EXPERIMENT 8

Experiment No: 8 **Date:** 27/04/2021

Aim: Implementation of BellmanFord algorithm (Dynamic Programming) and estimate its step count

Theory:

BellmanFord Algorithm

- Solves single shortest path problem in which edge weight may be negative but no negative cycle exists.
- This algorithm works correctly when some of the edges of the directed graph G may have negative weight.
- When there are no cycles of negative weight, then we can find out the shortest path between source and destination.
- It is slower than Dijkstra's Algorithm but more versatile, as it capable of handling some of the negative weight edges.

Algorithm

```
ALGORITHM bellmanFord(v,cost,dist,n)

//Single-source/ all -desitnation shortest

// paths with negative edges

{

    for i:= to n do

        dist[i] = cost[v,i]

    for k:=2 to n-1 do
```

EXPERIMENT 8

```
                for each u such that u not equal to v has at least one
                incoming edge
            for each <i,u> in graph do
                if dist[u]>dist[i]+cost[i][u] then
                    dist[u]  :=  dist[i]+cost[i,u]
    }
```

Algorithm writing

- 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 cycle is reported.
- This step initializes distances from the source to all vertices as infinite and distance to the source itself as 0.
- Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.
- 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

EXPERIMENT 8

- 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, 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

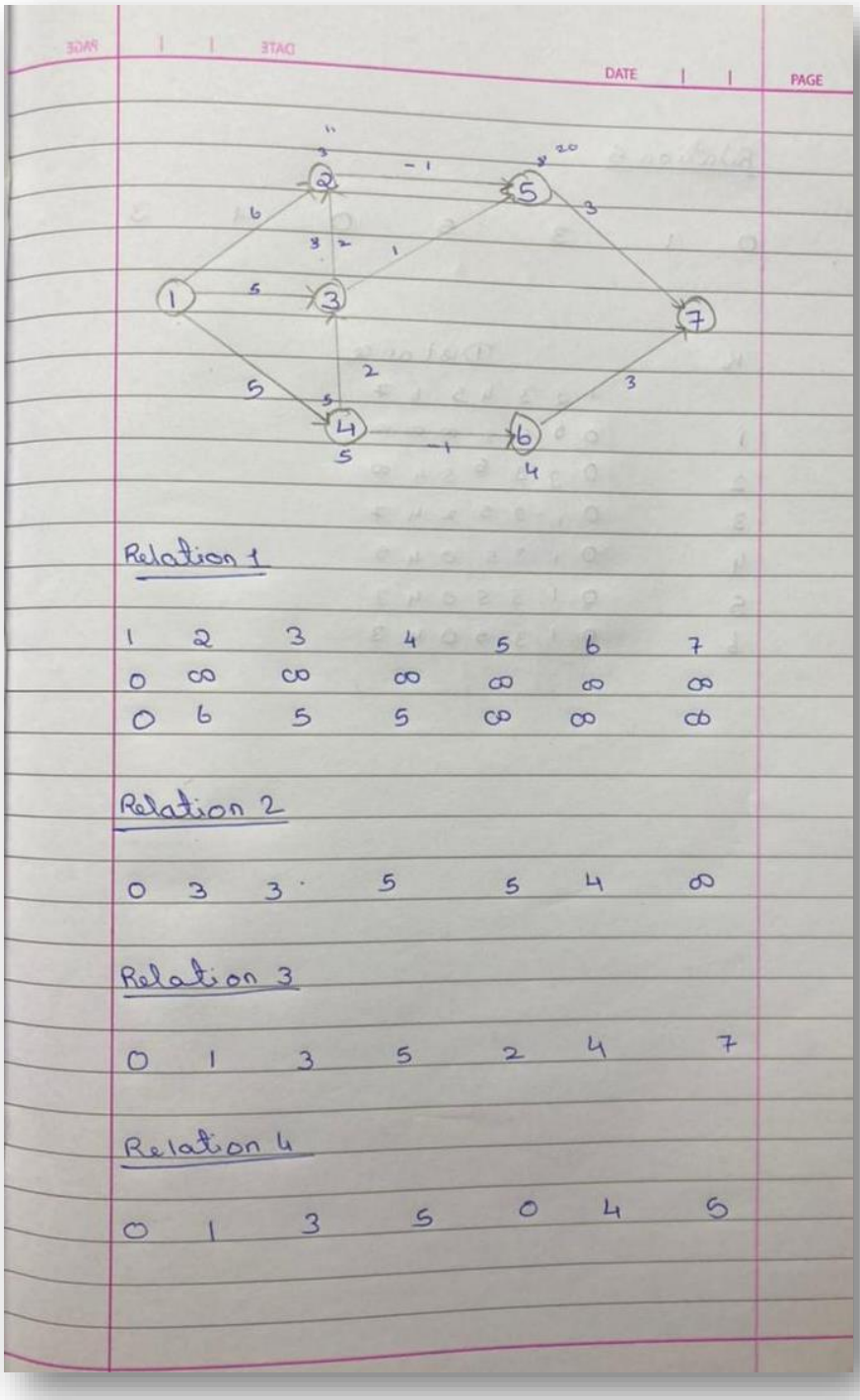
Complexity

- The running time of Bellman-Ford is O(VE) , where V is the number of vertices and E is the number of edges in the graph.
- On a complete graph of n vertices, there are around n2 edges, for a total running time of n3 .
- That starts to become impractical if you have more than a few thousand vertices.

