Contents

1 Format and Logistics **2**

2 Topics Covered **2**
2.1 Math Fundamentals . . . 2
2.2 Graph Search . . . 2
2.3 Minimum Spanning Trees . . . 3
2.4 Greedy . . . 3
2.5 Divide and Conquer . . . 3
2.6 Dynamic Programming . . . 3
2.7 Randomized Algorithms and Cryptography . . . 4
2.8 Linear Programming and Network Flows . . . 4
2.9 NP-Completeness . . . 5
2.10 RMQ and Suffix Trees . . . 5

3 Practice Problems **6**
Problem 1. True or False . . . 6
Problem 2. Big O Only . . . 6
Problem 3. Cyclic Sorted Array . . . 7
Problem 4. Counting Permutations . . . 7
Problem 5. Shortest Path LP . . . 8
Problem 6. Dijkstra's Variations . . . 8
Problem 7. Largest Submatrix of 1's . . . 9
Problem 8. Minimum Bottleneck Spanning Tree . . . 9
Problem 9. Building Hospitals . . . 9
Problem 10. MST Bound . . . 10
Problem 11. Well-Nested Brackets . . . 11
Problem 12. Hopfield Network . . . 12
Problem 13. NP-Completeness . . . 13
Problem 14. Two Player Game . . . 13
Problem 15. 3-SAT Random Walk . . . 14
Problem 16. Efficient Waiters . . . 15
Problem 17. Breaking RSA . . . 16
Problem 18. Edge Connectivity . . . 16

1 Format and Logistics

You will have 3 hours to complete an exam whose format is very similar to that of the midterm. It will be 1.5 times the length of the midterm, but you have twice the amount of time, so it should be a little easier.

You can expect short answer questions like true/false, multiple choice, run this algorithm and example/counter-example as well as many long answer questions that ask you to give and analyze an algortihm to solve a problem.

Solutions to all problems will be released at 10pm on Monday May 8, 2017. In the meantime, we strongly suggest that you try to work out as many of these problems as possible.

2 Topics Covered

There will be an emphasis on the material from the second half of class as well as important topics from the first half of the class.

This is a pretty comprehensive list of topics for the entire course, 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 $\forall n$, $P(n) \rightarrow P(n+1)$, 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: MAKESET(x), FIND(x), 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.

2.7 Randomized Algorithms and Cryptography

- **Hashing:** Relationship to the birthday problem, balls and bins.
- **Bloom Filters:** Uses less space than a hash table, but in return also may not give you the correct answer all the time. Has some probability of giving a false positive (i.e. saying something is in the bloom filter when it really isn't).
- **Fingerprinting:** Probabilistic technique for pattern matching, how to efficiently compute all fingerprints by having a sliding window, decreasing failure probability by choosing a prime p.
- **Document Similarity:** Set resemblance, shingling, document sketch, failure probability given say 90 out of 100 entries in the sketch match.
- **Primality Testing:** Fermat's test, a-psuedoprime, witness, Carmichael number, Rabin-Miller testing, non-trivial square root
- **RSA Encryption:** Extended Euclidean algorithm, modular exponentiation (repeated squaring with mod), public key, private key, factoring is hard
- **Random Walks:** Randomized algorithm for 2-SAT by randomly choosing one variable in an unsatisfied clause and flipping it. Recurrence relation for expected number of steps until reaching a boundary.

2.8 Linear Programming and Network Flows

- **Linear Programming:** Objective function and constraints must all be linear functions of the variables, Simplex algorithm. Duality (max flow LP to min cut LP, row player LP to column player LP)
- **Max Flow:** Intuitive formulation with water, reduction to linear programming with conservation, non-negativity, and capacity constraints. If capacities are integral, then there exists an integral max flow solution. Solving matching problems
- **Ford-Fulkerson Algorithm:** Finds augmenting paths in the residual graph until none can be found anymore. Run-time is $O(|E|f^*)$.
- **Min-Cut Max Flow:** The value of the minimum $s-t$ cut in a graph is exactly equal to the value of the maximum $s-t$ flow. How to find this min cut given the residual graph from Ford-Fulkerson.
- **Two-Player Games:** Row player LP and column player LP, solving for the value of the game

