Assignment 2

On

Shortest Path

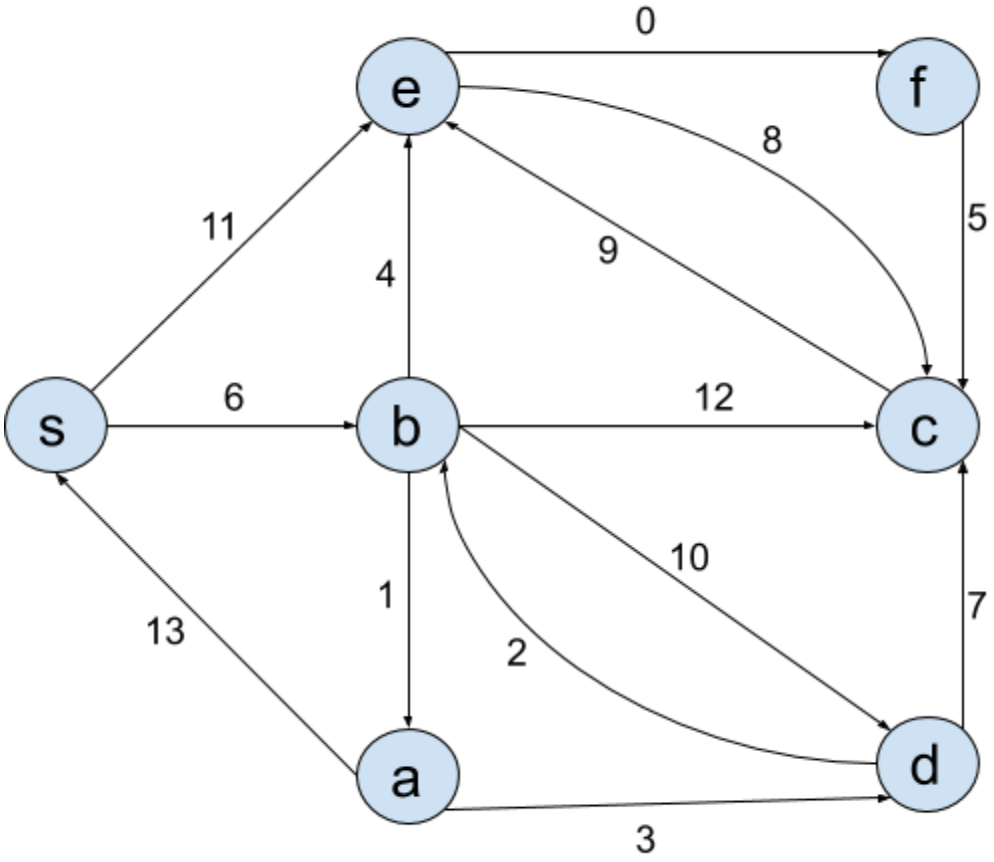
Name : Md. Mushfiqui Haque Omi
ID : 180041140
Section : A
Course no. : CSE 4404

Problem 1:Collaborators

1. Atiqur Rahman(ID: 180041123)

Problem 2(a):

The weighted graph from the given representation is given below:



Problem 2(b):

The weight of the shortest path from s to all nodes are as follows:
d[s]=0;
d[b]=d[s]+cost[s][b]=0+6=6
d[a]=d[b]+cost[b][a]=6+1=7
d[e]=d[b]+cost[b][e]=6+4=10
d[f]=d[e]+cost[e][f]=10+0=10
d[d]=d[a]+cost[a][d]=7+3=10
d[c]=d[f]+cost[f][c]=10+5=15

