

Registration Number: 19BCE2119

Name: Gaurav Kumar Singh

Course: Network and Communication CSE1004

Digital Assignment 4

Bellman-Ford Algorithm

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

using namespace std;

struct edge{
    int src,dst,wt;
};

int V,E;

void bellmanFord(vector<edge>& Edges)
{
    int parent[V];
    int cost_parent[V];
    vector<int> value(V,INT_MAX);
    parent[0] = -1;
    value[0] = 0;

    bool updated;
    for(int i=0;i<V-1;++i)
    {
        updated = false;
        for(int j=0;j<E;++j)
        {
            int U = Edges[j].src;
            int V = Edges[j].dst;
            int wt = Edges[j].wt;
```

```

        if(value[U]!=INT_MAX and value[U]+wt<value[V])
        {
            value[V] = value[U]+wt;
            parent[V] = U;
            cost_parent[V] = value[V];
            updated = true;
        }
    }
    if(updated==false)
        break;
}

for(int j=0;j<E and updated==true;++j)
{
    int U = Edges[j].src;
    int V = Edges[j].dst;
    int wt = Edges[j].wt;
    if(value[U]!=INT_MAX and value[U]+wt<value[V])
    {
        cout<<"Graph has -VE edge cycle\n";
        return;
    }
}

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

    cout<<"U->V: "<<parent[i]<<"->"<<i<<" Cost to reach "<<i<<" from source 0 =
"<<value[i]<<"\n";
}

int main()
{

```

```

cout<<"Enter no. of vertices and edges:\n";

cin>>V>>E;

vector<edge> Edges(E);

int src,dst,wt;

cout<<"\nEnter the edges with their weight in the format `Source Destination Weight` in a
directed manner: \n";

for(int i=0;i<E;++i)
{

    cin>>src>>dst>>wt;

    Edges[i].src = src;

    Edges[i].dst = dst;

    Edges[i].wt = wt;

}

bellmanFord(Edges);

return 0;

}

```

The screenshot shows the Code::Blocks IDE with the following components:

- main.cpp (Line 48-75):**

```

48     }
49
50     for(int i=1;i<V;++i)
51         cout<<"Q->V: "<<parent[i]<<"->"<<i<<endl;
52
53 }
54
55 int main()
56 {
57     cout<<"Enter no. of vertices and edges\n";
58     cin>>V>>E;
59     vector<edge> Edges(E);
60
61     int src,dst,wt;
62     cout<<"\nEnter the edges with their weight in the format `Source Destination Weight` in a
63     directed manner: \n";
64     for(int i=0;i<E;++i)
65     {
66         cin>>src>>dst>>wt;
67         Edges[i].src = src;
68         Edges[i].dst = dst;
69         Edges[i].wt = wt;
70     }
71     bellmanFord(Edges);
72     return 0;
73 }
74
75

```
- Output Window (C:\Users\DELL\Desktop\CN\Lab\DA4\Bellman_ford\bin\Debug\Bellman_ford.exe):**

```

Enter no. of vertices and edges:
5 7
Enter the edges with their weight in the format `Source Destination Weight` in a directed manner:
0 1 1
1 4 2
4 2 7
4 3 3
2 2 6
2 1 5
2 0 4
0->V: 0->1 Cost to reach 1 from source 0 = 1
0->V: 4->2 Cost to reach 2 from source 0 = 10
0->V: 4->3 Cost to reach 3 from source 0 = 6
0->V: 1->4 Cost to reach 4 from source 0 = 3
Process returned 0 (0x0)   execution time : 51.783 s
Press any key to continue.

```

Dijkstra Algorithm

```
#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);

int main() {

    int G[MAX][MAX], i, j, n, u;

    printf("Enter no. of vertices:");

    scanf("%d", & n);

    printf("\nEnter the adjacency matrix:\n");

    for (i = 0; i < n; i++)

        for (j = 0; j < n; j++)

            scanf("%d", & G[i][j]);

    printf("\nEnter the starting node:");

    scanf("%d", & u);

    dijkstra(G, n, u);

    return 0;

