CSCI 3104 Spring 2018

Problem Set 9

Merola, Michael

06/04/1998

Problem Set 9

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pylab as plt
        %matplotlib inline
```

Problem 1

(10 pts) Let $G = (V, E)$ be a graph with an edge-weight function w, and let the tree $T \subseteq E$ be a minimum spanning tree on G. Now, suppose that we modify G slightly by decreasing the weight of exactly one of the edges in $(x, y) \in T$ in order to produce a new graph G'. Here, you will prove that the original tree T is still a minimum spanning tree for the modified graph G'.

To get started, let k be a positive number and define the weight function w' as

$$w'(u, v) = \begin{cases} w(u, v) & \text{if } (u, v) \neq (x, y) \\ w(u, v) - k & \text{if } (u, v) = (x, y) \end{cases}$$

Now, prove that the tree T is a minimum spanning tree for G', whose edge weights are given by w'.

I don't know

Problem 2

(20 pts) Professor Snape gives you the following unweighted graph and asks you to construct a weight function w on the edges, using positive integer weights only, such that the following conditions are true regarding minimum spanning trees and single-source shortest path trees:

```
- The MST is distinct from any of the seven SSSP trees.
- The order in which Jarnıḱ/Prim’s algorithm adds the safe edges is differen
t from the order in which Kruskal’s algorithm adds them.
- Boruv̌ka’s algorithm takes at least two rounds to construct the MST.
```

Justify your solution by (i) giving the edges weights, (ii) showing the corresponding MST and all the SSSP trees, and (iii) giving the order in which edges are added by each of the three algorithms. (For Borůvka’s algorithm, be sure to denote which edges are added simultaneously in a single round.)

I don't know

Problem 3

I don't know

Problem 4

(40 pts) Bidirectional breadth-first search is a variant of standard BFS for finding a shortest path between two vertices $s, t \in V(G)$. The idea is to run two breadth-first searches simultanesouly, one starting from s and one starting from t, and stop when they “meet in the middle” (that is, whenever a vertex is encountered by both searches). "Simultaneously" here doesn’t assume you have multiple processors at your disposal; it’s enough to alternate iterations of the searches: one iteration of the loop for the BFS that started at s and one iteration of the loop for the BFS that started at t.

As we’ll see, although the worst-case running time of BFS and Bidirectional BFS are asymptotically the same, in practice Bidirectional BFS often performs significantly better.

Throughout this problem, all graphs are unweighted, undirected, simple graphs.

I don't know