NAME: Aditya Somani
PRN: 71901204L
ROLL: T1851061

SL-5(Group'B')

DAA Assignment No. (B)3

AIM: Write a program to implement Bellman-Ford Algorithm using Dynamic Programming and verify the time complexity.

OBJECTIVE:

1. To understand the concept of Dynamic Programming.
2. To study Bellman-Ford Algorithm.
3. To find Time Complexity of Bellman-Ford Algorithm.

THEORY:

Dynamic Programming:

Dynamic programming is a powerful technique for solving problems that might otherwise appear to be extremely difficult to solve in polynomial time. Algorithms built on the dynamic programming paradigm are used in many areas of CS, including many examples in AI (from solving planning problems to voice recognition).

Dynamic programming works by solving subproblems and using the results of those subproblems to more quickly calculate the solution to a larger problem. Unlike the divide-and-conquer paradigm (which also uses the idea of solving subproblems), dynamic programming typically involves solving all possible subproblems rather than a small portion. Often, dynamic programming algorithms are visualized as "filling an array" where each element of the array is the result of a subproblem that can later be reused rather than recalculated.

One use of dynamic programming is the problem of computing "all pairs shortest paths" in a weighted graph. Although Dijkstra's algorithm solves the problem for one particular vertex, it would be necessary to run it for every vertex, and would be a somewhat complicated algorithm. Dynamic programming produces a simpler, cleaner algorithm (though one that is not inherently faster).

The trick here is to find a useful subproblem that, by solving, we can use to solve a larger problem. In this case, it turns out that a useful subproblem is to consider the shortest path for any pair of vertices that goes through only the first k vertices. For simplicity, we'll just number each vertex 1 through n, where n is the total number of vertices. Now, each subproblem is just to find the distance from vertex i to vertex j using only the vertices less than k.

Bellman Ford Algorithm to find shortest path

This algorithm is used for single source shortest path on a graph with negative weights but no negative cycles.

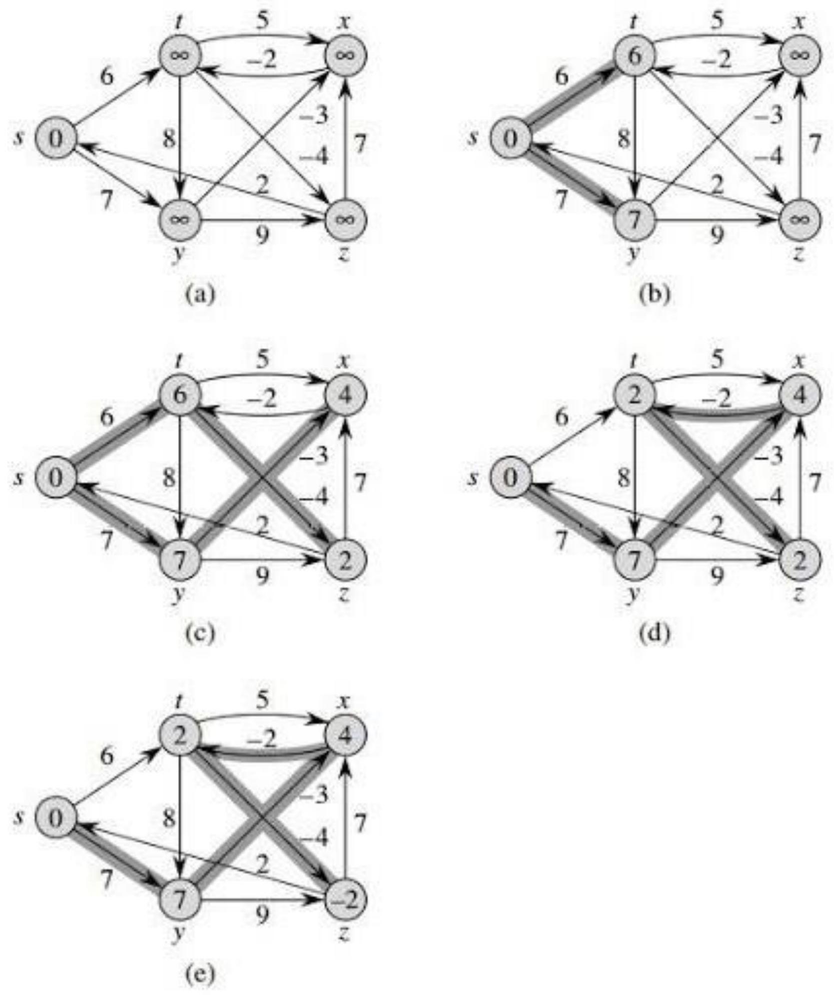
Here are some points to keep in mind regarding the different algorithms studied so far:

1. Breadth First Search : For graphs where the edge-weights are either zero or all same.

2. Dijkstra's Alogrithm : For graphs where the edge-weights are non-negative. Uses greedy strategy.

3. Bellman-Ford Algorithm : For graphs where the edge-weights may be negative, but no negative weight cycle exists. Uses dynamic programming.

An example graph taken from Introduction to Algorithms:



Algorithm:

Input: Graph and a source vertex src.

Output: Shortest distance to all vertices from src. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycles is reported.

1. Initialize distances from source to all vertices as infinite and distance to source itself as zero. Create an array distance[] of size |v| with all values as infinite except distance[src] where src is source vertex.
2. Repeat following |v|-1 times.
   Do following for each edge u-v.

