Qualifying Examination

Theoretical Computer Science

Monday, March 2, 2020

Part I: Algorithms

Name	

Problem	Maximum Points	Points Earned	Grader
1	25		
2	25		
3	25		
4	25		
Total	100		

Instructions:

1. This is a closed book exam.
2. The exam is from 9am–4:30pm and has four problems of 25 points each. Read all the problems carefully to see the order in which you want to tackle them.
3. Write clearly and concisely. You may appeal to some standard algorithms/facts from text books unless the problem explicitly asks for a proof of that fact or the details of that algorithm.
4. If you cannot solve a problem, to get partial credit write down your main idea/approach in a clear and concise way. For example you can obtain a solution assuming a clearly stated lemma that you believe should be true but cannot prove during the exam. However, please do not write down a laundry list of half baked ideas just to get partial credit.

May the force be with you.

Problem 1: Consider the following problem: given a binary string $w = a_1 a_2 \cdots a_n \in \{0,1\}^*$, decide whether w contains 00000 as a substring (i.e., whether w contains five consecutive 0's). There is an obvious $O(n)$-time algorithm. You will investigate the hidden constant factor, concerning the number of bits that need to be examined (i.e., the number of a_i's that need to be evaluated) to solve this problem.

1. (10/100 points) First argue that every deterministic algorithm solving this problem needs to evaluate $\Omega(n)$ bits in the worst case.

2. (30/100 points) Prove that for a sufficiently large constant c, for any string $a_1 \cdots a_c$ of length c that does not contain 00000, there must exist two indices $i, j \in \{1, \ldots, c\}$ with $a_i = a_j = 1$ and $2 \leq |i - j| \leq 5$.

3. (60/100 points) Design a Las Vegas randomized algorithm solving this problem (with zero error probability) that evaluates an expected $\alpha n + o(n)$ number of bits for any input, for some constant α strictly smaller than 1. (You do not need to optimize the constant α.)

Problem 2: Suppose you are given a directed graph $G = (V, E)$, where each edge has a *label* from some fixed finite alphabet Σ. The label of any walk in G is a string in Σ^*, formed by concatenating the labels of its edges in order. A *palindrome walk* in G is a walk in G whose label is equal to its reversal. Your task is to find a palindrome walk in G with maximum length, if such a walk exists. (Such a walk might not exist; consider a directed cycle where every edge has the same label!)

1. (20/100 points) Lets first consider the special case where G is acyclic. Describe an algorithm to find a longest palindrome walk in an edge-labeled dag.

2. (60/100 points) Describe an algorithm that either finds the longest palindrome walk in G, or correctly reports that there are arbitrarily long palindome walks in G.

3. (20/100 points) Prove that finding the longest palindrome *path* in G is NP-hard. (A path is a walk that never repeats vertices.)

Problem 3: Let $G = (V, E)$ be a edge-weighted directed graph with $c : E \to \mathbb{Z}_+$ denoting the edge costs/weights. Give two distinct nodes s, t we wish to find two edge disjoint paths from s to t.

1. (30/100 points) Suppose we want to find two edge-disjoint paths such that each path is of cost at most a given bound B. Prove that deciding the existence of such paths is NP-Complete.

2. (70/100 points) Consider a relaxed version where we minimize the sum of the costs of the edges in the two disjoint paths. In the literature there is an algorithm called Surballe's algorithm. It consists of two iterations. In the first iteration it finds a shortest path s-t P_1. It then reverses the edges in P_1, negates the costs on these edges, and finds a second shortest path P_2. It adds up the costs of the two paths. Prove that this algorithm is correct and can be implemented to run in polynomial time. In particular how can you compute the shortest path in the second iteration when there are some negative length edges?

Problem 4: Given an undirected graph $G = (V, E)$ with positive lengths on its edges, a subgraph $H = (V, E_H)$ is a *t-spanner*, if for any pair of vertices u and v of G, we have that $d_G(u, v) \leq d_H(u, v) \leq t d_G(u, v)$ where $d_G(u, v)$ and $d_H(u, v)$ are the distances between u and v in G and H respectively.

The greedy spanner construction is an extension of Kruskal's algorithm. Let $e_1, \ldots, e_m$ be the edges of G sorted in increasing order of lengths. The algorithm starts with an empty graph H over the vertices of G. In the i'th iteration, the algorithm checks if $d_H(u_i, v_i) > t d_G(u_i, v_i)$, where $e_i = u_i v_i$. If so, the algorithm adds the edge e_i to H otherwise it discards e_i; in both cases it moves to the next iteration.

1. Prove that the minimum spanning tree is a $(n-1)$-spanner.

2. Prove that the greedy algorithm outputs a t-spanner.

3. Suppose G has unit weights on its edges (this holds for the subsequent parts). Prove that the shortest cycle in the computed graph H is of length at least $t + 2$.

4. Prove that a graph G with more than (say) $10n^{1+1/t}$ edges must contain a cycle with $\leq 2t + 2$ edges. (Hint: Initially assume all the vertices have the same degree, and the graph is connected.)

5. Prove that the t-spanner computed above has at most $O(n^{1+2/t})$ edges.