CS 3510: Homework 4

Due on Tuesday, June 29, 11:59pm via Gradescope.
Late submission with no penalty until Wednesday, June 30, 11:59am.

Professor Brito

CS 3510 Staff

Instructions and black-box list.

For the graded problems, you are allow to use the algorithms from class as black-boxes without further explanation. These include

- DFS (outputs connected components,a topological sorting on a DAG. You also have access to the pre and post arrays!), BFS and the Explore subroutine (outputs a path between the input vertex v and any other vertex in the same connected component).
- Dijkstra's algorithm to find the shortest distance from a source vertex to all other vertices and a path can be recovered backtracking over the pre labels.
- Bellman-Ford and Floyd-Warshall to compute the shortest path when weights are allowed to be negative.
- SCCs which outputs the strongly connected components, and the metagraph of connected components.
- Kruskal's and Prim's algorithm to find a MST.

When using a black-box, make sure you clearly describe which input you are passing to it and how you use the output from or take advantage of the data structures created by the algorithm. To receive full credit, your solution must:

- Include the description of your algorithm in words (no pseudocode!).
- Explain the correctness of your design.
- State and analyse the running time of your design (you can cite and use the running time of black-boxes without further explanations).

Unless otherwise indicated, black-box graph algorithms should be used without modification.

Suggested reading.

1. Chapter 3 (for DFS, topological sorting and SCCs).
2. Chapter 4 (for distances and paths on graphs, BFS and Dijkstra's, Bellman-Ford and Floyd-Warshall algorithms).
3. Chapter 5.1 (for MST).

Suggested problems.

The numeration of the problems corresponds to the 2006 edition of the book.
You do not need to turn in these problems.

1. Chapter 3: $3.8, 3.18, 3.23, 3.26$.
2. Chapter 4: $4.4, 4.8, 4.9, 4.11, 4.12, 4.18, 4.21$.
3. Chapter 5: $5.1, 5.2, 5.3, 5.5, 5.6, 5.7, 5.9, 5.20$.

Practice problem.

(Exam 3, Fall 2020) Let $G = (V, E)$ be an undirected, connected, weighted graph, and let F be a subgraph of G such that it is a forest (i.e. F does not contain any cycles). Design an efficient algorithm to find a spanning tree that contains all the edges of F, and that has minimum cost among all spanning trees containing F.

DO NOT USE PSEUDOCODE
A solution using pseudocode will receive a zero, even if it is correct.

Problem 1

A directed graph $G(V, E)$ is said to be psuedo-connected if for every pair of vertices $s, t \in V$ with $s \neq t$, there is a path from s to t or a path from t to s. Design an algorithm that runs in $O(V + E)$ time that determines whether a given direcxted graph $G = (V, E)$ is psuedo-connected.

Problem 2

Consider an undirected graph $G = (V, E)$ where the weights of all the edges are non-negative and distinct. Each of the following two parts outlines a strategy for finding an MST in a graph with unique edge weights:

1. We use a **divide and conquer** strategy to find the MST. Divide V arbitrarily into disjoint sets V_1 and V_2, each of size roughly $\frac{V}{2}$. Define $G(V_1, E_1)$ as the graph with the set of edges E_1 where E_1 is the subset of E with both endpoints in V_1. We define $G(V_2, E_2)$ in a similar fashion, where E_2 is the subset of E with both endpoints in V_2. We then *recrusively* find the MSTs for G_1 and G_2, labeled as T_1 and T_2 respectively. Lastly, we find the edge with the least weight that crosses the cut between V_1 and V_2 and add that edge to the final MST.

2. We use a **cycle-breaking** strategy to find the MST. The strategy runs iterations until it has a spanning tree (while $|E| > |V| - 1$. In each iteration, the strategy finds some simple cycle in the graph and removes the edge with the maximum weight in that cycle. The strategy leaves us with an MST.

For each of the two strategies outlined above, prove (or disprove) whether or not the strategy returns the correct MST. If the strategy is correct, explain why. If the strategy is incorrect, provide a specific counterexample. *Note that you do not need to provide a design, just prove or disprove that the given ones work. Runtime analysis is not necessary.*