Name: Luke Joyce

ID: 107355873

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**

Problem Set 4b (45 points) **Fall 2019, CU-Boulder**

Instructions for submitting your solution:

- The solutions **should be typed** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Gradescope if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.
- Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.
- For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.
- You may work with other students. However, **all solutions must be written independently and in your own words.** Referencing solutions of any sort is strictly prohibited. You must explicitly cite any sources, as well as any collaborators.

Name: Luke Joyce
ID: 107355873

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 4b (45 points) **Fall 2019, CU-Boulder**

1. (10 pts) For a directed graph with positive weights, we define the max-weight of a path from s to d as the maximum of the edge weights along the path. For example, if the path from A to D has edges and weights of $e_{AB} = 5$, $e_{BC} = 4$, and $e_{CD} = 1$, the length of the path is defined as $e_{AB} + e_{BC} + e_{CD}$, and the max-weight is 5.

   (a) (5 pts) Give an algorithm to compute the smallest max-weight paths from a source vertex s to all other vertices. In this problem, you are changing the definition of length of the path from A to D to $max(e_{AB}, e_{BC}, e_{CD})$ (Hint: Your algorithm should be a modification of Dijkstra's algorithm presented in Lecture.)

   Solution. `Dijktras(G, s):  G is input graph, S is source vertex`

```
Initialize maxEdge, prev //dist changed to maxEdge
maxEdge(s) = 0
maxEdge(v) = infinity

Q = min priority queue of vertex weights in G
Q.add(v)
while Q = !empty:
    u = Q.pop()
    for each v in u.adjacent:
        if maxEdge(u) < edge(u, v):  // Compare max edge of path to
            d = maxEdge(u,v)         // vertex u to current edge to v
        if d > maxEdge(v):   // Compare d to current max edge in path
            maxEdge(v) = d   // to v
            prev(v) = u
```

(b) (5 pts) Prove the correctness of your algorithm.

Solution. Base Case: —S— = 1, therefore there is only one vertex, and the distance to S is 0. Max edge weight along that path is 0.

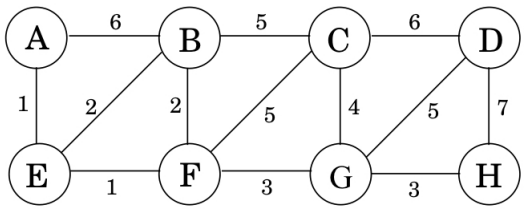
Inductive Hypothesis: Assume —S— = m hold for all m $\geq$ 1, where S is the solved vertices in G.

Inductive Step: Suppose we add vertex v to graph G. Then —S— now equals m + 1. Let us make u the previous vertex of v. We compare $maxEdge(u)$ with $e_(u, v)$. if $e_(u, v)$ is greater than $maxEdge(u)$, d is set to $e_(u, v)$ as opposed to $maxEdge(u)$. Next, the algorithm checks to see if d is less than $maxEdge(v)$. In other words it checks to see if our value for d is less than the current maximum edge set for v. If v has not been visited yet, then its maxEdge is ∞, in which case it would be reset as the value that d holds. If $maxEdge(v)$ is something else besides infinity, then if $maxEdge(u)$ is less than that value, $maxEdge(v)$ is set to $maxEdge(u)$, ensuring that the lowest maximum on a path to v is correct.

Name: Luke Joyce
ID: 107355873

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 4b (45 points) **Fall 2019, CU-Boulder**

2. (11 pts) Based on the following graph :



(a) (4 pts) In what order would Prim's algorithm add edges to the MST if we start at vertex A?

Solution. `The order each edge is added:`

```
A->E  (1)
E->F  (1)
E->B  (2)
F->G  (3)
G->H  (3)
G->C  (4)
G->D  (5)

Total weight is 19.
```

(b) (7 pts) In what order Kruskal's would add the edges to the MST? For each edge added by Kruskal's sequentially, give a cut that justifies it's addition.

Solution.
```
Priority Queue of Edges:
v:  length:
A-E (1) Add to graph
E-F (1) Add to graph
E-B (2) Add to graph
B-F (2) Do not add to graph (forms loop)
F-G (3) Add to graph
G-H (3) Add to graph
C-G (4) Add to graph
C-F (5) Do not add to graph (forms loop)
D-G (5) Add to graph
B-C (5) Do not add to graph (forms loop)
A-B (6) Do not add to graph (forms loop)
C-D (6) Do not add to graph (forms loop)
D-H (7) Do not add to graph (forms loop)
```

3. (10 pts) Let T be a MST of a given graph G. Will T still be the MST if we reduce the weight of exactly one of the edges in G by a constant **c**? Prove your answer.

   Solution. T will not necessarily still be the MST of a given graph after you reduce the weight of exactly one edge in G by a constant c.

