithm is necessarily an MST.

(c) T F Prim's algorithm works with negative weighted edges.

Exercise 8. Given a weighted graph with n vertices and $m \leq n + 10$ edges, show how to compute a minimum spanning tree in $O(n)$ time.

Exercise 9. Each of the following statements is **false**. Provide an explanation or counterexample for each.

(a) Suppose we have a disjoint forest data structure where we use the path compression heuristic. Let x be a node at depth d in a tree, then calling $\textsc{find}(x)$ a total of n times takes $\Theta(nd)$ time.

(b) Using Huffman encoding, the character with the largest frequency is always compressed down to 1 bit.

(c) Consider a variation on the set cover problem where each set is of size at most 3. Then the greedy set cover algorithm discussed in class is optimal. (You can choose how the greedy algorithm breaks ties)

Exercise 10. You are given perfect binary tree with $2^d - 1$ nodes. Suppose that each node x has a weight w_x and that all the weights are distinct for the nodes of the tree. A node is called a *local minimum* if it is smaller than its parent and both its children (if they exist). The root is a local minimum if it is smaller than its two children, and a leaf is a local minimum if it is smaller than its parent. Given a pointer to the root node of this binary tree, give a $O(d)$ algorithm for finding *any* local minimum.

Exercise 11. 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 $0 = d_0 < d_1 < d_2 < \ldots < d_n < d_{n+1} = L$.

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. Provide a time and space analysis of your algorithm.

Exercise 12. Consider the 0-1 knapsack problem: We have n items, each of which has size $s_i > 0$ and value $v_i > 0$. We have a knapsack of size M and want to maximize the sum of the values of the items inside the knapsack without the sum of the sizes of the items exceeding M. Let $V = \sum_{i=1}^{n} v_i$ be the sum of all the values of the items.

(a) Describe a solution that computes the maximum achievable value in $O(nM)$ time.

(b) Describe a solution that computes the maximum achievable value in $O(nV)$ time.