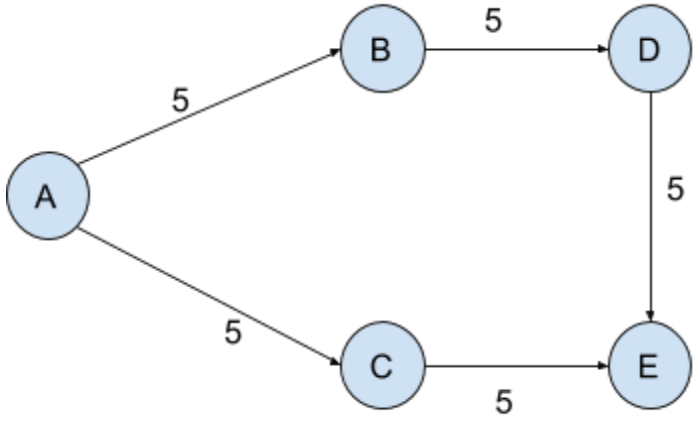
Problem 2(c):

The order in which the vertices are removed from Dijkstra queue is:

s > b > a > e > d > f > c

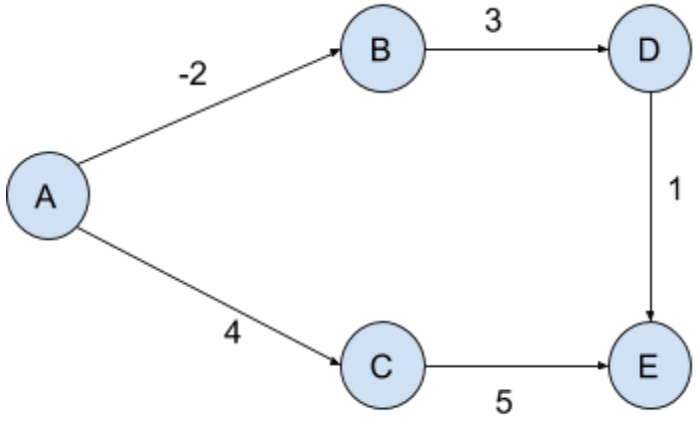
Problem 3:

Without modifying BFS, we can use it to find the shortest path in a weighted graph if weights of all the edges are same. For example:



From this graph we can see we can go from A to E using two paths: ABDE and ACE. Now if we run BFS without modifying we will get ACE as the shortest path as there are less number of edges we need to traverse to go from A to E. And using Dijkstra's algorithm we also get ACE as the shortest path. So for same weight edges we can use BFS without modifying.

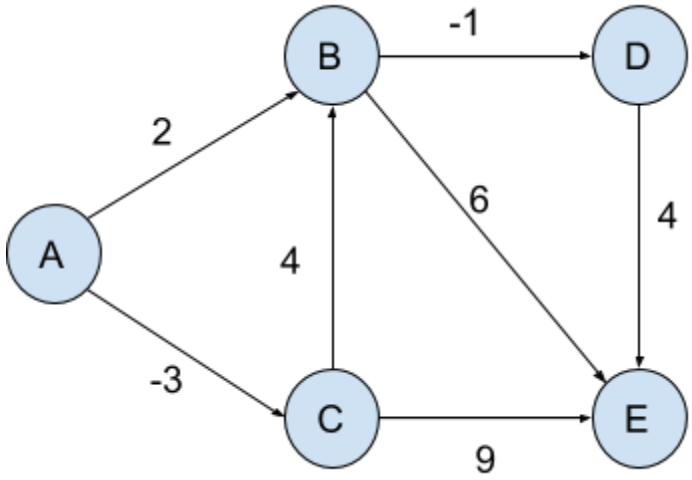
Now let's consider a graph which has different edges with different weights.Now if we run BFS without modifying we will get ACE as the shortest path.