If distance[v]> dist[u]+weight of edge(u,v) then update distance[v]
distance[v]= dist[u]+weight of edge(u,v)

3. This step reports if there is a negative weight cycle in graph.
   Repeat following for each edge(u-v)
   If distance[v]> dist[u]+weight of edge(u-v) then
   Graph contains negative weight cycles.

Time Complexity:

O(VE)

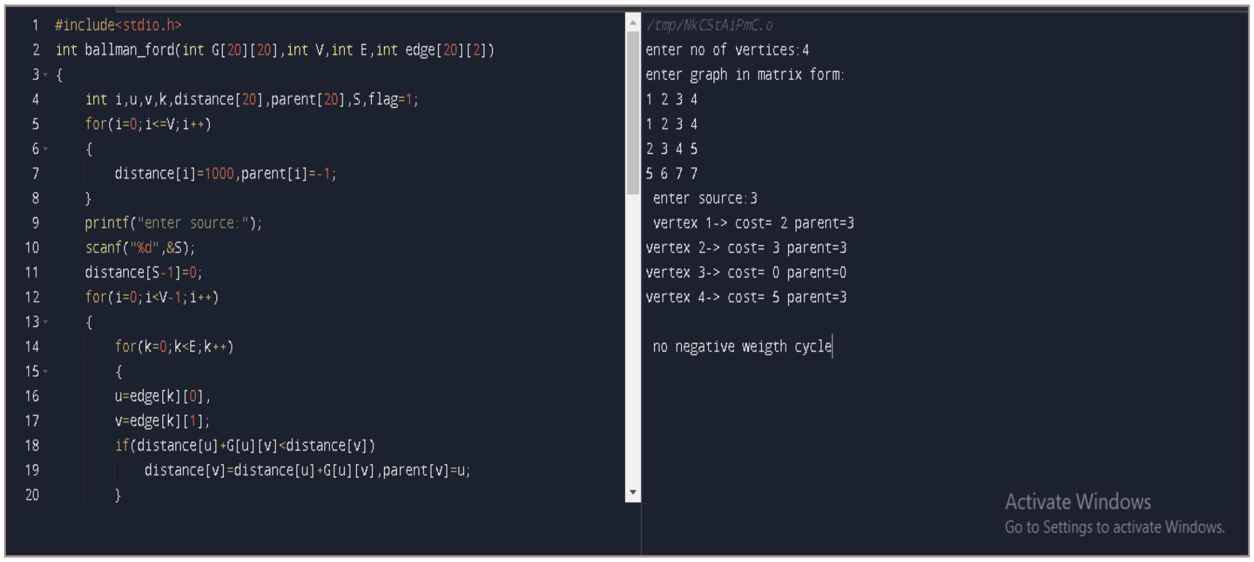
PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
int Bellman_Ford(int G[20][20] , int V, int E, int edge[20][2])
{
   int i,u,v,k,distance[20],parent[20],S,flag=1;
   for(i=0;i<V;i++)
      distance[i] = 1000 , parent[i] = -1 ;
      printf("Enter source: ");
      scanf("%d",&S);
      distance[S-1]=0 ;
   for(i=0;i<V-1;i++)
   {
      for(k=0;k<E;k++)
      {
         u = edge[k][0] , v = edge[k][1] ;
         if(distance[u]+G[u][v] < distance[v])
            distance[v] = distance[u] + G[u][v] , parent[v]=u ;
      }
   }
   for(k=0;k<E;k++)
      {
        u = edge[k][0] , v = edge[k][1] ;
        if(distance[u]+G[u][v] <
            distance[v])flag = 0 ;
    }
    if(flag)

        for(i=0;i<V;i++)
                                printf("Vertex %d -> cost = %d parent =
                                    %d\n",i+1,distance[i],parent[i]+1);


    return flag;
}
int main()
{
    int V,edge[20][2],G[20][20],i,j,k=0;
    printf("BELLMAN
    FORD\n");    printf("Enter
    no.    of    vertices:    ");
    scanf("%d",&V);
    printf("Enter graph in matrix form:\n");
    for(i=0;i<V;i+
        +)
        for(j=0;j<V
        ;j++)
        {
            scanf("%d",&G[i][j]);
            if(G[i][j]!=0)
                edge[k][0]=i,edge[k++][1]=j;
        }
    if(Bellman_Ford(G,V,k,edge))
        printf("\nNo negative weight
        cycle\n");
    else printf("\nNegative weight cycle exists\n");
    return 0;
}
```

Conclusion:

Thus we have implemented Bellman-Ford Algorithm using Dynamic Programming andverified the time complexity.

OUTPUT:-



```
#include<stdio.h>
int ballman_ford(int G[20][20],int V,int E,int edge[20][2])
{
    int i,u,v,k,distance[20],parent[20],S,flag=1;
    for(i=0;i<=V;i++)
    {
        distance[i]=1000,parent[i]=-1;
    }
    printf("enter source:");
    scanf("%d",&S);
    distance[S-1]=0;
    for(i=0;i<V-1;i++)
    {
        for(k=0;k<E;k++)
        {
        u=edge[k][0],
        v=edge[k][1];
        if(distance[u]+G[u][v]<distance[v])
            distance[v]=distance[u]+G[u][v],parent[v]=u;
        }
```

```
/tmp/NkCStAiPmC.o
enter no of vertices:4
enter graph in matrix form:
1 2 3 4
1 2 3 4
2 3 4 5
5 6 7 7
 enter source:3
 vertex 1-> cost= 2 parent=3
vertex 2-> cost= 3 parent=3
vertex 3-> cost= 0 parent=0
vertex 4-> cost= 5 parent=3

 no negative weigth cycle
```

Activate Windows
Go to Settings to activate Windows.