CSE 591: Foundation of Algorithms

Programming Assignment# 1

Arun Karthikeyan 1208595982

Ponneeswaran Natarajan 1208815591

Vishnu Priya Chandra Sekar 1207848859

Algorithm:

We have a single-source shortest-path problem and to this problem we have an algorithm inspired from Dijkstra's algorithm. We are given a weighted, directed graph with n cities and m roads. We are also given the option of building a new roads like a road going from city 0 to city 2 with length 3. We consider this new edge to city 2 separate and add a new map to the graph as city 2'(prime). After this we run the Dijkstra's algorithm with a check to include only one prime edge is visited in a single path and keep track of all such paths with length lower than the shortest path without visiting any prime city edges.

```
MODIFIED_DIJKSTRA (G, w, s)
1 INITIALIZW-SINGLE-SOURCE (G, s)
2 S = Φ
3 Q = G.V
4 while Q ≠ Φ
5       u = EXTRACT-MIN (Q)
6       S = S ∪ {u}
7       for each vertex v ϵ G.Adj[u]
8           if new optional edge is not visited in the path at least once
9               RELAX (u, v, w, optionalEdgeUsed)


RELAX (u, v, w, optionalEdgeUsed)
1 if v.d > u.d + w (u, v)
```

2 v.d = u.d + w (u, v)

3 v. π = u

4 globalOptionalEdgeUsageMap.add(u,v)

Proof of Optimality:

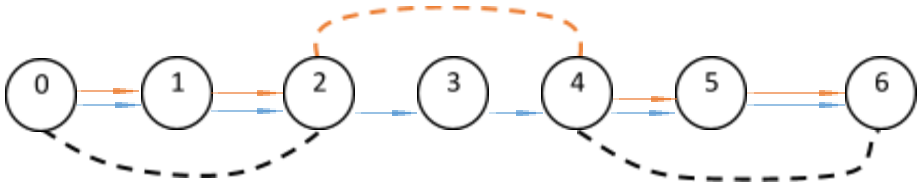
We have implemented our algorithm to **keep track of shortest path using every optional road**, and also without using any optional road and a **check to visit the new optional edges only once in a path**. The base algorithm is the same as Dijkstra's with a little more checks for consistency and constraints, So same proof of optimality for the Dijkstra's algorithm can be applied here and if Dijkstra's algorithm provides the shortest path then our modified Dijkstra's algorithm also finds the shortest path in the given conditions. The time complexity of our algorithm is O(m + k + nlogn) where O(m + nlogn) is the running time of Dijkstra's algorithm. K is the number of new edges and c is a constant.

Programming Language used: Java

Reasons: We have not used any external libraries for this assignment. We needed priority queue for the implementation and Java has this feature built in.

Sample Output:

1.



Existing path
New path
Shortest path without new edge
Shortest path with new edge

Input:

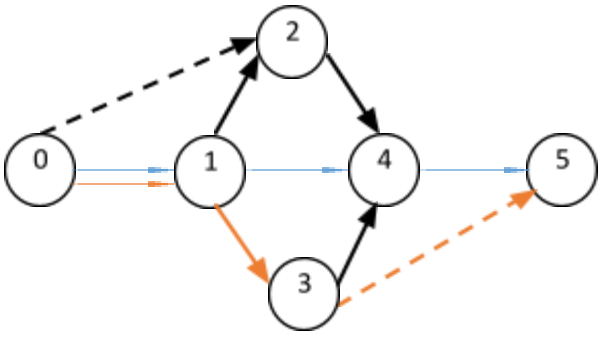
7
6
0:1:3
1:2:1
2:3:3
3:4:4
4:5:3
5:6:1
3
0:2:2
2:4:2

4:6:2
0
6

Output:

EDGE INFO : SHORTEST PATH LENGTH
2 --> 4 : 10
4 --> 6 : 13
0 --> 2 : 13
Above listed edge(s) can be used to shorten the length of the shortest path s-t
Shortest path s-t length with using only pre-existing roads : 15

2.



Existing path
New path
Shortest path without new edge
Shortest path with new edge

Input:

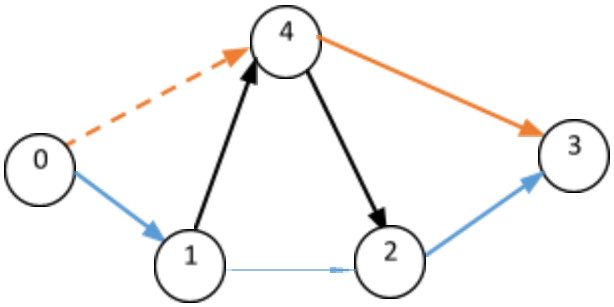
6
7
0:1:3
1:2:3
1:3:3
1:4:3
2:3:3
3:4:3
4:5:3
2
0:1:2
0:3:1

0
5

Output:

EDGE INFO : SHORTEST PATH LENGTH
0 --> 3 : 7
0 --> 1 : 8
Above listed edge(s) can be used to shorten the length of the shortest path s-t
Shortest path s-t length with using only pre-existing roads : 9

3.



Existing path
New path
Shortest path without new edge
Shortest path with new edge

Input:

5
6
0:1:5
1:2:5
1:4:5
2:3:5
2:4:5
4:3:6
1
0:4:5
0
3

Output:

EDGE INFO : SHORTEST PATH LENGTH
0 --> 4 : 11
Above listed edge(s) can be used to shorten the length of the shortest path s-t
Shortest path s-t length with using only pre-existing roads : 15

4.

Existing path

Input:

4
2
0:1:5
2:3:10
0
0
3

Output:

\Programming\Workspaces>java ProgrammingAssignment1
Enter choice of input(1.Console Input | 2.File Input)
1
4
2
0:1:5
2:3:10
0
0
3
Shortest Path cannot be found