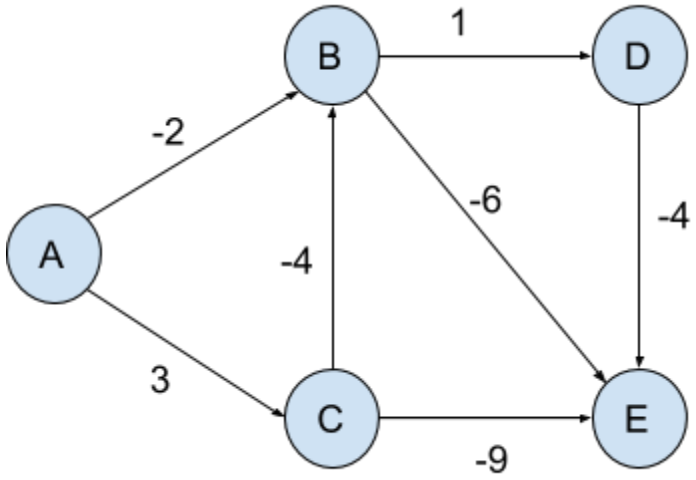
Using Dijkstra's algorithm we get ABDE as the shortest path. So not modifying BFS will give us a wrong path.So we can conclude that if the weights of every edge is same, we can find the shortest path of an unweighted graph without modifying BFS. But if the weights are not same we must modify BFS for finding the shortest path.

Problem 4(a):

We can consider this graph:



1. After negating all the edge width of the graph:



2. Now there is no negative edge cycle in the graph so we can use Bellman-Ford algorithm to find the shortest path. Here,

d[A]=0
d[B]=-2
d[C]=3
d[D]=-1
d[E]=-8

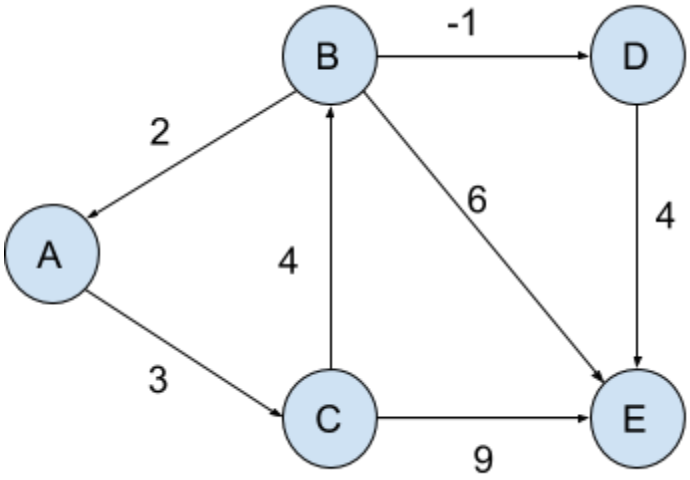
3. After negating all the distance found:

d[A]=0
d[B]=2
d[C]=-3
d[D]=1
d[E]=8

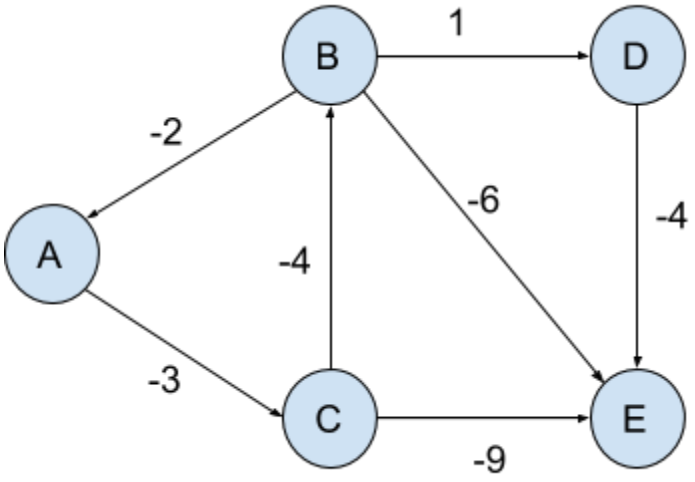
And these are the longest distance from the source node to any particular node

Problem 4(b):