}

void dijkstra(int G[MAX][MAX], int n, int startnode) {

    int cost[MAX][MAX], distance[MAX], pred[MAX];

    int visited[MAX], count, mindistance, nextnode, i, j;

    for (i = 0; i < n; i++)

        for (j = 0; j < n; j++)

            if (G[i][j] == 0)

                cost[i][j] = INFINITY;

            else

                cost[i][j] = G[i][j];

    for (i = 0; i < n; i++) {

        distance[i] = cost[startnode][i];

        pred[i] = startnode;
```

```

    visited[i] = 0;
}
distance[startnode] = 0;
visited[startnode] = 1;
count = 1;
while (count < n - 1) {
    mindistance = INFINITY;
    for (i = 0; i < n; i++)
        if (distance[i] < mindistance && !visited[i]) {
            mindistance = distance[i];
            nextnode = i;
        }
    visited[nextnode] = 1;
    for (i = 0; i < n; i++)
        if (!visited[i])
            if (mindistance + cost[nextnode][i] < distance[i]) {
                distance[i] = mindistance + cost[nextnode][i];
                pred[i] = nextnode;
            }
    count++;
}
for (i = 0; i < n; i++)
    if (i != startnode) {
        printf("\nDistance of node%d=%d", i, distance[i]);
        printf("\nPath=%d", i);
        j = i;
        do {
            j = pred[j];
            printf("<-%d", j);
        } while (j != startnode);
    }
}

```

}

The screenshot shows the Code::Blocks IDE with a C program for Dijkstra's algorithm. The program is named 'main.c' and is located in the 'Dijkstra' project. The code defines a graph with 4 vertices and an adjacency matrix. The main function prompts the user to enter the number of vertices, the adjacency matrix, and the starting node. It then calls the 'dijkstra' function to calculate the shortest path from the starting node to all other nodes. The output of the program is displayed in the console window, showing the shortest path from node 1 to node 2 is 0, from node 1 to node 3 is 2, and from node 1 to node 4 is 2. The execution time is 82.266 s.

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX], int n, int start)
{
    int cost[MAX][MAX], distance[MAX], pred;
    int visited[MAX], count, mindistance, m;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (i == j) cost[i][j] = 0;
            else cost[i][j] = INFINITY;
        }
    }
    for (i = 0; i < n; i++)
    {
        cost[i][i] = 0;
        distance[i] = cost[start][i];
    }
    visited[start] = 1;
    count = 1;
    while (count < n)
    {
        mindistance = INFINITY;
        for (m = 0; m < n; m++)
        {
            if (!visited[m] && distance[m] < mindistance)
            {
                start = m;
                mindistance = distance[m];
            }
        }
        visited[start] = 1;
        count++;
        for (j = 0; j < n; j++)
        {
            if (!visited[j] && distance[j] > distance[start] + cost[start][j])
            {
                pred[j] = start;
                distance[j] = distance[start] + cost[start][j];
            }
        }
    }
}

int main()
{
    int G[MAX][MAX], i, j, n, u;
    printf("Enter no. of vertices:");
    scanf("%d", &n);
    printf("\nEnter the adjacency matrix:\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            scanf("%d", &G[i][j]);
        }
    }
    printf("\nEnter the starting node:");
    scanf("%d", &u);
    dijkstra(G, n, u);
    return 0;
}

void dijkstra(int G[MAX][MAX], int n, int start)
{
    int cost[MAX][MAX], distance[MAX], pred;
    int visited[MAX], count, mindistance, m;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (i == j) cost[i][j] = 0;
            else cost[i][j] = INFINITY;
        }
    }
    for (i = 0; i < n; i++)
    {
        cost[i][i] = 0;
        distance[i] = cost[start][i];
    }
    visited[start] = 1;
    count = 1;
    while (count < n)
    {
        mindistance = INFINITY;
        for (m = 0; m < n; m++)
        {
            if (!visited[m] && distance[m] < mindistance)
            {
                start = m;
                mindistance = distance[m];
            }
        }
        visited[start] = 1;
        count++;
        for (j = 0; j < n; j++)
        {
            if (!visited[j] && distance[j] > distance[start] + cost[start][j])
            {
                pred[j] = start;
                distance[j] = distance[start] + cost[start][j];
            }
        }
    }
}
```

Enter no. of vertices:4
Enter the adjacency matrix:
0 1 3 4
2 0 3 3
2 1 0 4
2 2 2 0
Enter the starting node:1
Distance of node0-2
Path0-1
Distance of node2-3
Path2-1
Distance of node3-3
Path3-1
Process returned 0 (0x0) execution time : 82.266 s
Press any key to continue.