Assignment 3

COMP 3804, Fall 2021

Upload in Brighspace by 11:59PM on November 12, 2021

1 Guidelines

General guidelines are as follows:

1. Since we are only accepting assignments via Brightspace, no late submissions will be entertained after the cut-off time & date.
2. Please write clearly and answer questions precisely. It is your responsibility to ensure that what is uploaded is clearly readable. If we can't read, we can't mark!
3. Please cite all the references (including web-sites, names of friends, etc.) which you used/consulted as the source of information for each of the questions.
4. All questions/problems carry equal marks.
5. When a question asks you to design an algorithm - it **requires** you to
   (a) Clearly spell out the **steps** of your algorithm in pseudo code.
   (b) **Prove** that your algorithm is correct
   (c) **Analyze** the running time.
6. You can assume that a graph $G = (V, E)$ uses adjacency list representation.
7. n is a positive integer, and it typically represents the size of the input to a problem.

2 Problems

1. Let $G = (V, E)$ be a directed acyclic graph. We say that G is *semi-connected* if for every pair of distinct vertices $u, v \in V$, we have that there is a directed path from u to v or there is a directed path from v to u in G. Given G in the adjacency list representation, design an algorithm running in $O(|V|+|E|)$ time to determine whether G is semi-connected. (Hint: First, construct examples of directed-acyclic graphs on four vertices that are semi-connected and that aren't. Determine what property (with respect to their linear order) distinguishes them.)

2. Let $G = (V, E)$ be a simple connected graph, where each edge weight is 5. Devise an algorithm, running in $O(|V| + |E|)$ time, for computing shortest path distances from a specific vertex $s \in V$ to all other vertices of G.

3. Prove that the distance values extracted from the priority queue (heap) over the entire execution of Dijkstra's single-source shortest path algorithm, in a directed connected graph with positive edge weights, is a NON-Decreasing sequence. Where is this fact used in the correctness of the algorithm?

4. In Internet routing, there are delays on edges and significant delays on switches (i.e. vertices). Suppose that in addition to having non-negative edge weights in the graph $G = (V, E)$, we have a positive cost associated with each vertex in G. Let the cost associated to a vertex v be $c(v)$. The cost of a path is defined as the sum total of the weights of the edges in the path + the sum total of the costs of all the vertices in the path. Present an efficient algorithm to compute the shortest path costs from vertex s to all the vertices in the graph, with this modified definition of the path's cost. Present a Pseudocode - Analyze the Complexity and - Justify why your algorithm is correct.

5. In the summer vacation, you decided to travel to various communities in Northern Canada by your favourite ATV (All-Terrain Vehicle). Each of the communities you want to visit is represented as a vertex in your travel graph (a total of $|V|$ communities). Moreover, you are provided with distances between all pairs of communities. Think of your input graph as a complete graph (i.e. every pair of vertices are joined by an edge), and the weight of an edge, say $e = (uv)$ is the distance between the community u and v. Since this is in the far North, and the routes between communities are not used that often, the gas stations are only located in communities (there are no gas stations outside a community). Furthermore, we can assume that each community has at least one gas station. Once you fill-up the tank of your ATV, it has an upper limit, say of Δ kilometres, which it can travel, and to travel any further, it needs to fill up (which means at that point it needs to be in a community!). Answer the following questions:

   (a) First design a method, running in $O(|V| + |E|)$ time, which can answer whether there is some path which your ATV can take so that you can travel between two particular communities, say s and t. If the distance between s and t is at most

Δ, you can travel directly without refuelling. Otherwise, you can travel between s and t, provided there are communities where we can refuel and proceed.

(b) Design an algorithm running in $O(|E| \log |V|)$ time to determine the smallest value of Δ, which will enable you to travel from s to t. (Please present Pseudocode, correctness, analysis) and use the algorithms discussed in the class/book as black boxes).

6. Let $G = (V, E)$ be an edge-weighted simple connected undirected graph and assume that edge weights are distinct. Show that for any cycle C in G, the edge e with maximum weight on the cycle is not in any minimum spanning tree of G.

7. Let $G = (V, E)$ be an edge-weighted simple undirected connected graph, and assume that all edge weights are distinct and positive. A *nice* tree T of G is a spanning tree of G whose largest edge weight is minimum over all spanning trees of G. Construct an example of a weighted graph G and a spanning tree T of G such that T is a nice tree but not a minimum spanning tree of G. Show that any minimum spanning tree of G is a nice tree.

8. Consider a connected graph $G = (V, E)$ where each edge has a non-zero positive weight. Furthermore, assume that all edge weights are distinct. Show that for each vertex $v \in V$, the edge incident to v with minimum weight belongs to a Minimum Spanning Tree (MST). Can you use this to devise an algorithm for MST? Note that the above step identifies at least $|V|/2$ edges in MST. Now we can collapse these edges by *identifying* the vertices and then recursively applying the same technique - the graph in the next step has at most half of the vertices that you started with - and so on. The recursion terminates when we are left with a single vertex. At that point, we would have collected $|V| - 1$ edges that are in an MST.

   Note that for an edge $e = uv$ in the graph $G = (V, E)$, *identifying* vertex u with v or *collapsing* e is the following operation: Replace the vertices u and v by a new vertex, say u'. Remove the edge between u and v. If there was an edge from u (respectively, v) to any vertex w ($w \neq u$ and $w \neq v$), then we add an edge (with the same weight as of edge uw (respectively, vw)), between the vertices u' and w. This transforms graph G to a new graph $G' = (V', E')$, where $|V'| < |V|$ and $|E'| < |E|$. Note that G' may be a multigraph (i.e., between a pair of vertices, there may be more than one edge). For example, if uv, uw, and vw are edges in G, then G' will have two edges between u' and w when we identify u with v. We can transform G' to a simple graph by keeping the edge with the lower weight among uw and vw as the representative for $u'w$ for the computation of MST.

9. Let $G = (V, E)$ be an edge-weighted undirected connected graph. Assume that each edge weight is positive and distinct. Let T be a minimum spanning tree of G. Suppose we modify the weight of each edge of G to be the square root of its original weight (e.g. if the weight of an edge e is 16, its modified edge weight will be $4 = \sqrt{16}$). Is T a minimum spanning tree of the modified edge-weighted graph? Justify your answer.

10. Let $G = (V, E)$ be an edge-weighted directed connected graph. Assume that each edge weight is positive and distinct. Let $s, t \in V$ be two specific vertices in G and let $\pi(s, t)$ be a shortest path in G between s and t. Suppose we modify the weight of each edge of G to be the square root of its original weight. Will $\pi(s, t)$ continue to be a shortest path between s and t in the modified graph? Justify your answer.