VEDANT ALIAS SURYAJI SANJAY MALKARNEKAR
201205008

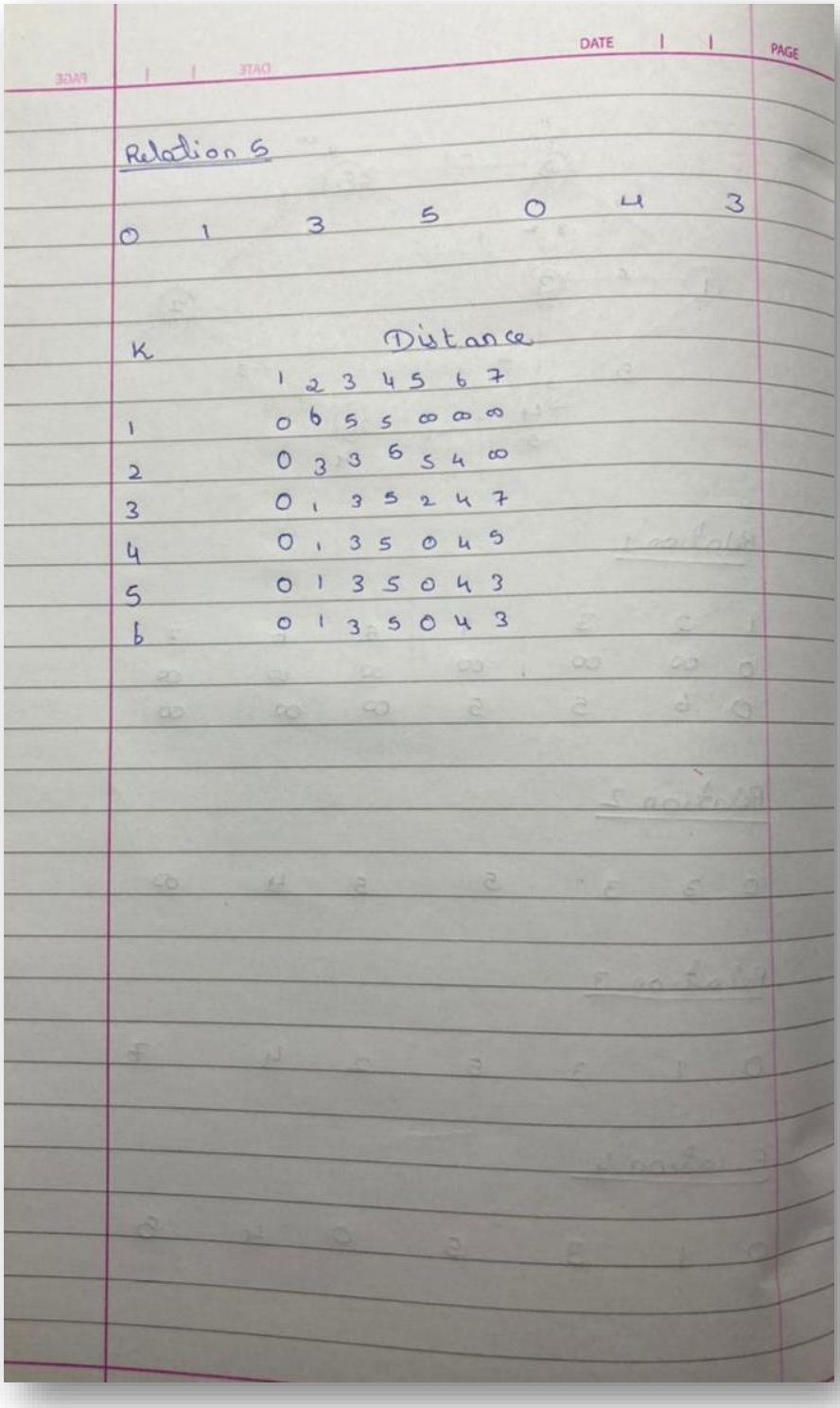
EXPERIMENT 8

Sample Problem Solving



VEDANT ALIAS SURYAJI SANJAY MALKARNEKAR
201205008

EXPERIMENT 8



DATE / / PAGE

Relations

0 1 3 5 0 4 3

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

EXPERIMENT 8

Program

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

using namespace std;

int c=0;

typedef struct  edge{

        int x,y,w;

};


void path(vector<int> parent,int i)

{

        c++;

        if(parent[i]==-1)

        {

                c++;

           return;

