Comp202 HW4.1 Report

Time and Space Complexity Analysis:

UnionSet Class:

makeSet:

```
public void makeSet(int[][]array_graph){
 int set[]=new int[array_graph[0].length];

  for(int i=0; i<array_graph[0].length; i++){
    set[i]=-1;

  }
this.set=set;
}
```

Time Complexity:

In code above, there is a for loop that iterating over first row of the graph so time complexity is O(n) and that n is the number of vertices in the given graph.

Space Complexity:

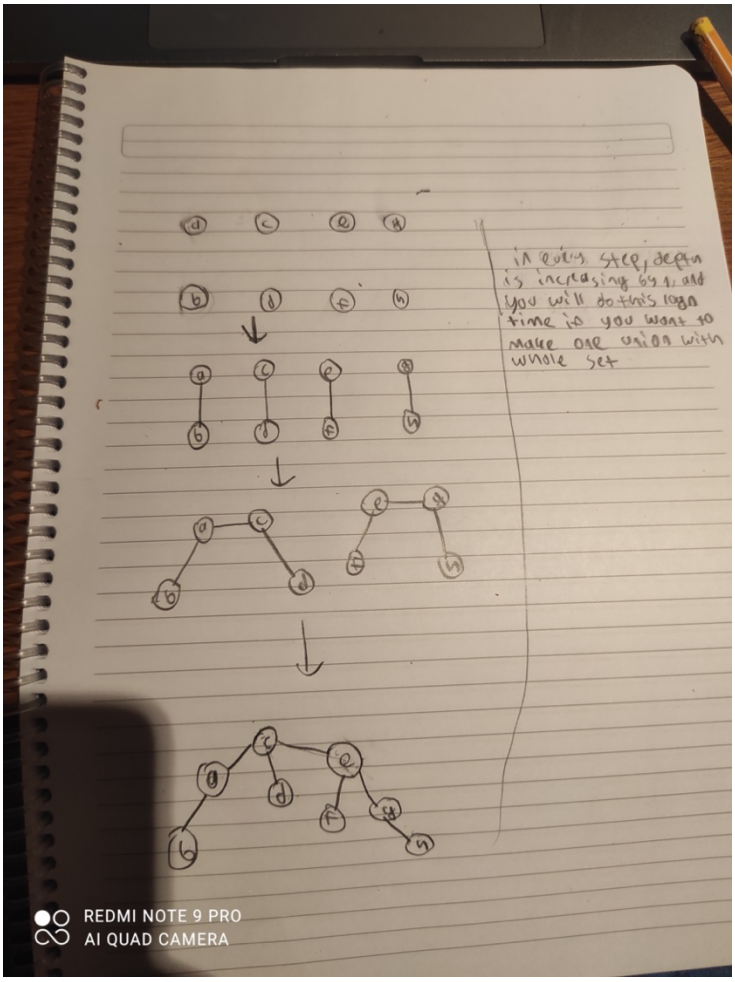
There is been created a set that contains all the vertices in the graph so space complexity is O(n).

find:

```
public int find(int i){

 if(set[i]<0){

  return i;
}
else{
  return find(set[i]);
}

 }
```

Time Complexity:

It's the depth of tree (union actually) in worst case because you go up in the tree until you find the root so its O(depth) and depth of tree is O(logn) in worst case because in makeUnion function, unions are made in order to merge the union with lower depth with the union with higher depth and that leads to tree's depth being O(logn). To prove that, lets consider the worst case: Worst case will be the situation that first combining all pairs of nodes, and then combining two of those two eaches and then combining 2 of them… I will put a photo below to make this statement clearer. This is the worst case because you change depth of that uinion in every makeUnion function, for example if you add(I mean makeUnion) one node to a pair of nodes you will waste a node and you don't increase the depth or if you add k-1 depth union to a k depth union, depth is still k. And for my worst case, depth is O(logn) because in every step, depth is increasing by 1, and you will do this logn time if you want to make one union with whole set.



Space Complexity:

In stack frame, depth many function will be pile so it's O(depth), as I proved above depth is O(logn) so space complexity is O(logn).

makeUnion:

```
void makeUnion( int i, int j){

  int x=set[find(i)];
  int y=set[find(j)];

  if(x>y){

    set[find(i)]=find(j);

    set[find(j)]+=x;
  }
  else{
    set[find(j)]=find(i);

    set[find(i)]+=y;
  }

  }
```

Time Complexity:

We are just finding roots of two index and than compare and change, find takes O(depth) time and others are O(1) so it's O(depth) and depth is O(logn) as I proved above so the time complexity is O(logn).

Space Complexity:

Even the function contain a recursive function, it will return and also nothing has created so space complexity is O(1).

