Walchand College of Engineering, Sangli
Computer Science & Engineering
Third Year

Course: Design and analysis of algorithm Lab

Lab course coordinator:
Dr. B. F. Momin- Batch: - T6, T7, T8
Mr. Kiran P. Kamble- Batch: - T1, T2, T3, T4, T5

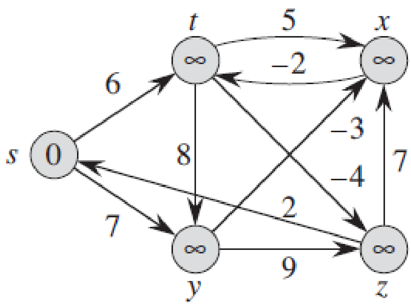
Week 9 Assignment

Dynamic Programming

Name: Trupti Rajendra Patil

Prn: 2020BTECS00051

Q) From a given vertex in a weighted connected graph, Implement shortest path finding Bellman-Ford algorithm.



Algorithm:

1. Initialize the distances from the source to all vertices as infinite and distance to the source itself as 0.
2. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.
3. This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.
4. Do following for each edge u-v
   If dist[v] > dist[u] + weight of edge uv, then update dist[v] to
   dist[v] = dist[u] + weight of edge uv
5. This step reports if there is a negative weight cycle in the graph. Again, traverse every edge and do following for each edge u-v
   If dist[v] > dist[u] + weight of edge uv, then "Graph contains negative weight cycle"
   The idea of step 5 is, step 4 guarantees the shortest distances if the graph does not contain a negative weight cycle. If we iterate through all edges

one more time and get a shorter path for any vertex, then there is a negative weight cycle.

Code:

```
     #include<bits/stdc++.h>

using namespace std;

void BellmanFord(vector<vector<int>>& edges,int n,int src){
    // vector to store the distance for node from src
    vector<int> distance(n,1e8);

    // initially src distance is 0
    distance[src]=0;

    // iterate for n-1 times through all edges
    for(int i=0;i<n-1;i++){
        for(auto it: edges){
            int u=it[0];
            int v=it[1];
            int w=it[2];

            // if u is reached and distancce[u]+w if less than
distance[v]
            // then update distance[v]
            if(distance[u]!=1e8 && distance[u]+w < distance[v]){
                distance[v] = distance[u] + w;
            }
        }
    }

    bool flag=false;
    // Check for negative cycle
    // Nth relaxation
    for(auto it: edges){
        int u=it[0];
        int v=it[1];
        int w=it[2];

        if(distance[u]!=1e8 && distance[u]+w < distance[v]){
            flag=true;
        }
```

```
    }

    if(flag==true){
        cout<<"Negative cycle present"<<endl;
    }else{
        cout<<"Vertex   Distance from Source\n"<<endl;
        for (int i = 0; i < n; ++i)
            cout<<i<<"           "<< distance[i]<<endl;
    }
}

int main(){
    // n- no. of vertices
    // m- no. of edges
    int n=5,m=8;
    // cin>>n>>m;

    vector<vector<int>> edges={
        {0,1,6},{0,2,7},{1,3,5},{1,4,-4},{1,2,8},{2,3,-
3},{2,4,9},{3,1,-2},
        {4,3,7},{4,0,2}
    };

    // for(int i=0;i<m;i++){
    //     vector<int> temp;
    //     for(int j=0;j<3;j++){
    //         int x;
    //         cin>>x;
    //         temp.push_back(x);
    //     }
    //     edges.push_back(temp);
    // }

    int src=0;
    // cin>>src;

    BellmanFord(edges,n,src);
}
```

Output:

