CS 420\520 Winter 2019
HW 2

1. ***Reachability***. Let G = (V, E) be a directed graph in which each vertex u ∈ V is labeled with a unique integer L(u) from the set { 1, 2, …|v|}. For each vertex u ∈ V, let reach(u) = { v ∈ V: there exist a path from u to v } be the set of vertices that are reachable from u. Define min(u) to be the vertex in reach(u) whose label is minimum, i.e. min(u) = L(v) = min{ L(w) : w ∈ reach(u) }. Give an O(V+E) time algorithm that computes min(u) for all vertices u ∈V.

2. The police department in the city of Computopia has made all streets one-way. The mayor contends that there is still a way to drive legally from any intersection in the city to any other intersection, but the opposition is not convinced. A computer program is needed to determine whether the mayor is right. However, the city elections are coming up so there is just enough time to run a linear-time algorithm.

a) Formulate this problem graph-theoretically, and explain why it can indeed be solved in linear time.

b) Suppose it now turns out that the mayor's original claim is false. She next claims something weaker: if you start driving from town hall, navigating one-way streets, then no matter where you reach, there is always way to drive legally back to the town hall. Formulate this weaker property as a graph-theoretical problem and carefully show how it too can be checked in linear time.

3. Let G = (V,E) be a connected weighted graph, and let T be a minimum spanning tree of G. Graph G and tree T are given. Assume the cost of one of the edges e = (u,v) in G is decreased from w_e to w'_e. Design an algorithm to find a minimum spanning tree in the modified graph. The running time of your algorithm should be O(n). Describe all details and write pseudo-code for your algorithm.

4. Euclidean MST Implementation

Implement an algorithm in the language of your choice (C, C++ or Python) that determines the weight of a MST in a graph $G=(V,E)$ where the vertices are points in the plane and the weight of the edge between each pair of points is the Euclidean distance between those points. To avoid floating point precision problems in computing the square-root, we will always round the distance to the nearest integer. For all $u, v \in V$, $u = (x_1,y_1)$, $v = (x_2,y_2)$ and

$$w(u,v) = d(u, v) = nearestint\ \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

For example, if u = (0, 0), v = (1, 3),

$$\begin{aligned} d(u,v) &= \text{nearestint}\left(\sqrt{(0-1)^2+(0-3)^2}\right) \\ &= \text{nearestint}\left(\sqrt{(-1)^2+(-3)^2}\right) \\ &= \text{nearestint}\left(\sqrt{1+9}\right) \\ &= \text{nearestint}\left(\sqrt{10}\right) \\ &= \text{nearestint}(3.1622\ldots) \\ &= 3 \end{aligned}$$

Assume that the graph $G=(V,E)$ is complete so that there is an edge between ever two vertices Your program should read information about the graph from a file called graph.txt. The first line in graph.txt

is the number of tests cases ($k \leq 10$) followed by the number of vertices in the test case ($n \leq 100$) and the coordinates of the points (vertices) in that test case.
An example of graph.txt is shown below:

```
2
3
0 1
2 1
2 5
5
0 0
0 4
4 4
4 8
1 1
```

The output should be displayed to the terminal and give the test case number followed by the weight of the MST.
Test case 1: MST weight 6
Test case 2: MST weight 12

Submit to Canvas:

- a verbal describe of your algorithm and data structures
- the pseudo-code
- theoretical running time

Submit to TEACH

- README file with instructions on how to compile and run your code
- All code files

CS 520: A ***bottleneck spanning tree*** *T* of an undirected graph *G* is a spanning tree of *G* whose largest edge weight is minimum over all spanning trees of *G*. We say that the value of the bottleneck spanning tree is the weight of the maximum-weight edge in *T*.

a**)** Argue that a minimum spanning tree is a bottleneck spanning tree.
Part a) shows that finding a bottleneck spanning tree is no harder than finding a minimum spanning tree. In the remaining parts, we will show how to find a bottleneck spanning tree in linear time.

b) Give a linear-time algorithm that given a graph *G* and an integer *b*, determines whether the value of the bottleneck spanning tree is at most *b*.

c) Use your algorithm for part b) as a function in a linear-time algorithm for the bottleneck-spanning-tree problem. (*Hint:* You may want to use a function that contracts sets of edges, as in the MST-REDUCE procedure described in CLRS Problem 23-2.)

Written solutions must be typed using a word processor or text editor and submitted to Canvas. Submit a copy of all your code files and a README file that explains how to compile and run your code on flip in a ZIP file to TEACH.