(CS1.301) Advanced Algorithm Design (Monsoon 2023)

Practice Problem Set 2

Instructions:

- Discussions amongst the students are not discouraged.
- Referring sources other than the lecture notes and textbooks is discouraged as the corresponding solutions available on the internet need not be accurate.
- Please attend tutorials to ensure that you understand the problem correctly.

Question 1 Let the graph G have n vertices and m edges whose edge weights are all distinct. Give an algorithm to decide whether a given edge e is contained in a minimum spanning tree of G without actually constructing the Minimum Spanning Tree.

Question 2 [Retracted]

Question 3 Let $G = (V, E)$ be an undirected graph with edge costs $c_e > 0$ on the edges $e \in E$. Let T be a Minimal Spanning Tree in G. Let a new edge (v, w) with edge weight c, be added to G. Give an efficient algorithm to test if T remains a Minimal Spanning Tree after addition of (v, w) to G (and not to T). If T is no longer the minimal spanning tree, then give an efficient algorithm to update the tree T to the new Minimum Spanning Tree.

Question 4 Given a list of n natural numbers $d_1, d_2, ..., d_n$, show how to decide in polynomial time whether there exists an undirected (simple) graph $G = (V, E)$ whose node degrees are precisely the numbers $d_1, d_2, ..., d_n$

Question 5 Given a list of n jobs $J_1, J_2, ..., J_n$ with processing times $p_1, p_2, ..., p_n$ and weights $w_1, w_2, ..., w_n$. Starting from $t = 0$, the cost for completing a job J_i is given by w_i**(total time from* $t = 0$ *to the time at which job* J_i *finishes).* Give an algorithm to find the order to perform the jobs with minimum total cost.

Question 6 [Updated]

Suppose we are given a set U of objects labeled $p_1, p_2, \ldots, p_n$. For each pair p_i and p_j, we have a numerical distance $d(p_i, p_j)$. We futher have the property that for all $1 \leqslant i \leqslant n$, $d(p_i, p_i) = 0$ and for all $1 \leqslant i \neq j \leqslant n$, $d(p_j, p_i) = d(p_i, p_j) > 0$.

For a given parameter k as input, k-clustering of U is a partition of U into k nonempty sets $C_1, C_2, \ldots, C_k$. Spacing of a k-clustering is defined to be the minimum distance between any pair of points lying in different clusters. That is,

$$\text{Spacing}(C_1, C_2, \ldots, C_k) = \min_{1 \leqslant u \neq v \leqslant k} \{\min\{d(p, p') \mid p \in C_u \text{ and } p' \in C_v\}\}.$$

Given that we want points in different clusters to be far apart from one another, a natural goal is to seek the k-clustering with the maximum possible spacing. In other words, we want to find the partition of U into k non-empty sets that maximizes the following expression.

$$\max_{U = C_1 \sqcup C_2 \sqcup \ldots \sqcup C_k} \{\text{Spacing}(C_1, C_2, \ldots, C_k)\}$$

The question now becomes the following – how can we efficiently find the one that has maximum spacing?

Question 7 Let $G = (V, E)$ be a connected graph with m edges, n vertices with positive distinct edge costs. Let $T = (V, E)$ be a spanning tree of G; we define the bottleneck edge of T as the edge with maximum weight.

We define the *minimum-bottleneck spanning tree* as the spanning tree T with the minimum bottleneck edge. Prove or give a counter-example of the following:

(a) Is every minimum-bottleneck tree a minimum spanning tree of G?

(b) Is every minimum-spanning tree a minimum bottleneck tree of G?

Question 8 Let $G = (V, E)$ be a graph with n nodes in which each pair of nodes is joined by an edge. There is a positive weight $w_{i,j}$ on each edge (i, j); and we will assume these weights satisfy the triangle inequality $w_{i,k} \leqslant w_{i,j} + w_{j,k}$. For a subset $V' \subseteq V$, we will use $G[V']$ to denote the subgraph (with edge weights) induced on the nodes in V'.

We are given a set $X \subseteq V$ of k terminals that must be connected by edges. We say that a Steiner tree on X is a set Z so that $X \subseteq Z \subseteq V$, together with a spanning subtree T of $G[Z]$. The weight of the Steiner tree is the weight of the tree T. Show that the problem of finding a minimum-weight Steiner tree on X can be solved in time $n^{O(k)}$.