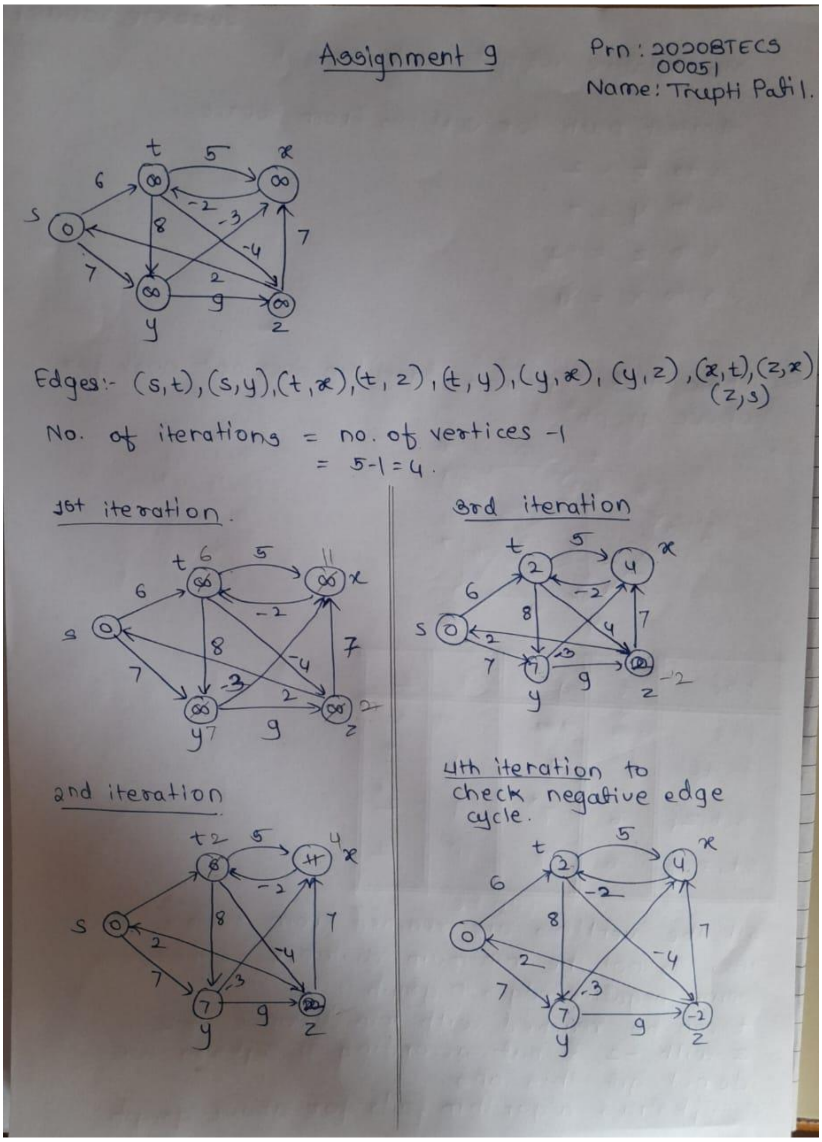
```
PS C:\Users\trupti patil\OneDrive\Desktop\ACADEMICS\SEM5\DAA\ExpQ> cd
CADEMICS\SEM5\DAA\ExpQ\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tem
unnerFile }
Vertex   Distance from Source

0          0
1          2
2          7
3          4
4          -2
PS C:\Users\trupti patil\OneDrive\Desktop\ACADEMICS\SEM5\DAA\ExpQ>
```

Complexity Analysis:

Time complexity: O (V * E), where V is the number of vertices in the graph and E is the number of edges in the graph

Space Complexity: O(E)



2020BTECS00051

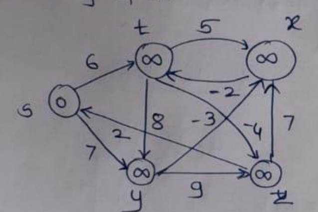
No negative weight cycle present.

Shortest path for vertices from source.

s to t = 2
s to y = 7
s to z = -2
s to x = 4.

Q. Show that Dijkstra's algorithm does not work for above graph.

→



{s}	t	y	x	z
{s}	6 ~~∞~~	7 ~~∞~~	∞	∞
{s,t}	6*	7	11	2*
{s,t,z}	6*	7*	9	2*
{s,t,z,y}	6	7	4*	2

All the vertices ~~are~~ can be reached from source, but this is not the minimum distance as there are some negative edges in graph.
t can be reached with min distance of 2, z with -2 & but according to Dijkstra we do not get this ans.
∴ Dijkstra's algorithm fails for above graph.

Q) Show that Dijkstra's algorithm does not work for above graph

Q) Given a weighted, directed graph G = (V. E) with no negative-weight cycles, let m be the maximum over all vertices v belongs to V of the minimum number of edges in a shortest path from the source s to v. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in m+1 passes, even if m is not known in advance.

Ans:

We can simply implement this optimization of Bellman ford algorithm by remembering if v was relaxed or not.

If v is relaxed then we wait to see if v was updated (which means being relaxed again).
If v was not updated, then we would stop.

Because the greatest number of edges on any shortest path from the source is m, then the path-relaxation property tells us that after m iterations of Bellman Ford, every vertex v has achieved its shortest-path weight in v.d. By the upper-bound property, after m iterations, no d values will ever change. Therefore, no d values will change in the (m+1)st iteration. Because we do not know m in advance, we cannot make the algorithm iterate exactly m times and then terminate. But if we just make the algorithm stop when nothing changes any more, it will stop after m + 1 iteration.