WA9 - Assignment Group 16

Sravani Kirla - skirla
Matthew Heil - heilx069
Terrence Hanrahan - hanra028

1. Assume G=(V,E) is the graph given and the minimum spanning tree computed from G have T edges. Here the number of vertices is V and edges E. Consider G' is the new graph with one additional vertex. For G' consider the number of vertices is V' and edges E'. Computed a minimum spanning tree for G' will have edges T'.
    - When we compute minimum spanning tree from graph G number of edges that are not included in MST are E-T (if we consider the number of edges in minimum spanning tree are T and total edges in G are E).
    - If run the same Prim's algorithm on G and G' then it will produce a minimum spanning tree of T and T' respectively and the remaining edges will be E-T and E'-T'.
    - For the first iteration set T(MST) will be empty, and an empty set trivially satisfies the condition.
    - When we add each vertex at ith iteration we always take the minimum key value which is nothing but the lowest weight from the queue as all other vertices unprocessed will have infinity at this point.
    - So each iteration next minimum edge from the queue will be added to set T leaving the highest possible edge in the Queue. Remaining edges at termination will be the maximum weight edges call it E-T for G.
    - If say some of the edges from E-T is added to the MST with edges T' for G'. This statement will contradict because if any of the edge from E-T can be added for T' then it could have added it to the T when computing MST for G which means non of the E-T edges have minimum weights to form a spanning tree.
    - So we can say T is a subset of T' and E-T edges are always maximum weighted edges which will not be included in the MST.

<u>RUNTIME</u>:

- From above conclusion we can use already computed MST T of Graph G to compute MST T' for Graph G'. So to compute T' we can use a graph which is T(MST) plus all the edges from new added vertex.
- As we know by definition T(MST) will have maximum number of edges = |V|-1, and maximum number of edges possible from newly added vertex is |V|. New Graph to compute T'(MST) will have maximum edges of |V|-1+|V| => i.e., 2|V|-1
- Now if we use Prim's algorithm to compute MST T' on new Graph with edges=2|V|-1 and vertices = V' = V+1, replacing the edges in the Prim's runtime O(E lg V), we will get O(2V-1 lg V') => O(V lg V). Hence we can generate MST T' of G' in O(V lg V).

2.

<u>**MST_A**</u>:

As we order our edges in non-increasing order, all the edges will be sorted from maximum weight first to minimum weight at the end of the set. When we loop through all of these edges we remove the edge which if we can still get a connected graph meaning whatever we take out all of those edges will have higher weights than that are included in the set T. By definition, a spanning tree can not be disconnected and we are not removing any edge which can form a disconnected graph in this algorithm. And if there is a cycle we will remove that edge in line 5 from set T if we can still get a connected graph without that edge. Hence this **does return a MST**.

If we use DFS only one time for finding the connectivity on a given vertex to all other vertices we will know whether this graph is connected or not. Considering the DFS runtime O(V+E). If we use Heapsort for sorting the edges then it will have O(E lg E) worst case runtime. And we are iterating all edges at line 3, so the overall runtime can be written as O(E (V+E)) => O(EV + E^2)=> O(E^2) this will be the worst case runtime of this algorithm.

<u>**MST_B**</u>:

If we take a graph with 3 edges and 3 vertices(A,B,C) which forms a cycle. Let's say edges are with weights (A,B) = 2, (B,C) = 3 and (C,A) = 6. This algorithm may produce different results each time because of the arbitrary edge selection.

Scenario 1(***Returns a MST***):

If at line 2 of the algorithm it chooses the edges in this order with weights 2, 3, 6 then it will form an MST as first two edges are with minimum weights already. It will add (A,B) edge to the empty set T for the first iteration because the set is empty before adding this edge and it doesn't form a cycle. For 2nd iteration it will pick (B,C) edge and it will add it to the set T at this point by adding B will not result a cycle either. But in our last iteration by adding an edge (C,A) forms a cycle and it removes that edge from the set and therefore we will end up getting a MST.

Scenario 2 (***Does not return a MST***):

If at line 2 of the algorithm it chooses the edges in this order with weights 6, 3, 2 then it will NOT return a MST. Because it will add (C,A) and (B,C) in first and second iteration but when you add (A,B) it will form a cycle so it will remove it from the set T. But in this example (A,B) + (B,C) with weights 2+3 gives a MST unlike (C,A) + (B,C) which will be 6+3.

<u>**MST_C**</u>:

This will return a MST because at line 4 it checks whether it has a cycle and if it has a cycle it removes the maximum weighted edge of the cycle from the set T. And when we remove an maximum weighted edge from a cycle it will result in spanning tree if it is connected.

We can use the disjoint sets data structure to find a cycle in a graph for this algorithm at line 4. UNION() function will not add the same vertex if the FIND-SET() returns the same root vertex to avoid creating a cycle. By using this when FIND-SET() of two edges is equal we can say those edges will form a cycle. FIND-SET() runtime is O(E) times and UNION() runtime is O(V) times. So to find a cycle or avoid making a cycle it can be done in O(EV). And to find which edge is maximum we can run DFS on the each of the edges which forms a cycle and update each vertex instance variable ‘w’ with values as weights of the edges and check if the maximum weight of any edge is less than that of the newly added edge then remove this new edge if not remove the maximum weighted edge and add this new edge. And at most at this point there can be only one cycle can exist so it will be of O(V) times. So total run time would be O(EV) for this implementation.

3. Yes, shortest path is possible.
e to a: e -> c -> a, final weight 4
e to b: either no valid shortest path or e -> c -> b, final weight -3 (zero cycle w/ b and c)
e to c: e -> c, final weight -2 (zero cycle w/ b and c)
e to d: not reachable
e to f: e -> f, final weight 4
e to g: e -> c -> a -> g, final weight 6