2.9 NP-Completeness

- **Definition of P and NP:** NP requires a polynomial-length certificate that verifies the instance to be *yes*. $P \subseteq NP$
- **Polynomial-time Reductions:** Show that A is easy by reducing A to B, which is known to be easy. Show that A is hard by reducing B, which is known to be hard, to A.
- **NP-Completeness:** Is in NP and all other problems in NP reduce to it. Circuit SAT is NP-complete. Circuit SAT $\rightarrow$ 3-SAT, 3-SAT $\rightarrow$ Integer Linear Programming, 3-SAT $\rightarrow$ Independent Set, Independent Set $\rightarrow$ Vertex Cover, Vertex Cover $\leftrightarrow$ Maximum Clique
- **Local Search:** Start with a set of possible solutions and an easy to compute neighbor function. Try to minimize a cost function by moving to neighbors in a particular way.
- **Approximations:** $O(\log n)$, 2-approximation for vertex cover, randomized and local search max cut approximation, 2-approximation for Euclidean travelling salesman probelm, LP relaxation, Max-SAT with randomness,

2.10 RMQ and Suffix Trees

- **Range Minimum Query:** Naive solutions, dynamic programming solution to RMQ, reduction chain: RMQ $\rightarrow$ LCA $\rightarrow$ RMQ ± 1.
- **Suffix Trees:** how to construct, pattern matching, longest common extension, maximal palindromes,

3 Practice Problems

Problem 1. True or False

Answer True or False to the following questions:

(a) If x is a $2-$psuedoprime, then x is also a 4-psedoprime.

(b) Let x be an integer and write $x = 2^t \cdot u$ with u odd. Then, if $x^{2^i u} = 1$ but $x^{2^{i+1} u} \neq -1, 1$, then x is composite.

(c) When using both the Union by Rank and Path Compression heuristics, each FIND operation takes $O(\log^* n)$ time.

(d) Kruskal's algorithm is better for dense graphs, while Prim's algorithm with a Fibonacci heap is better for sparse graphs. (Recall that Fibonacci heap gives $O(1)$ insert and $O(\log |V|)$ delete-Min).

(e) If an iteration of the Ford-Fulkerson algorithm on a network places flow 1 through an edge (u, v), then in every later iteration, the flow through (u, v) is at least 1.

Problem 2. Big O Only

Is it possible to find two functions $f(n), g(n)$ such that $f(n) = O(g(n))$, but $f(n) \neq o(g(n)), f(n) \neq \Omega(g(n))$? If so, give an example. If not, give a proof.

Problem 3. Cyclic Sorted Array

You're given an array A of n distinct numbers that is sorted up to a cyclic shift, i.e. $A = (a_1, a_2, \ldots, a_n)$ where there exists a k such that $a_k < a_{k+1} < \ldots < a_n < a_1 < \ldots < a_{k-1}$, but you don't know what k is. Give an efficient algorithm to find the largest number in this array.

Problem 4. Counting Permutations

How many permutations of $\{1, 2, \ldots, N\}$ have exactly K inversions? An inversion is a pair of numbers i, j in the permutation such that $i > j$ but i comes before j. For example, for $N = 3$ and $K = 1$ the answer is 2: $\{2, 1, 3\}$ and $\{1, 3, 2\}$.

Problem 5. Shortest Path LP

Given a weighted, directed graph $G = (V, E)$, with weight function w mapping edges to positive real-valued weights, a source vertex s, and a destination vertex t. Show how to compute the value $d[t]$, which is the weight of a shortest path from s to t, by linear programming.
Hint: Have variables $x_{(u,v)}$ for each edge and think about constraints similar to that of flow.

Problem 6. Dijkstra's Variations

In lecture, we showed that Dijkstra's algorithm may not work correctly if negative edge weights are present. In this problem. Let $G = (V, E)$ be a weighted, directed graph with no negative-weight cycles. For each of the following property of G, give an algorithm that finds the length the shortest path from s to all other vertices in V. In both cases, your algorithm should have the same run-time as that of Dijkstra's.

