Pseudocode

totalTransitTime(graph):

```
sum = 0
vertexCount = graph.vertices.size()
Double distMatrix = Double[vertexCount][vertexCount]
for i in range(vertexCount):
    Arrays.fill(distMatrix[i], Double.POSITIVE_INFINITY)
    distMatrix[i][i] = 0.0
for edge in graph.edges:
    distMatrix[graph.vertices.indexOf(edge.src)][graph.vertices.indexOf(edge.dst)] = edge.latency
    distMatrix[graph.vertices.indexOf(edge.dst)][graph.vertices.indexOf(edge.src)] = edge.latency
for k in range(vertexCount):
    for i in range(vertexCount):
        for j in range(vertexCount):
            if(distMatrix[i][j] > distMatrix[i][k] + distMatrix[k][j]):
                distMatrix[i][j] = distMatrix[i][k] + distMatrix[k][j]
for i in range(vertexCount):
    for j in range(vertexCount):
        sum += distMatrix[i][j]
return sum
```

initialization of distance matrix

if there is a shorter path to j from i, change distMatrix[i][j]

cheapestTransitTime(graph)

```
Graph MST = new Graph()
for i=0 to graph.vertices.size():
    graph.vertexMap.put(i, -1)
PriorityQueue edges = new PriorityQueue(graph.edges)
while edges.size>0: -> while PQ is not empty
    Edge temp = edges.remove() -> edge to inspect
    if graph.find(graph.vertices.indexOf(temp.src)) != graph.find(graph.vertices.indexOf(temp.dst))
        graph.union(graph.vertices.indexOf(temp.src), graph.vertices.indexOf(temp.dst))
        MST.addEdge(temp) -> add edge to MST
return totalTransitTime(MST)
```

initializing the disjoint set

timeIncrease(graph)

return cheapestTransitTime(graph) − totalTransitTime(graph)

Analysis

this was opposite in part 2, I changed it because I think this is correct representation. ←

m = # of edges
n = # of vertices

totalTransitTime(graph):

There are some for loops with range vertexCount and one for loop with graph.edges and there are constant operations in the most inner for loop of nested for loops. Since m can be $\frac{n(n-1)}{2}$ at maximum in an undirected graph, we can assume that the for loop with range graph.edges is a for loop with range n^2. At most, we have 3 nested for loops with range(vertexCount) therefore the total time complexity is $O(n^3)$.

The largest data structure that is created is distMatrix and it has m^2 doubles, therefore the space complexity is $O(n^2)$.

If I used Dijkstra, I would iterate over each vertex using priority queue, find the incident edges and the corresponding adjacent vertices and change the values in the arrays accordingly. Since Dijkstra is run until all vertices are visited i.e removed from P.Q. and since removeMin takes logn time, this part contributes with nlogn to time complexity. Also, since we change cost of vertices that are adjacent to the vertex which is just visited in the iteration and since we do it $\sum_v deg(v) = 2m$ times, this part contributes with mlogn to time complexity. Therefore the total time complexity is $O((m+n)\log n)$.

The space complexity would be $O(n)$ since the largest data structure would be PQ and it would have n vertices.

cheapestTransitTime(graph)

Creating vertexMap is $O(n)$. Creating PQ from graph.edges and sorting is $O(m\log m)$ since PQ has m elements. While iteration is run m times and in each iteration, removeMin takes $O(\log m)$, find and union takes $O(\log n)$ (from previous HW part) and addEdge takes $O(1)$. totalTransitTime(mst) contributes with $O(m^3)$. So total time complexity is $O(n + m\log m + m(\log m + \log n) + m^3) = O(n + m\log n + m^3)$

The total space complexity would be $O(m+n^2)$. n^2 comes from totalTransitTime. m comes from PQ and MST. There is also n that comes from MST but I ignored it since I have n^2.

time Increase (graph)

The total time complexity would be sum of two functions, which is $O((m+n)(\log n + m^3))$.

The total space complexity would be the sum as well, which is $O(m+n^2)$.

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