Pseudocode

For union-find algorithm, I added vertexMap variable to Graph definition which represents disjoint set of a graph. All values corresponding to each vertex are initialized as -1.

find(int e)

```
temp = vertexMap.get(e)
if (temp <0): { vertex is the root of a union
    return e

else:
    while vertexMap.get(temp) > 0: { finds the root of the union
        temp= vertexMap.get(temp)
    return temp
```

union(int a, int b)

```
if vertexMap.get(find(a)) <= vertexMap.get(find(b)) ⇒ a's partition has more elements than b's partition
    vertexMap.put(find(a), vertexMap.get(find(a)) + vertexMap.get(find(b)))
    vertexMap.put(find(b), find(a))        ~ according changes in vertexMap

else:
    vertexMap.put(find(b), vertexMap.get(find(a)) + vertexMap.get(find(b)))
    vertexMap.put(find(a), find(b))
```

totalLinkCost(graph)

```
sum= 0
for edge in graph.edges:
    sum+=edge.cost
return sum
```

cheapestNetwork(graph)

```
sum = 0
for i = 0 to graph.vertices.size():
    graph.vertexMap.put(i, -1)
PriorityQueue edges = new PriorityQueue(graph.edges)
TreeMap<String, Integer> vertexIndex = new TreeMap()
for i = 0 to graph.vertices.size():
    vertexIndex.put(graph.vertices.get(i), i)
while edges.size > 0 :
    Edge temp = edges.remove()
    if graph.find(vertexIndex.get(temp.src)) != graph.find(vertexIndex.get(temp.dst)):
        graph.union(vertexIndex.get(temp.src), vertexIndex.get(temp.dst))
        sum += temp.cost
return sum
```

Initializing the disjoint set

min priority queue for edges

map → key: name of vertex
→ value: index of vertex in vertexMap

while PQ is not empty

edge to inspect

if vertices are not in same partition

I also added compareTo method to Edge class to make PQ min heap.

savedAmount(graph)

```
return totalLinkCost(graph) - cheapestNetwork(graph)
```

Analysis

m: # of vertices
n: # of edges

find

In the worst case, the height of the tree can be logm at maximum because of the union implementation. Therefore there are clogm to reach to roof from a leaf in the worst case. Therefore, the time complexity is O(logm).

Since the function is not recursive, the space complexity is O(1).

Union

There are constant find calls. This corresponds to c.O(logn) complexity. There are also constant put calls which corresponds to c.O(1). The total time complexity is O(logn)

There are constant function calls inside union and it's not recursive, therefore the total space complexity is O(1).

totalLinkCost

We sum cost of each edge. Therefore we have (n-1) addition. So, the time complexity is O(n).

There is no other function call, therefore the space complexity is O(1).

cheapestNetwork

Initializing disjoint set's time complexity is O(m). I assume that initializing priority queue from graph.edges has O(nlogn) complexity because priority queue must be using heap sort. Initializing vertexIndex's time complexity is also O(m).

Then, we run the while loop n times. In the while loop, we run find twice and union once in each iteration in the worst case. Therefore, while loop's complexity is O(n logm).

So, the total time complexity is $O(n\log n) + O(m) + O(n\log m) = O(n(\log n + \log m) + m)$

There are constant function calls and the function is not recursive, therefore the total space complexity is O(1).

savedAmount

The time complexity is the sum of both functions, therefore it is $O(n(\log n + \log m) + m)$

The space complexity is also the sum of both functions' space complexity, which is O(1).

I have completed this assignment individually, without support from anyone else. I hereby accept that only the below listed sources are approved to be used during this assignment:

(i) Course textbook,

(ii) All material that is made available to me by the professor (e.g., via Blackboard for this course, course website, email from professor / TA),

(iii) Notes taken by me during lectures.

I have not used, accessed or taken any unpermitted information from any other source. Hence, all effort belongs to me.

Umur Berkay Karakaş