(a) G only has one negative edge. Give an algorithm to find length of the shortest path from s to all other vertices in V.

(b) The edges leaving from the source s may have negative weights, but all other edges in E have non-negative weights,

Problem 7. Largest Submatrix of 1's

You are given a matrix M with m rows and n columns consisting of only 0's and 1's. Give an efficient algorithm for find the maximum size square sub-matrix consisting of only 1's.

Problem 8. Minimum Bottleneck Spanning Tree

Let $G = (V, E)$ be an undirected graph with weights w_e on the edges. A **minimum bottleneck spanning tree** of G is a spanning tree such that the weight of the heaviest edge is as small as possible. Note that for minimum bottleneck spanning trees, we do not care about the sum of the weights of the chosen edges–only the maximum edge.

(a) Show that any minimum spanning tree is necessarily a minimum bottle neck spanning tree

(b) Give an example of a graph and a minimum bottle neck spanning tree that is not a minimum spanning tree.

Problem 9. Building Hospitals

Suppose we are given a set of cities represented by points in the plane, $P = \{p_1, \ldots, p_n\}$. We will choose k of these cities in which to build a hospital. We want to minimize the maximum distance that you have to drive to get to a hospital from any of the cities p_i. That is, for any subset $S \subset P$, we define the cost of S to be

$$\max_{1 \le i \le n} \text{dist}(p_i, S) \quad \text{where} \quad \text{dist}(p_i, S) = \min_{s \in S} \text{dist}(p_i, s).$$

This problem is known to be NP-hard, but we will find a 2-approximation. The basic idea: Start with one hospital in some arbitrary location. Calculate distances from all other cities — find the "bottleneck city," and add a hospital there. Now update distances and repeat.

(a) Give a precise description of this algorithm, and prove that it runs in time $O(nk)$.

(b) Prove that this is a 2-approximation

Problem 10. MST Bound

Suppose that we are given a graph with the following guarantee: for any two disjoint sets S, T of vertices, there exists some edge between a vertex in S and one in T with length no more than $\frac{1}{\max(|S|,|T|)}$. Find a bound on the size of the minimum spanning tree for this graph, in terms of the number of vertices. *Hint:* Think Prim's algorithm.

Problem 11. Well-Nested Brackets

There are four types of brackets: (,), <, and >. We define what it means for a string made up of these four characters to be *well-nested* in the following way:

(a) The empty string is well-nested.

(b) If A is well-nested, then so are <A> and (A).

(c) If S, T are both well-nested, then so is their concatenation ST.

For example, (), <>, (()), (<>), ()<>, and ()<()> are all well-nested. Meanwhile, (, <,),)(, (<)>, and (<(> are not well-nested.

Devise an algorithm that takes as input a string s of length n made up of these four types of characters. The output should be the length of the shortest well-nested string that contains s as a subsequence (not sub-string). For example, if (<(> is the input, then the answer is 6; a shortest string containing s as a subsequence is ()<()>.

Problem 12. Hopfield Network

A Hopfield Network is an undirected graph where each node v is assigned a state $s(v) \in \{-1, 1\}$. Every edge (i, j) has an integer weight w_{ij}.

For a given configuration of node state assignments, we call an edge (u, v) *good* if $w(u, v)s(u)s(v) < 0$ and *bad* otherwise. A node u is satisfied if the sum of the weights of the good edges incident to u is larger than the sum of the weights of the bad edges incident to u, i.e.

$$\sum_{v:(u,v)\in E} w(u, v)s(u)s(v) \leq 0$$

A configuration is *stable* if all nodes are satisfied. In this problem, we will use local search to find a stable configuration.

(a) Define a cost function f.

(b) Define a neighborhood function N.

(c) Explain why this algorithm will always succeed in finding a stable configuration. Remember that there are two parts to succeeding: you must terminate and must be correct.

Problem 13. NP-Completeness

Show that the following 2 problems are NP-complete:

(a) **Subgraph isomorphism**: Consider graphs $G = (V, E)$, $H = (V_2, E_2)$. Does G contain a subgraph (V_1, E_1) which is isomorphic to H?

