**Experiment No: 8**

**AIM:** Implementation Bellman Ford algorithm (Dynamic Programming) and obtaining its step count.

**THEORY:**

Bellman Ford algorithm is used to find the shortest path from a source vertex to all the other vertices in a graph. The major difference between this algorithm and the Dijkstra’s single source shortest path algorithm is that the latter fails for graphs with negative weight edges.

Bellman Ford is also simpler than Dijkstra’s algorithm and suites well for distributed systems. However, it fails when there is a negative weight cycle in the graph. Hence it is also not perfect.

**Algorithm writing:**

1. First, the distances from source vertex to all other vertices are initialised to infinity and the distance to the source itself is initialised to 0.
2. This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.  
    Do following for each edge u-v  
   ………………If dist[v] > dist[u] + weight of edge uv, then update dist[v]  
   ………………….dist[v] = dist[u] + weight of edge uv
3. This step reports if there is a negative weight cycle in graph. 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 3 is that step 2 guarantees the shortest distances if the graph doesn’t 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.*

**ALGORITHM:**

AlgorithmBellmanFord(v, cost,dist,n)

// Single-source/all-destinations shortest

// paths with negative edgecosts

{

**for** i :=1to n **do**// Initialize dist.

dist[i]:=cost[v,i];

**for** k :=2 to n – 1 **do**

**for** each u such that 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

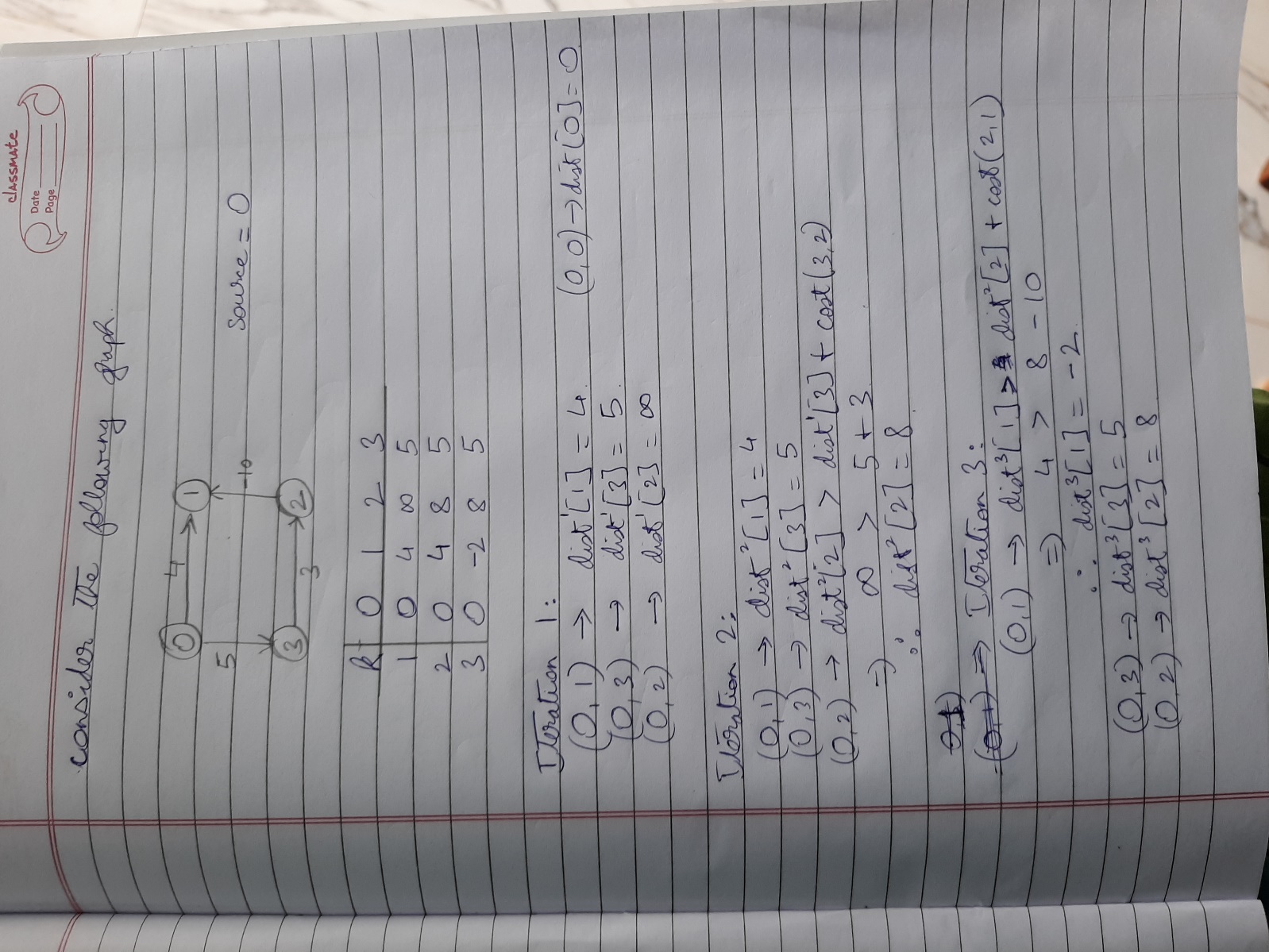
}

*Time Complexity*

• Time Complexity of Bellman Ford is O(VE) or O()

* If the shortest paths are to be computed in a complete graph, then the time complexity of this algorithm increases to O(). This is because in a complete graph, the total number of edges are e\*(e-1)/2.

