Question#3

Algorithm idea:

With input of Graph in adjacent list (n vertices and m edges) and a dictionary of edges with its corresponding costs. First I need to generate a dictionary of all node and the value is node’s current cost so far, also generate a dictionary to keep track of the node’s parent. And then run BFS to get all connected node from s (initial node). In BFS algorithm, I need to add a condition which is to check if new value of current cost is less than the node which explored by this node, if yes, update the value to lower one and set the parent of that node to current node. After BFS, just trace back from the t(target node) if explored, I can get my shortest path in a minimum cost.

Algorithm detail:

#input: Edges:key=edge,value=cost; Graph V,E

parent = {}

list = []

cost = {}

answer = []

list.append(s)

cost[s]=0

for node in Graph:

cost[node]=∞

while ! list.empty(): #BFS while

temp = list.pop()

for item in Graph[temp]:

if cost[item]>cost[temp]+Edges[(temp,item)]: #check the cost

cost[item]= cost[temp]+Edges[(temp,item)]

list.append(item)

parent[item]=temp

#end of BFS traceback if cost[t]!= ∞

If cost[t]!= ∞: #trace back to get answer list

answer.append(t)

item = parent[t]

While item!=s:

answer.append(item)

item=parent[item]

answer.append(s)

answer.reverse()

return answer

Proof idea:

As the problem state that the Graph is a DAG which mean I will never go back to the node that explored before, so it is a always straight line. So my algorithm sum up the minimum cost by compare the target node’s current value to the value that current node’s value+the cost of edge. And in order to do that, I use BFS algorithm walk through entire Graph. So in the end I can get a minimum cost of shortest path.

Proof detail:

With the idea of BFS, I am able to walk through the entire Graph. The parent dictionary keep track of which node is connect to which. So I am able to trace back the dictionary in order to obtain answer list( if s somewhat to connect t). The different part of my algorithm, I also keep track of the cost of current node, so I can get a way to calculate the next node’s cost. Therefore, I can keep updating the value of the cost and push the node into the queue(list) for further execution. Lastly, since it is a DAG, means that the input is a direct Graph with no cycle in it. So I can go through the entire graph with my algorithm with no possibility of running into an infinite loop. As a result in the end, I can get a minimum cost shortest path from t to s.

Runtime Analysis:

Initialize cost dictionary: O(n) times

BFS while loop: O(m)times

Reverse: O(n) times

Overall: O(n)+O(m)+O(n)=O(m+n) times