HW4 Class:

totalLinkCost:

```
int totalLinkCost(Graph graph) {
```

```
    int sum=0;

    for(Edge e : graph.edges){
      sum+=e.cost;
    }
                    return sum;
            }
```

Time Complexity:

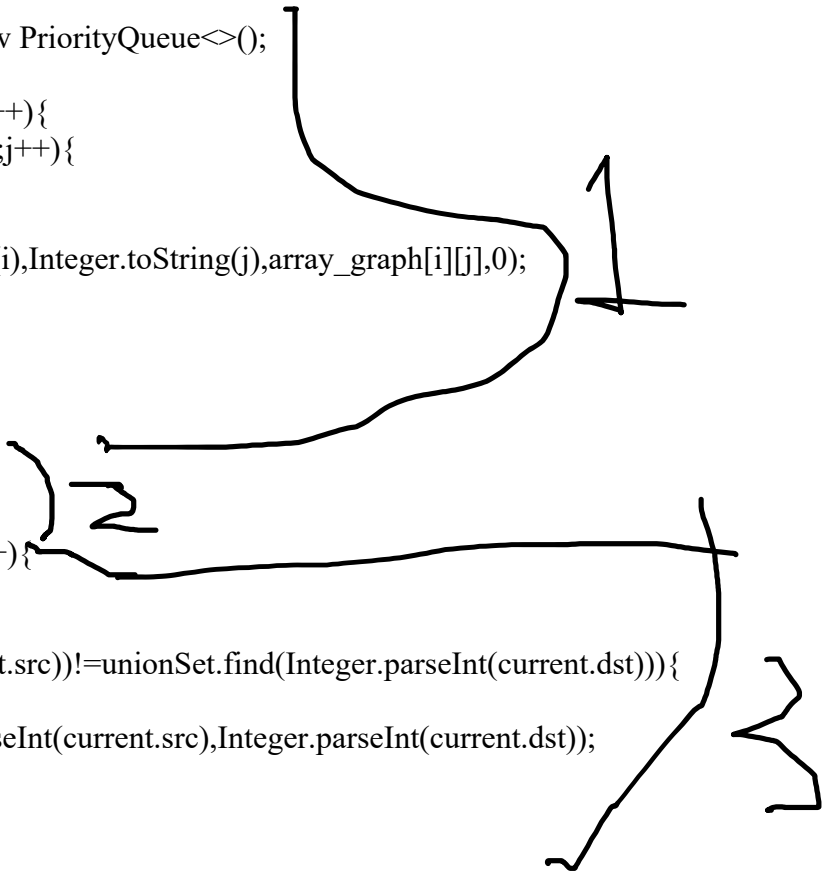
For all edges(O(m)), we are summing their costs(O(1)) so time complexity is O(m).

Space Complexity:

Nothing has been created so its O(1).

cheapestNetwork:

```
int cheapestNetwork(int[][] array_graph) {

    int sum=0;

    PriorityQueue<Edge> edgeHeap=new PriorityQueue<>();

    for(int i=0;i<array_graph[0].length;i++){
    for(int j=i+1;j<array_graph[0].length;j++){

      if(array_graph[i][j]>0){
      Edge e= new Edge(Integer.toString(i),Integer.toString(j),array_graph[i][j],0);
      edgeHeap.add(e);
      }
    }
  }
  UnionSet unionSet=new UnionSet();
  unionSet.makeSet(array_graph);
  Edge current;
  for(int i=0;i<array_graph[0].length;i++){
     current=edgeHeap.remove();

if(unionSet.find(Integer.parseInt(current.src))!=unionSet.find(Integer.parseInt(current.dst))){
         sum+=current.cost;
         unionSet.makeUnion(Integer.parseInt(current.src),Integer.parseInt(current.dst));
       }

  }
```



Time Complexity:

1) This part is $O(n^2 + m * logm)$ so its $O(n^2)$ (n is number of vertices and m is number of edges). n^2 comes from loop is itareted over above half of the matrice and there is $n^2/2$ elements in that range. M * logm part is a bit tricky, its not n^2 * logm but $n^2 + m * logm$ because you only add the edges to the heap and you pass if current element is 0 and you have m edges, heap's add method takes logm time so it's m * logm. Actually, m * logm is $O(n^2 * logn)$ because m is $O(n^2)$ because $m <= n*(n-1)/2$ so m is $O(n^2)$ and if we take logarithm of two sides, $logm <= log(n*(n-1)/2) = logn + log(n-1) -1 < = 2logn$, $logm <= 2logn$ so logm is O(logn) so m * logm is $O(n^2 * logn)$ so overall complexity is $O(n^2 +m * logm)$ but in lectures we always distinguish m and n so I want to leave the complexity as $O(n^2 + m * logm)$.

2) This is simple, there's just makeSet method that effects time complexity and its proven that makeSet is O(m).

3) There is a for loop that iterates n time (n is number of vertices) and inside that, there is removeMin method of heap and that's O(logm) and above that, there's set operations and they are O(logn) as I proved in that functions, so complexity is $O(n * (logm+logn))$ and that is O(n * logm).

Overall: Time complexity is $O(n^2 + m * logm) + O(m) + O(n * logm)$ so it's overall $O(n^2 + m * logm)$, it's not optimal because of the part that constructing edge heap but with adjacency matrix parameter, I couldn't resolved how to make that part more efficient because there is cost values of edges in the matrix but I belive that my union finding mechanism is optimal in terms of time complexity.

Space Complexity:

1) There is been created a heap with m elements so space complexity is O(m).

2) There is been created a disjoint set with n elements (n is the vertice number) so space complexity is O(n).

3) Nothing has been created so its O(1).
   O

Overall: Its $O(m) + O(n) + O(1)$ so its O(m) in order to m is probably bigger than n.

savedAmount:

```
int savedAmount(Graph graph) {

return totalLinkCost(graph)-cheapestNetwork(graph.asArray(false));

}
```

Time Complexity:

Its calling totalLinkCost (O(m)) and cheapestNetwork (O(n^2 + m * logm)) so the time complexity is O(n^2 + m * logm).

Space Complexity:

Its calling totalLinkCost (O(1)) and cheapestNetwork (O(m)) so the space complexity is O(m).

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 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.

Ali Taylan Akgülle