Steven Bui, Ethan Ng, Alex Young
CS 325

Group Assignment 4

Description of Algorithm

Generating priority queue and dictionary

The algorithm starts by initializing a priority queue to store all the possible edges and corresponding vertices in (as tuples). A dictionary is also implemented to store the (V - 1) best path for the minimum spanning tree.

Next, the algorithm reads in the input file. This will be done by looping through the input file and reading in the edge weights in the upper right "triangle". The weight and the connected vertices are put into a priority queue using the weight as the priority value. Next, the algorithm will use this priority queue and aspects of Kruskal's Algorithm to return the minimum length of the spanning tree as well as calculate the second and third minimum spanning tree.

Check for if Edge is Safe or Unsafe

The function that our algorithm uses to find the MST implements the main ideas from Kruskal's algorithm, mainly being the use of a set to mark edges as "safe" or unsafe. Instead of using a set directly, our algorithm uses a list of tuples to hold each vertex (essentially a disjoint set). When checking if an edge is safe or not, the algorithm will refer to this "path" of vertices to see if they have been marked as part of a path or not. In order to do this, the algorithm loops through the list of paths to check if both the vertices of the current edge (which is the lowest weighted edge in the graph) are in the same path. If the vertices connected to the current edge are in the same path, that means that this edge is "unsafe" because both vertices have been traveled through already. If the vertices are not in the same path, then the current edge is also an optimal and safe edge for the minimum spanning tree.

Edge is safe - MST

As we loop through the (V - 1) total edges, if an edge is safe, the MST dictionary will update to include the current edge. After that, the list of paths will update to show that this edge in the MST is marked. Each tuple holding the corresponding vertex to the safe edge will merge together. For example, [(1, 7, 2), (3, 0, 5), (4, 8), (9)] could have an edge of 7, 3 entered and the union would be a path of [(1, 7, 2, 3, 0, 5), (4, 8), (9)]. By the end of function, after looping through every optimal edge, the path should just be one long tuple with every vertex inside of it and the MST dictionary should have an edge entered for every (V - 1). Finally, we need to update a dictionary that holds all the previous lists of paths inside of it, to access for when we are checking the 2nd and 3rd MSTs.

In addition to updating the edge dictionary and the list of paths, every time the algorithm finds a safe edge, it will increment the count of the minimum spanning tree. The final safe edge will bring the count up to the final first minimum spanning tree.

Edge is unsafe - 2nd or 3rd MST

If the current edge that has been popped off the priority queue is deemed as unsafe, that means it doesn't fit into the minimum spanning tree, however it does mean that it could fit into the second or third minimum spanning tree. To start here, we will initialize delta1 and delta2. Delta1 is the difference between the minimum spanning tree and the second minimum spanning tree. Delta2 is the difference between the MST and the 3rd MST .

To check if the edge could fit into the next MSTs, we compare the edge to all the currently found edges in the MST dictionary. If the difference between the current edge's weight and the weight of the optimal edge is less than delta2 (or delta2 doesn't have a value yet) then it can possibly be a replacement edge that connects two known paths to create the second or third MST.

First we need to check if the vertices of the current edge are in different paths in the list of paths for the current MST edge that we are comparing to. This is done by referencing the dictionary that holds all the previous lists of paths and is updated for every new path.

If the vertices are in different paths and the difference between the current edge weight and MST edge weight is still relevant, update the delta1 or delta2 as necessary to take into account the new possible second or third minimum spanning tree.

Optimization

We know we can be done looping through all the edges in the priority queue once the difference in weight of these edges and the current delta is large enough that the delta cannot be decreased any further.

Output

After finding the length of the MST and delta1 and delta2, by adding together the length to delta1 and delta2 the algorithm will find the second and third MST. If the value of V of the original input is 1 or 2, then there will be no second and third MST based on how the graph will be formed. The weight of these 3 MSTs will be sent to an output file.

Runtime Analysis

The algorithm has a loop which iterates (V) amount of times in order to consider each edge in the final MST. Inside this loop, we will be looping in the range of the current edge (V). For each edge that this loop looks at, the algorithm loops through every previous list of paths generated, up to (V) - 1 to compare the current edge vertices to the previous vertices in the MST. Overall the deepest number of iterations done by the algorithm is (V^3), which is shown in more detail below.

O(E) (input files)
O(V(V + V) + (V(V + V))) (mst creation function, (V+V) edge safe, (V(V+V)) edge not safe)
O(1) (difference calculation function)
T(n) = O(E) + O(V^3) = O(V^3)
The number of edges in a complete graph is E = V(V - 1) = V^2 - V
Therefore O(E) = O(V^2 - V) = O(V^2)
Therefore T(n) = O(V^3) = O(VV^2) = O(VE)
$T(n) = O(E) + O(V((V + V) + (V(V + V)))) = O(V(V + V^2)) = O(VV^2) = O(VE)$

Correctness of Algorithm

<u>Proving our algorithm finds the minimum spanning tree correctly</u>

We know that our minimum spanning tree produced by our algorithm is the optimal MST because for every iteration of the algorithm, the produced graph (or path) of vertices has the least possible weight.