*Problem Tracing*



PROGRAM IMPLEMENTATION:

#include<iostream>

using namespace std;

int count=0;

void bellford(int graph[][3], int v, int e)

{

int dist[v]; //array of distances from source to all v

dist[0]=0; //dist from source to source itself is zero

count++;

for(int i=1;i<v;i++)

dist[i]=99, //99 represents infinity

count+=2; //assignment and for

count++; //end of for loop

for(int i=1;i<v;i++)

{

count++; //for

for(int j=0;j<e;j++)

{

count+=2; //for and if

if(dist[graph[j][0]] + graph[j][2] < dist[graph[j][1]])

{

dist[graph[j][1]] = dist[graph[j][0]] + graph[j][2];

count++;

}

}

count++; //inner for loop

}

count++; //outer for loop

for(int i=0;i<e;i++)

{

count+=2; //for and if

if(dist[graph[i][0]] + graph[i][2] < dist[graph[i][1]])

cout<<"Negative weight cycle detected. Cannot compute shortest path\n",

count++, //for exit;

exit(1);

}

count++; //for loop end

cout<<"vertex distance from source:\n";

for(int i=0;i<v;i++)

cout<<i<<"\t"<<dist[i]<<endl;

}

int main()

{

int v,e;

cout<<"Enter the number of vertices:";

cin>>v;

cout<<"\nEnter the number of edges:";

cin>>e;

cout<<"Enter 8 directed edges in the order : Source,destination,Weight\n";

int total\_edges = v\*(v-1)/2;

count++;

int graph[total\_edges][3];

for(int i=0;i<e;i++)

{

cout<<"Enter edge:"<<i+1<<endl;

cin>>graph[i][0];

cin>>graph[i][1];

cin>>graph[i][2];

}

bellford(graph,v,e);

cout<<"Stepcount="<<count<<endl;

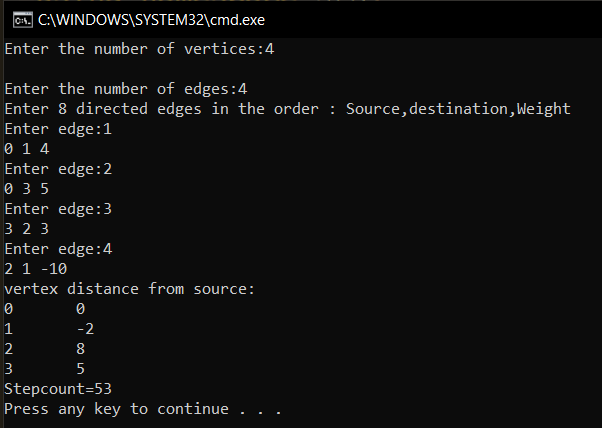
return 0;

}

OUTPUTS:

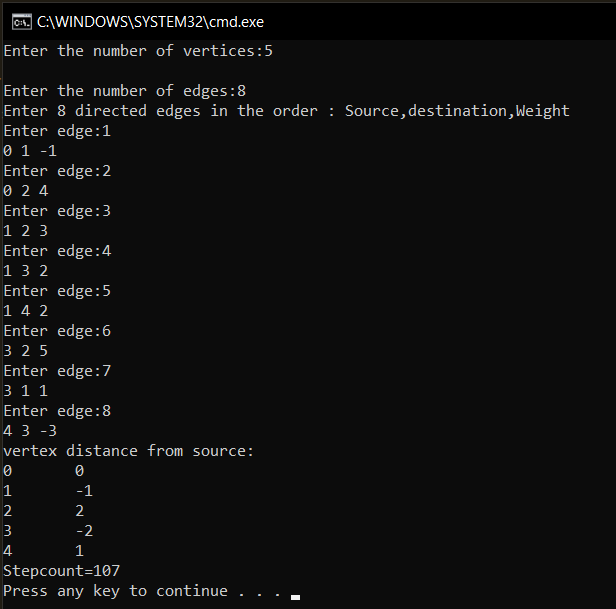
1. When v=4 and e=4

**Count=53**

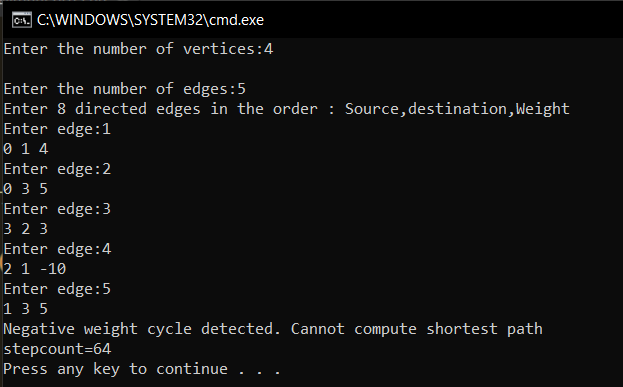


1. When v=5 and e=8

**Count=107**

****

1. When negative weight cycle is present

****

**Conclusion**:

1. **Bellman Ford algorithm works for general weights of a graph, unlike Dijkstra’s algorithm, which fails in the presence of negative weights. However, the former fails to compute shortest paths in the presence of negative weighted cycle.**
2. **The time complexity of Bellman Ford algorithm ranges from** O() **to** O()