# Fourth Homework Assignment

Write the solution to each problem on a single page. The deadline for handing in solutions is October 30.

**Problem 1.** ($20 = 10 + 10$ points). Consider a free tree and let $d(u, v)$ be the number of edges in the path connecting $u$ to $v$. The *diameter* of the tree is the maximum $d(u, v)$ over all pairs of vertices in the tree.

(a) Give an efficient algorithm to compute the diameter of a tree.

(b) Analyze the running time of your algorithm.

**Problem 2.** (20 points). Design an efficient algorithm to find a spanning tree for a connected, weighted, undirected graph such that the weight of the maximum weight edge in the spanning tree is minimized. Prove the correctness of your algorithm.

**Problem 3.** ($7 + 6 + 7$ points). A weighted graph $G = (V, E)$ is a *near-tree* if it is connected and has at most $n + 8$ edges, where $n$ is the number of vertices. Give an $\mathrm{O}(n)$-time algorithm to find a minimum weight spanning tree for $G$.

**Problem 4.** ($10 + 10$ points). Given an undirected weighted graph and vertices $s, t$, design an algorithm that computes the number of shortest paths from $s$ to $t$ in the case:

(a) All weights are positive numbers.

(b) All weights are real numbers.

Analyze your algorithm for both (a) and (b).

**Problem 5.** ($20 = 10 + 10$ points). The *off-line minimum problem* is about maintaining a subset of $[n] = \{1, 2, \ldots, n\}$ under the operations INSERT and EXTRACTMIN. Given an interleaved sequence of $n$ insertions and $m$ min-extractions, the goal is to determine which key is returned by which min-extraction. We assume that each element $i \in [n]$ is inserted exactly once. Specifically, we wish to fill in an array $E[1..m]$ such that $E[i]$ is the key returned by the $i$-th min-extraction. Note that the problem is *off-line*, in the sense that we are allowed to process the entire sequence of operations before determining any of the returned keys.

(a) Describe how to use a union-find data structure to solve the problem efficiently.

(b) Give a tight bound on the worst-case running time of your algorithm.