CMPS 102 — Fall 2018 – Homework 3

Alyssa Melton

I have read and agree to the collaboration policy.

Collaborators: none

Solution to Problem 2

Because we have limited RAM, we can only read each edge off the disk once, meaning we must decide whether to save it or discard it on the spot.

$|E| = m$
$|V| = n$
edge e has two vertices associated with it: $e.1$ and $e.2$

Consider the following algorithm and helper functions:

MSTincludes(u)
    returns true if the vertex u exists in the MST set
    returns false if the vertex u is not yet in the MST set

getMaxEdge(u,v)
    returns the maximum edge in the path from vector u to v in the MST set.
    returns null if no path from u to v exists.

getWeight(e)
    returns the weight of an edge e

for each edge *e* from 1 to m:
    if ($MSTincludes(e.1)$ && $MSTincludes(e.2)$) {
        variable maxEdge = getMaxEdge(e.1, e.2)
        if maxEdge == null: add e to MST.
        else if getWeight(e) < getWeight(maxEdge)
            delete maxEdge from MST
            add e to MST
    } else { add edge e to MST }

Claim 1. *All vertices are in the MST.*

Proof. The algorithm says that if one of the vertices on either end of the edge we are looking at is not already in the tree, add the edge. Adding the edge necessarily adds both vertices on either end of that edge as well. □

Claim 2. *All vertices are connected.*

Proof. G itself is a connected graph, and our algorithm goes through every edge, checking whether or not a path to the two vertices on either end of e exists. If a path to the two vertices in questions, on either end of

edge e does not exist, we add edge e.

If the resulting tree was not connected, G itself would not have been connected, since every connection in G is preserved through, "if maxEdge == null: add e to MST" □

Claim 3. *There are no cycles.*

Proof. A cycle can only be created when a path from vertex u to v already exists, and we add another edge that connects those two vertices. However, the algorithm addresses this, but removing the maximum weight edge in the path from u to v if it exists. By removing the max weight edge, we are breaking the tree into two, but by adding the edge from u to v we are connecting it again. □

Claim 4. *The given tree has the minimum possible total edge weight.*

Proof. Suppose we have a path from u to v, and an edge e with weight e_w. Suppose the path from u to v was the sum of weights of edges, namely, $w_{u,v} = w_1 + w_2 + w_3... + w_k$. Say w_j is the heaviest weight in the path from u to v. If $w_j > e_w$, we can add edge e and remove e_k. This will necesarily decrease $w_{u,v}$. If say, we were to remove more than the heaviest edge, then we would disconnect the Tree, since there are no cycles as proved above. □

Claim 5. *The result will the Minimum Spanning Tree for the Graph G.*

Proof. By definition, a minimum spanning tree (MST) is a subset of the edges of a connected, edge-weighted (un)directed graph that connects ***all*** the vertices together, without any cycles and with the minimum possible total edge weight.

By the above claims, the result contains all the vertices, has no cycles, and has the minimum possible total edge weight. Thus, the resulting Tree is an MST. □

Claim 6. *The above algorithm's space complexity is $O(n)$.*

Proof. We have proved that we are storing a tree, versus a graph. It has been proved that a connected tree of n nodes has exactly $n-1$ edges. Each iteration, we are looking at one edge at a time, making us holding data for the $n-1$ in the tree, and 1 edge while deciding whether to add it to the tree or toss it. Thus, $n-1+1 =$ total: $O(n)$ □