(An isomorphism of graphs (V_1, E_1) and (V_2, E_2) is a 1-1 function $f : V_1 \to V_2$ such that the map $(u, v) \mapsto (f(u), f(v))$ is a 1-1 function $E_1 \to E_2$.)

(b) **4-Exact-cover**: Given a finite set X of size $4q$ and a collection $\mathcal{C}$ of 4-element subsets of X, does $\mathcal{C}$ contain an **exact cover** (i.e. a partition) of X?

You may use without proof that 3-EXACT-COVER is NP-complete.

Problem 14. Two Player Game

Consider the following zero-sum game, where the entries denote the payoffs of the row player:

$$\begin{pmatrix} 2 & -4 & 3 \\ 1 & 3 & -3 \end{pmatrix}$$

Write the column player's maximization as an LP. Then, write the dual LP, which is the row player's minimization problem.

Problem 15. 3-SAT Random Walk

In lecture, we saw a random walk algorithm for 2-SAT that ran in $O(n^2)$ time. In this problem, we will attempt to extend that algorithm for 3-SAT.

(a) Consider a random walk on the vertices $\{0, 1, 2, n-1, n\}$ where we start at vertex 1. At each step, we go up with probability $\frac{1}{3}$ and down with probability $\frac{2}{3}$. When we are at 0, we will always move to 1. Let T_i be the expected number of moves until we reach n starting from vertex i. Verify that the formula $T_i = 4(2^n - 2^i) - 3(n - i)$ is correct.

(b) Suppose we attempted to use the same random walk algorithm for 2-SAT on 3-SAT. That is, we start with all variables set to false, and for each unsatisfied clause, we randomly pick a variable and flip it. Explain why we do not get a polynomial-time algorithm for 3-SAT like we did for 2-SAT.

Problem 16. Efficient Waiters

A restaurant has n tables and 2 waiters. Table i is located at coordinate (x_i, y_i) in the restaurant for $i = 1, 2, \ldots, n$. At the beginning, one waiter is at table 1 and the other waiter is at table 2. We will then receive a sequence of m queries. Each query is simply a number in $\{1, 2, 3, \ldots, n\}$ representing that the customer at that table needs some kind of service. Hence, one of the two waiters needs to be service that table. The waiters are complaining of walking too much and so the restaurant has hired you to help them be more efficient.

Suppose an omniscient being has told you in advance the order in which the tables would need servicing. Let $t_1, t_2, \ldots, t_m$ be the sequence of queries (i.e. tables that need servicing). The goal is to minimize the total walking distance of the two waiters, where the distance between two tables is the standard Euclidean distance. Assume that there is enough time between the queries for whichever of the 2 waiters is chosen to walk to the table and service it.

(a) Consider the following greedy algorithm. In order to service table t_i, we will have whichever waiter is closer to service that table. Give an example to show that this greedy algorithm is not optimal.

(b) Give a $O(n^2 m)$ time and $O(n^2)$ space solution that finds the value of the shortest total walking distance.

Problem 17. Breaking RSA

(a) Show that RSA is multiplicative. That is, if C_1 is the encryption of M_1 and C_2 is the encryption of M_2, then C_1C_2 is the encryption of M_1M_2.

(b) Let (n, e) be a public key of RSA. Show that if we have an efficient algorithm A which decrypts 1 percent of the messages, we can build an efficient randomized algorithm that decrypts every message with high probability.

Hint: If you need to decrypt C, pick a random M_0, compute its encryption C_0 and try decrypting CC_0.

Problem 18. Edge Connectivity

The edge connectivity of an undirected graph is the minimum number k of edges that must be removed to disconnect the graph. For example, the edge connectivity of a tree is 1, and the edge connectivity of a cyclic chain of vertices is 2. Let $V = \{v_0, v_1, \ldots, v_n\}$. For each i, consider the flow network F_i where v_0 is the source, v_i is the sink, and both (u, v) and (v, u) have capacity 1 for each $(u, v) \in E$. Let f_i be the total flow of F_i. Prove that the edge connectivity is $\min_i f_i$.