        }


        path(parent,parent[i]);

        c++;

        cout<<" - "<<i<<" ";
```

EXPERIMENT 8

```
}



void printPath(vector<int> parent,int src,int n)
{
        for(int i=0;i<=n;i++)
        {
                c++;
                c++;
                cout<<"Path to  "<<i<<" from "<<src<<" : ";
                cout<<src<<" ";
                path(parent,i);

                c++;
                cout<<endl;
        }
        c++;
}
```

EXPERIMENT 8

```
void bellmonFord(vector<edge> v,int n,int e,int S)
{
	vector<int> dist(n+1);
	for(int i=1;i<=n;i++)
	dist[i] = INT_MAX,c++;
	c++;
	dist[S] = 0;
	vector<int> parent(n+1,-1);
	for(int i=1;i<n;i++)
	{
		c++;
		for(int j=0;j<e;j++)
		{
			c++;
			c++;
			int src = v[j].x;
			c++;
			int dest = v[j].y;
			c++;
			int W = v[j].w;
```

EXPERIMENT 8

```
        c++;
        if(dist[src]!=INT_MAX&&dist[dest]>dist[src]+W)
                {
                        c++;
                         dist[dest] = dist[src]+W;
                         c++;
                         parent[dest] = src;
                }
                c++;
        }
        c++;
}
for(int j=0;j<e;j++)
        {
                c++;
                c++;
                int src = v[j].x;
                c++;
                int dest = v[j].y;
                c++;
```

EXPERIMENT 8

```
        int W = v[j].w;
        c++;
        if(dist[dest]>dist[src]+W)
        {
                c++;
                cout<<"Negetive Edge Cycle"<<endl;


                c++;
                return;
        }
}
c++;
for(int i=1;i<=n;i++)
{
        c++;
        c++;
        cout<<"The Distance From "<<S<<" To "<<i<<"
        = "<<dist[i]<<endl;
}
c++;
```

EXPERIMENT 8

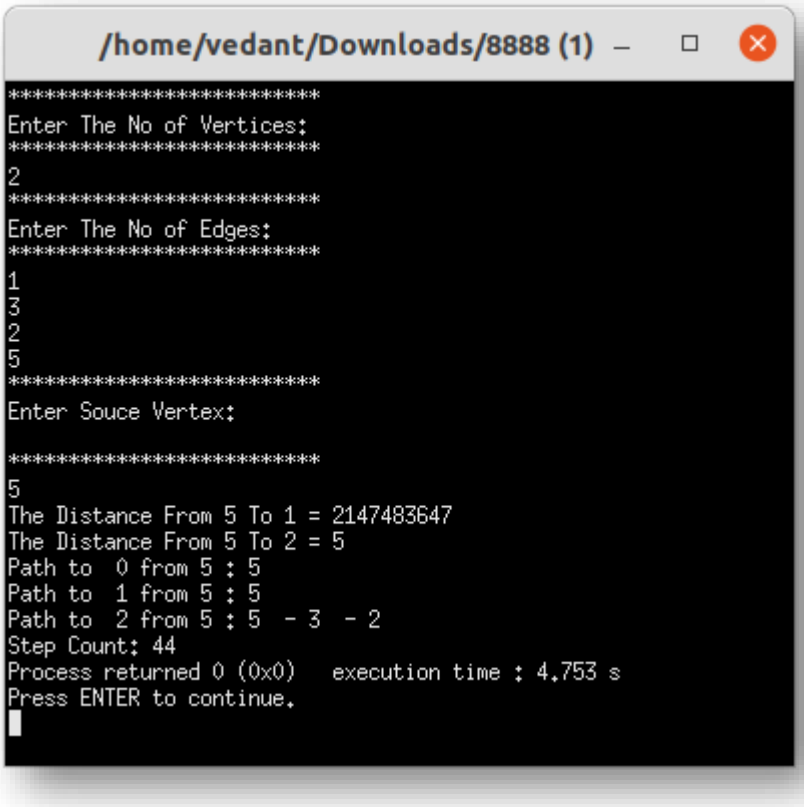
```
        printPath(parent,S,n);
}
int main()
{
    int n,e;
    cout<<"*************************"<<endl;
    cout<<"Enter The No of Vertices: ";
    cout<<"\n*************************"<<endl;
    cin>>n;
    cout<<"*************************"<<endl;
    cout<<"Enter The No of Edges: ";
    cout<<"\n*************************"<<endl;
    cin>>e;
    vector<edge> v(e);
    for(int i=0;i<e;i++)
    {
        int x,y,w;
        cin>>x>>y>>w;
        v[i].x=x;
        v[i].y=y;
```

EXPERIMENT 8

```
        v[i].w=w;
    }
    int S;
    cout<<"*************************"<<endl;
    cout<<"Enter Souce Vertex: "<<endl;
    cout<<"\n*************************"<<endl;
    cin>>S;
    bellmonFord(v,n,e,S);
    cout<<"Step Count: "<<c;
}
```

EXPERIMENT 8

Output



Conclusion

- Detailed concept of BellmanFord Algorithm (Dynamic Programming) was studied successfully.
- Program using BellmanFord Algorithm was executed successfully.
- The step count for the BellmanFord Algorithm was obtained.