Experiment Number: 8

Aim: - Write a program to find the single source shortest path using Dynamic Programming.

Problem Statement: - Finding Single Source Shortest Path in a directed graph with Negative edge lengths, using Bellman Ford Algorithm.

Theory:-

In Single source shortest path problem, a directed graph G = (V, E) with n vertices is given. Edges can have negative lengths, but it should not form a cycle of negative length. So there will be a shortest path between any two vertices of an n-vertex graph that has at most n-1 edges on it. Elimination of the cycles from the path results in another path with the same source and destination. If $dist^l[u]$ is the length of a shortest path from the source vertex v to vertex u, then it contains at most l edges.

Hence, $dist^1[u] = cost[v,u]$ $1<=u<=n$ and $dist^{n-1}[u]$ is the length of an unrestricted shortest path from v to u. Algorithm computes $dist^{n-1}[u]$ for all u. It is implemented using Dynamic programming methodology as follows:

1. If the shortest path from v to u with at most k, k>1, edges has no more than k-1 edges, then $dist^k[u] = dist^{k-1}[u]$.
2. If the shortest path from v to u with at most k, k>1, edges has exactly k edges, then it is made up of a shortest path from v to some vertex j followed by the edge <j,u>. The path from v to j has k-1 edges, and its length is $dist^{k-1}[j]$. All edges i such that the edge <i,u> is in the graph are candidates for j. Algorithm finds the value of i that minimizes $dist^{k-1}[u] + cost[i,u]$

So we have the following recurrence to find single source shortest path:

$$dist^k[u] = \min \{ dist^{k-1}[u], \min_i \{ dist^{k-1}[i] + cost[i,u] \} \}$$

Given the seven vertex graph, together with the arrays $dist^k$, k= 1, 2,….6. These arrays are computed using the recurrence given above.

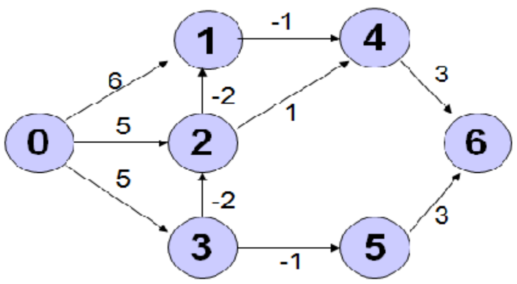


Fig.1. Shortest path with negative edge lengths

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

Fig.2. $dist^k$

Algorithm: -

```
BellmanFord(v, cost, dist, n)
        {
        for i =1 to n do // Initialize dist.
                Dist[i] =cost[v,i];
         for k =2 to n-1 do
             for each u such that u ! =v and u has at least one incoming edge do
                    for each <i,u> in the graph do
                           if dist[u] > dist[i]  + cost[i,u] then
                                  dist[u] = dist[i]  + cost[i,u];
        }
```

The complexity of single source shortest path is O (ne) where n is the number of vertices and e is the number of edges in the graph.

NAME: HARSHBHAI SOLANKI
ROLL.N0:62
SE-4-D

EXPERIMENT N0-8

AIM: Write a program to find the single source shortest path using Dynamic Programming.

CODE:

```
#include<stdio.h>
#include<stdlib.h>

int Bellman_Ford(int DistMat[20][20], int numVertex, int E, int edge[20][2])
{
   int i,u,v,k,distance[20],parent[20],S,flag=1;
   for(i=0;i<numVertex;i++)
   {
      distance[i]=1000,parent[i]=-1;
   }
   printf("Enter Source Vertex Number :-->");
   scanf("%d",&S);
   distance[S-1]=0;
   for(i=0;i<numVertex-1;i++)
   {
      for(k=0;k<E;k++)
      {
         u=edge[k][0], v=edge[k][1];
         if(distance[u]+DistMat[u][v]<distance[v])
         {
            distance[v]=distance[u]+DistMat[u][v] ,
            parent[v]=u;
```

