Problem Set 10

Data Structures and Algorithms, Fall 2022

Due: December 1.

Problem 1

Devise an algorithm which takes as input a directed graph $G = (V, E)$, and determines whether or not there is a vertex $s \in V$ from which all other vertices are reachable. Your algorithm should output one such vertex if such vertices exist. To get full credit, your algorithm should have runtime $O(|V| + |E|)$.

Problem 2

A directed graph $G = (V, E)$ is "weak-connected" if, for every pair of vertices u and v, either u is reachable from v or v is reachable from u (or both).

(a) Give an example of a directed acyclic graph with a unique source that is *not* weak-connected.

(b) Devise an algorithm with $O(|V| + |E|)$ runtime to determine whether a given directed *acyclic* graph is weak-connected.

(c) Devise an algorithm with $O(|V| + |E|)$ runtime to determine whether a given directed graph is weak-connected.

Problem 3

(a) Let $G = (V, E)$ be an arbitrary connected graph with weighted edges. Prove that for any cycle in G, the minimum spanning tree of G does *not* include the maximum-weight edge in that cycle.

(b) Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a safe edge for A crossing $(S, V - S)$. Professor Bacon claims (u, v) is a light edge for the cut. Do you agree with Professor Bacon? You need to justify your answer.

(c) Professor Bacon proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a graph $G = (V, E)$, partition the set V of vertices into two sets V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1. Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2. Recursively solve a minimum-spanning-tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge in E that crosses the cut (V_1, V_2), and use this edge to unite the resulting two minimum spanning trees into a single spanning tree. Do you think Professor Bacon's algorithm is correct? You need to justify your answer.

Problem 4

In this problem, we give pseudocode for three different algorithms. Each one takes a connected graph and a weight function as input and returns a set of edges T. For each algorithm, either prove that T is always a minimum spanning tree or prove that T is not necessarily a minimum spanning tree.

MAYBE-MST-A(G, w)

```
1: Sort the edges into nonincreasing order of edge weights w.
2: T ← E.
3: for (each edge e taken in nonincreasing order by weight) do
4:     if (T − {e} is a connected graph) then
5:         T ← T − {e}.
6: return T.
```

MAYBE-MST-B(G, w)

```
1: T ← ∅.
2: for (each edge e taken in arbitrary order) do
3:     if (T ∪ {e} has no cycles) then
4:         T ← T ∪ {e}.
5: return T.
```

MAYBE-MST-C(G, w)

```
1: T ← ∅.
2: for (each edge e taken in arbitrary order) do
3:     T ← T ∪ {e}.
4:     if (T has a cycle c) then
5:         Let e′ be the maximum-weight edge on c.
6:         T ← T − {e′}.
7: return T.
```

Problem 5

You are given a graph $G = (V, E)$ with positive edge weights, and an MST $T = (V, E')$ with respect to these weights. You may assume G and T are given as adjacency lists. Now suppose the weight of a particular edge $e \in E$ is modified from $w(e)$ to a new value $\hat{w}(e)$. You wish to quickly update the MST T to reflect this change, without recomputing the entire tree from scratch. There are four cases. In each case give an $O(|V| + |E|)$ time algorithm for updating the tree. You do *not* need to argue the correctness of your algorithms.

(a) $e \notin E'$ and $\hat{w}(e) > w(e)$.

(b) $e \notin E'$ and $\hat{w}(e) < w(e)$.

(c) $e \in E'$ and $\hat{w}(e) < w(e)$.

(d) [Bonus Question] $e \in E'$ and $\hat{w}(e) > w(e)$.

Problem 6

Consider a weighted version of the activity scheduling problem discussed in class, where different activities offer different number of credits (totally unrelated to the duration of the activity). Your goal is now to choose a set of non-conflicting activities that give you the largest possible number of credits, given arrays of start times, end times, and credits as input.

(a) Prove that the greedy algorithm discussed in class does not always return an optimal schedule.

(b) Devise an algorithm that always computes an optimal schedule, and analyze its runtime. Try to make it as efficient as possible. *(Hint: Your algorithm will not be greedy.)*