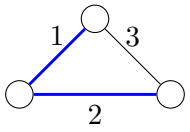
CSE421: Design and Analysis of Algorithms April 25, 2019

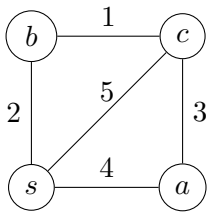
Homework 4

Shayan Oveis Gharan Due: May 2, 2019 at 5:00 PM

P1) Prove or disprove the following claim: Suppose we are given a graph G where the cost of edge e is c_e where all c_e's are positive. For simplicity, assume that all c_e's are distinct. Suppose T is the minimum spanning tree which is the output of the Kruskal's algorithm. Now suppose we update the cost of every edge e to c_e^2. Then, T remains the minimum spanning tree of G. For example, in the following graph the tree shown in blue is the output of the Kruskal's algorithm. In this case if we update the costs to $1, 4, 9$ respectively the blue tree remains a minimum spanning tree.



P2) Consider the following variant of the shortest path problem: Suppose we are given a weighted connected undirected graph G and a source vertex s. For simplicity you can assume all c_e's are different and positive. Define the *bottleneck* of a path P from s to v to be the cost of the maximum edge along this path. Design a polynomial time algorithm that outputs the bottleneck of minimum bottleneck path from s to all vertices of G. For example, in the following graph, the bottleneck of the minimum bottleneck from s to a is s, b, c, a with bottleneck 3, to b is s, b with bottleneck 2 and to c is s, b, c with bottleneck also 2. So, you can just output $3, 2, 2$.



P3) Suppose you are choosing between the following three algorithms:

a) Algorithm A solves the problem by dividing it into six subproblems of half the size, recursively solves each subproblem, and then combines the solution in linear time.

b) Algorithm C solves the problem by dividing it into sixteen subproblems of one fourth the size, recursively solves each subproblem, and then combines the solutions in quadratic time.

c) Algorithm B solves problems of size n by recursively solving two subproblems of size $n-3$, and then combines the solution in constant time.

What are the running times of each of these algorithms? To receive full credit, it is enough to write down the running time.

P4) Suppose there is a list of n integers $x_0, \ldots, x_{n-1}$ hidden from us, and n is a power of 2. We have access to this list through an oracle, A. For every integer $0 \leq k < n$ and any power of two, say 2^a, we can ask the oracle $A(k, 2^a)$ and it will spit out sum of the numbers in the interval $[k, k+2^a-1]$, i.e., $\sum_{i=k}^{k+2^a-1} x_i$. Design an algorithm that given a, b outputs sum of the numbers in the interval $[a, b-1]$ and makes at most $O(\log n)$ calls to the oracle. For example, for $n = 16$, and $a = 3, b = 9$ your algorithm can just output $A(8,1)+A(0,8)-A(0,2)-A(2,1)$. This works because

$$\begin{aligned} A(8,1) &= x_8 \\ A(0,8) &= x_0 + x_1 + \cdots + x_7 \\ A(0,2) &= x_0 + x_1, \\ A(2,1) &= x_2. \end{aligned}$$

P5) **Extra Credit** The spanning tree game is a 2-player game. Each player in turn selects an edge. Player 1 starts by deleting an edge, and then player 2 fixes an edge (which has not been deleted yet); an edge fixed cannot be deleted later on by the other player. Player 2 wins if he succeeds in constructing a spanning tree of the graph; otherwise, player 1 wins.

The question is which graphs admit a winning strategy for player 1 (no matter what the other player does), and which admit a winning strategy for player 2.

Show that player 1 has a winning strategy if and only if G does not have two edge-disjoint spanning trees. Otherwise, player 2 has a winning strategy.