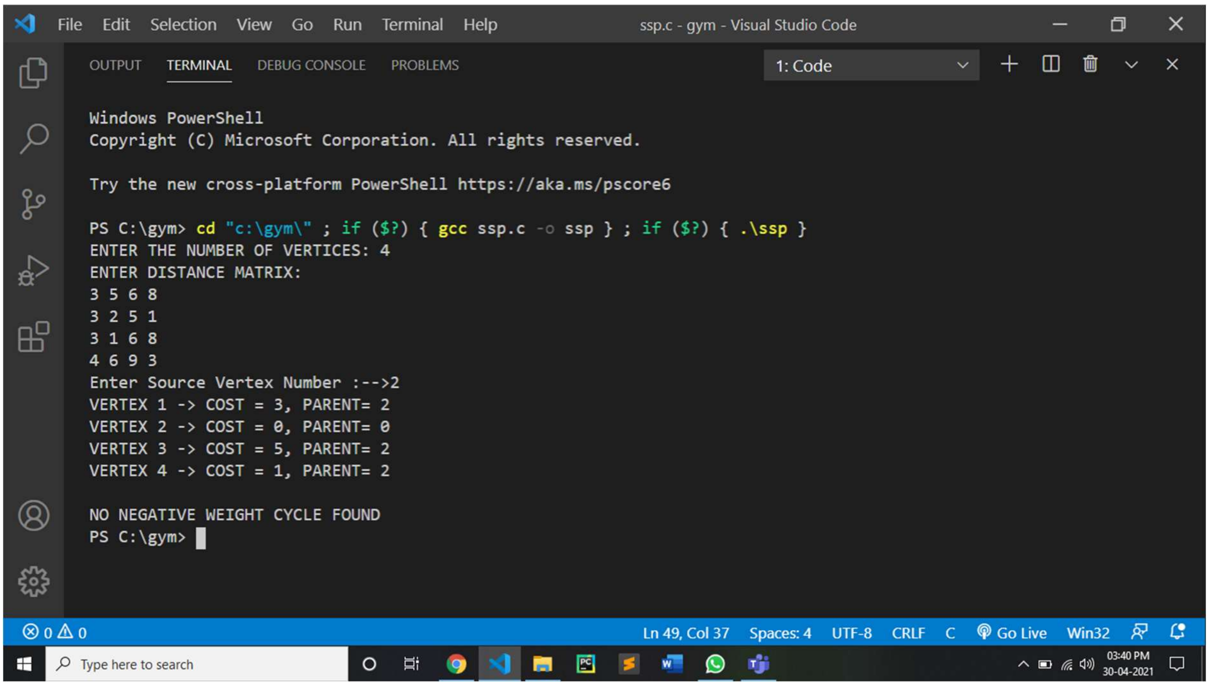
```
            }
        }
    }
    for(k=0;k<E;k++)
    {
        u=edge[k][0], v=edge[k][1];
        if(distance[u]+DistMat[u][v]<distance[v])
        {
            flag=0;
    }
}
if(flag)
{
    for(i=0;i<numVertex;i++)
    {
        printf("VERTEX %d -> COST = %d, PARENT=
%d\n",i+1,distance[i],parent[i]+1);
    }
}
return flag;
}

int main()
{
    int numVertex,edge[20][2],DisMat[20][20],i,j,k=0;
    printf("ENTER THE NUMBER OF VERTICES: ");
    scanf("%d",&numVertex);
    printf("ENTER DISTANCE MATRIX:\n");
```

```
    for (i = 0; i < numVertex; i++)
    {
        for ( j = 0; j< numVertex; j++)
        {
            scanf("%d",&DisMat[i][j]);
            if(DisMat[i][j]!=0)
            {
                edge[k][0]=i,edge[k++][1]=j;
            }
        }
    }
    if(Bellman_Ford(DisMat,numVertex,k,edge))
    {
        printf("\nNO NEGATIVE WEIGHT CYCLE FOUND ");
    }
    else{
        printf("\nNEAGTIVE WEIGHT CYCLE EXISTS");
    }
    return 0;

}
```

<u>**OUTPUT:**</u>



<u>CONCLUSION:</u>

By performing the above experiment we can conclude it is slower than dijkstra’s algorithm but it can handle negative weight cycle.

The time complexity of algorithm is O(|V|.|E|) where the V is the number of vertices and E is the number of Edges. And the space complexity is O(V)