Base Case:

- Initially, each edge is added to the priority queue and is sorted by weight. Our algorithm starts by selecting the lowest weighted edge and adding it to the graph
- For this graph with two vertices, we know that the minimum spanning tree is the edge from the top of the priority queue because it has the minimum weight.
- We also know that any other edge that has more weight would not be the MST

Induction Hypothesis

- We assume that not only is our algorithm forming an MST for the first edge is correct, but also that every additional edge that is added to the graph is the minimum weight possible to safely add

Induction Step

- If the edge that we are adding is the minimum weight possible to add to the graph then adding it to the graph will create an optimal minimum spanning tree of the length of the graph + 1 (with a weight of the weight of the graph + weight of edge)
- Because we keep track of the path that every vertex adds we know that this next edge is safe to add to the MST and only combines vertices that haven't been added to the MST yet
- We know that each time we add an edge to the MST, it is correctly adding the minimum weight and covering more vertices, and thus once we have gone through (V - 1) edges, we will reach every vertice with the minimal possible edges, creating a minimum spanning tree.

<u>Proving that our algorithm finds the next (2nd and 3rd) minimum spanning tree correctly</u>

The algorithm finds the second and third MST by comparing every unsafe edge to all the found optimal MST edges. It says that if the unsafe edge vertices are in the same corresponding paths that the MST edge combines together, then that edge can be removed and replaced by the unsafe edge to give a new (larger) MST. These new MSTs are calculated, and the shortest two are used for the 2nd and 3rd MST.
We can simplify this algorithm by saying that it is operating on the logic that if we remove one edge from the MST, then there will be a non-optimal edge that can replace it by connecting the two paths that it was connecting previously, and that this new complete path will create the next minimum MST.

Thus, in order to prove that our algorithm will give the correct 2nd and 3rd MST, we need to prove that the next minimum MST can be found by replacing one optimal edge (that connects two paths), with the next minimum weighted edge that connects the same two paths.

Assume:

- We have the correct minimum spanning tree for a complete graph and have found the correct path and edges to create the minimum spanning tree

Proof:

- The number of edges in the MST is (V - 1) because that is the minimum number of edges that will reach every vertex in the tree.
- In order to create the next MST, we cannot have all the current edges in the MST because they are the optimal minimum edges and will give the optimal MST
- Thus, the next MST must have one of the edges removed from it and a next minimum edge needs to replace this edge.

- We know that we only need to replace one edge because if we replace 2 optimal edges, then we will be adding two non-optimal edges and the unnecessary weight will be added to the spanning tree.
- Because we know this, we also know that by removing one of these optimal edges from the possible optimal edges that the MST algorithm searches through (in the priority queue), the minimum spanning tree produced will be the most optimal minimum spanning tree without one of the optimal edges.
- If we count the total edges in the MST and calculate the minimum spanning tree in every iteration with the corresponding edge taken out of the options, then all of the minimum spanning trees generated will be not optimal (because they are missing an optimal edge), but as close to optimal as possible because they are only missing one optimal edge and every other edge is optimal.

- In order to show that our algorithm finds the minimum spanning tree in each iteration with the corresponding edge taken out, we can use contradiction.
- Our algorithm replaces one optimal edge (that connects two paths), with the minimum weighted edge that connects the same two paths. If we assume that this is wrong then there are several other ways to find the minimum spanning tree instead
    - First Case: replace the optimal edge with a minimum weighted edge that connects different paths than the optimal edge had connected.
      (Contradiction) If we replace an edge with an edge that doesn't connect to the same paths as the original edge, then there will be vertices missing in the final spanning tree, which won't actually be a spanning tree
    - Second Case: replace more than one optimal edge with minimum weighted edges
      (Contradiction) As proven above, if more than one optimal edge is replaced, it will no longer form the next minimum spanning tree
    - Third Case: do not replace any edges
      (Contradiction) If no edges are replaced, then we are left with the original MST
    - Fourth Case: replace the optimal edge with a non-minimal weighted edge
      (Contradiction) If we are replacing an optimal edge with another edge (e1) and created a new MST with weight (t - e + e1(w)), then if we had replaced the optimal edge with an edge (e2) that has less weight than e1, (t - e + e2(w)) with e2(w) < e1(w), then the minimal-weighted edge will create a smaller spanning tree and the non-minimal weighted edge will not create the next MST
- By showing that if we assumed our algorithm was wrong and contradicted all the other possibilities, it follows that our algorithm is correct and that we are finding the correct next MST