CISC 621 - Homework Problem R

Model Solution

a) Kruskal's algorithm produces a MSF for G even when G is unconnected because it adds safe edges to the set A in such a way that the edges in the set do not need to form a single tree.
The algorithm in the first step considers a forest of |V| trees. Then, at each step, it finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees in the forest, an edge (u,v) of least weight. Thus, the algorithm ends up producing a set of connected components, with each connected component a MST.

b) MSF-Kruskal (G,w)

```
1     A <- 0
2     num_vertices <- 0   \\ variable to store number of vertices in
      the graph
3     num_msf_edges<- 0  \\ variable to store the number of MSF
      edges for G
4     num_components <- 0 \\ variable to store the number of
      components in G
5     for each vertext v ε V[G]
6         do Make-set (v)
7             num_vertices <- num_vertices + 1  \\** new  addition**
\\ After the loop, the variable num_vertices contains the number of
vertices in the graph
8     sort the edges of E by nondecreasing weight w
9     for each edge (u,v) ε E , in order by nondecreasing weight
10       do if FIND-SET(u) ≠ FIND-SET (v)
11           then A<- A U {(u,v)}
12                 UNION (u,v)
13                 num_msf_edges = num_msf_edges + 1
                                                  \\** new addition **//
      \\ After the 'for' loop, the variable num_msf_edges contains
      the number of edges in MSF
```

```
14  num_components<-num_vertices - num_msf_edges
                                              \\ ** new addition **\\
15  return  A, num_components \\  finally return the number of
    components stored in the variable num_components
```

c) Prim’s algorithm first processes the nodes within the component pointed to by root r. The algorithm then adds at each step a safe edge that is always a least weight edge connecting the tree to a vertex not in the tree but present in the same component. In this way, the algorithm constructs MST for that component.
Once the construction is complete, all the nodes in the priority queue Q have weight ∞. Assuming that the Extract-Min() function arbitrarily selects one of those nodes, the algorithm then starts executing in a similar fashion as above. The algorithm decreases the weight of adjacent nodes and sets the parent pointers of these nodes.
However, the algorithm does not keep track of the root nodes of different MST’s formed. It only contains pointer to the root node of the first MST formed. Also, the parent of root nodes other than the root node of first MST is not set to NIL & their key value still equals to ∞.
Thus, Prim’s algorithm does not work when G is unconnected

d) MSF_PRIM(G,w,r)
// sibling list is used to link root nodes of different MSTs

```
1   Q<- V[G]
2   num_components <- 0 //stores the number of components in G
3   for each u $\varepsilon$ Q
4       do key[u]<- ∞
5   key[r]<- 0
6   Π [r]<- NIL
7   sibling[r] <- NIL //set root’s sibling to NIL
8   num_components <- 1 // increment the num_component
    variable to count the first component
9   while  $Q \neq 0$
10      do u <- Extract-Min (Q)
            // new addition
11          if  key[u] = ∞
12              num_components <- num_components + 1 // since a
        //new node is being traversed when the key value is ∞
```

```
            key[u] <-0  // as u is the new root node
            Π[u] <- NIL // since u is the new root node
            sibling[r] <- u // make sibling of r point to u
            sibling[u]<-NIL //make sibling of u point to NIl
            r  <- u // assign u to r
        for each v ε Adj[u]
             do if v ε Q and w(u,v) < key[v]
                  then  Π [v]<- u
                        key[v] <- w(u,v)
return  num_components // finally return the number of
components in G
```