   When finding the MST of graph G, suppose there was a comparison between two edges to a vertex who's difference in weights of the two edges was 2 ($e_1 - e_2 = 2$). Therefore e_2 would be chosen because it is the smaller edge. Lets say the constant c we choose is 5. If we subtract 5 from e_1, the the difference between the two, $e_1 - e_2 = -3$, and therefore e_1 would have been chosen for the graph.

   $\therefore$ the MST would change in that case of one edge being lower by a constant c.

4. (14 pts) One of the uses of MSTs is finding a set of edges that span a network for minimum cost. Network problems could also have another type of objective: designing a spanning tree for which the most expensive edge is minimized. Specifically, let $G = (V, E)$ be a connected graph with n vertices, m edges, and positive edge costs that you may assume are all distinct. Let $T = (V, E)$ be a spanning tree of G; we define the **limiting edge** of T to be the edge of T with the greatest cost. A spanning tree T of G is a minimum-limiting spanning tree if there is no spanning tree T of G with a cheaper limiting edge.

   (a) (7 pts) Is every minimum-limiting tree of G an MST of G? Prove or give a counterexample.

   Solution. Minimum-limiting trees of G are not necessarily minimum-spanning trees of G. Consider an graph that has two minimum-spanning trees, A and B, where the total weight of the MST is 15. Let us suppose that both of these minimum-spanning trees are also minimum-limiting trees. Let us also suppose A contains edges x and y, that are weighted 2 and 3 respectively, and 3 is the limiting edge of spanning tree A. If we were to increase the weight of x in spanning tree A to 3, A would still be minimum-limiting tree because edge $x = 3$ is less than or equal to edge $y = 3$. But now the weight of spanning tree A is now 16, since we increased the weight of one edge. Spanning tree A is now still a minimum-limiting tree, but is no longer a minimum-spanning tree.
   $\therefore$ Not all minimum-limiting trees are minimum-spanning trees.

(b) (7 pts) Prove that every MST of G is a minimum-limiting tree of G. [**Hint:** Let T be an MST of G, and let T' be a minimum-limiting tree of G. If T is not a minimum-limiting tree, can we replace the heaviest edge of T? Think about how to use T' here.]

Solution. Considering the fact that the edges of G are ordered in a priority Queue, each lowest edge is added first to the tree unless it forms a loop. Let us say that T', a minimum limiting tree of G, has maximum edge m. We proved above that all minimum limiting trees of G are not necessarily also minimum spanning trees of G. That being said, the total weight of any given minimum limiting tree will always be greater than or equal to the total weight of a minimum spanning tree. A minimum limiting tree is a spanning tree of G with the lowest maximum weight, which means that it too checks all the lowest weights first and only adds them if they do not form a loop in the current spanning tree in G, just as the minimum spanning tree algorithm does. Therefore, there is no possibility that the minimum spanning tree would consist of an edge greater than the maximum edge in a minimum-limiting tree of the same graph, in which case replacing any edge of the MST with the max edge of T', the minimum limiting tree. $\therefore$ all MST of G are a MLT of G.

Name: Luke Joyce
ID: 107355873

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 4b (45 points) **Fall 2019, CU-Boulder**

Ungraded questions - These questions are for your practice. We won't grade them or provide a typed solution but we are open to discuss these in our OHs and you should take feed backs on your approach during the OHs. These questions are part of the syllabus.

1. Suppose you are given the minimum spanning tree T of a given graph G (with n vertices and m edges) and a new edge $e = (u, v)$ of weight w that will be added to G. Give an efficient algorithm to find the MST of the graph $G \cup e$, and prove its correctness. Your algorithm should run in $O(n)$ time.

   Solution.

2. Based on the following graph :

   PS4/mst_graph_q2.jpg

   (a) Run Kruskal's and find the MST. You can break the ties however you want. Draw the MST that you found and also find it's total weight.
   Solution.

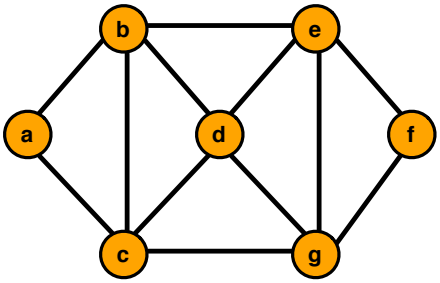
(b) Run Prim's starting from vertex A and find the MST. You can break the ties however you want. Draw the MST that you found and also find it's total weight. Is the total weight same as what you get from the above?

Solution.

3. Consider the following unweighted graph, and assign the edge weights (using positive integer weights only), such that the following conditions are true regarding minimum spanning trees (MST) and single-source shortest path (SSSP) trees:

   - The MST is distinct from any of the seven SSSP trees.
   - The order in which Prim's algorithm adds the safe edges is different from the order in which Kruskal's algorithm adds them.

   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.



Solution.