In Starney's Algorithm after negating all the edge weight we need to run Bellman_Ford. And we know Bellman-Ford can not find the shortest path of a graph if there is a negative edge cycle. So at first if there is a positive edge cycle and we negate the edge weight it will turn into a negative edge cycle where we can not use Bellman-Ford. In this case, Starney's algorithm doesn't work. So Starney's algorithm can not find the longest path in a graph having a positive edge cycle. We can give the graph shown below:



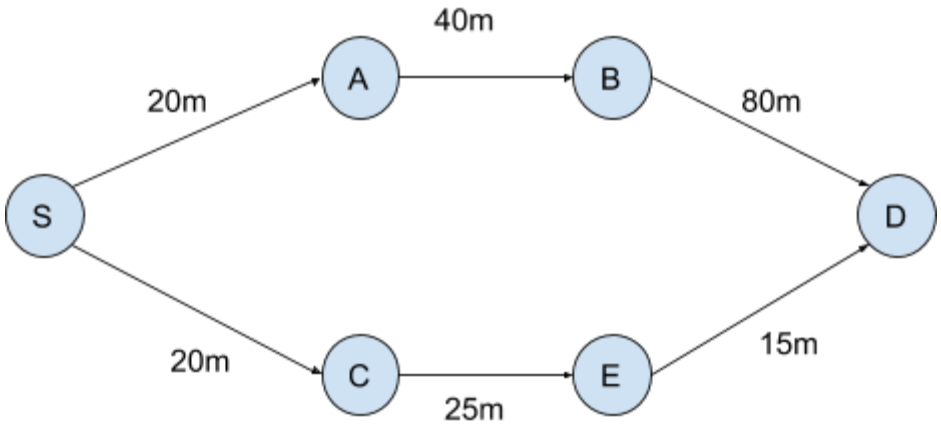
In this graph there is a positive edge cycle between A,B,C. After negating all the edge weight we get:



Now we need to run Bellman-Ford to find the shortest distances. But as we can see there is a negative edge cycle between A,B and C, Bellman-Ford algorithm will fail to find the shortest path. So Starney's algorithm will not work.

Problem 5:

In Pert Charts some tasks are dependent on other task.So if we construct a graph using these dependencies we will get a weighted DAG (Directed Acyclic Graph). Then we can use topological sort so that we can maintain these dependencies. Let's consider a scenario:



Here S is the starting point of the Pert chart and D is the destination. Here B is dependent on A, and E is dependent on C. Each of the tasks take a certain amount of time to complete.We can assume that we can do tasks optimally which means we can do two tasks at the same time. So in this scenario we can do A and B at first 20 minutes. After that we can do B and E, Here B takes 40 minutes, but E takes only 25 minutes. So if we can finish B then by that time E will be finished as E takes less time. So we can conclude if we take the longest path to reach from S to D it will indicate the shortest time to complete all the activities.

As it is a DAG, there will be no loop. So we can traverse each node and maintain a distance variable from the source node. We will update the distance while traversing if we find a longer path than the previous path. As we are traversing each vertices and their adjacent vertices we can run this algorithm in linear time.

Algorithm:

1. Start
2. Initialize the distance of all the nodes as minus infinite and the distance to the source as 0
3. Create a topological order of all the nodes.
4. Process every node and update distance for every adjacent vertex.
5. For every vertex u do the followings:
   For every adjacent vertex v of u:
   if (distance[v] < distance[u] + cost(u, v))
   then distance[v] = distance[u] + cost(u, v)
6. Finish

Correctness of the Algorithm:

First we are creating topological order to maintain the dependencies. We are updating the distance of every node from the source node taking the longest path. If we take the longest path to reach to the destination from the source node, all the processes will be completed when we reach the destination

Time Complexity:

We know for topological sort time complexity is O(V+E). Then we need to process all nodes and for every node, it runs a loop for all adjacent vertices. The total adjacent vertices in a graph are O(E). So the inner loop will run O(V+E) times. So the overall time complexity for this algorithm is O(V+E).