

IT300 – Design and Analysis of Algorithms

Lab Assignment – 5

Name: Niraj Nandish

Roll No: 191IT234

1. Program 1 – Johnson's Algorithm
 - a. Code

```
#include <iostream>
#include <limits.h>
#include <math.h>
#include <stdio.h>
#include <vector>

using namespace std;

int minDistance(vector<int> dist, vector<bool> vis)
{
    int min = INT_MAX;
    int min_ind;
    for (int i = 0; i < 4; i++)
    {
        if (vis[i] == false && dist[i] <= min)
        {
            min = dist[i];
            min_ind = i;
        }
    }

    return min_ind;
}

void dijkstra(vector<vector<int>> mod_adj_mat, int u, int v)
{
    vector<bool> vis(4, false);
    vector<int> dist(4, INT_MAX);
    vector<int> path(4);
    dist[u] = 0;

    for (int i = 0; i < 3; i++)
    {
        int min_ind = minDistance(dist, vis);
        vis[min_ind] = true;
        for (int j = 0; j < 4; j++)
        {
            if (!vis[j] && mod_adj_mat[min_ind][j] != -1 && dist[min_ind] + mod_adj_mat[min_ind][j] <
dist[j] && dist[min_ind] != INT_MAX)
```

```

        {
            dist[j] = dist[min_ind] + mod_adj_mat[min_ind][j];
            path[j] = min_ind;
        }
    }
}

if (dist[v] != INT_MAX)
{
    int temp_arr[4];
    int x = 0;
    int a = v;
    temp_arr[0] = v;
    x++;

    do
    {
        temp_arr[x] = path[a];
        a = path[a];
        x++;
    } while (a != u);

    cout << "Shortest path from vertex " << u << " to " << v << " is: " << u;
    for (int z = x - 2; z >= 0; z--)
    {
        cout << " -> ";
        cout << temp_arr[z];
    }
    cout << endl;
}
else
{
    cout << "Shortest path from vertex " << u << " to " << v << " is not possible to be found."
<< endl;
}
}

void dijkstraToAll(vector<vector<int>> mod_adj_mat, int u)
{
    for (int i = 0; i < 4; i++)
    {
        if (i != u)
        {
            dijkstra(mod_adj_mat, u, i);
        }
    }
}

vector<int> bellmanFord(vector<vector<int>> &edge_mat, int adj_mat[][4])
{
    vector<int> dist(5, INT_MAX);
    vector<int> temp;

```

```

dist[4] = 0;

for (int i = 0; i < 4; i++)
{
    temp.push_back(4);
    temp.push_back(i);
    temp.push_back(0);
    edge_mat.push_back(temp);
    temp.clear();
}
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < edge_mat.size(); j++)
    {
        if ((dist[edge_mat[j][0]] != INT_MAX) && (dist[edge_mat[j][0]] + edge_mat[j][2] <
dist[edge_mat[j][1]]))
        {
            dist[edge_mat[j][1]] = dist[edge_mat[j][0]] + edge_mat[j][2];
        }
    }
}

return dist;
}

void johnson(int adj_mat[][4])
{
    vector<vector<int>> edge_mat;
    vector<int> temp(3);
    vector<vector<int>> mod_adj_mat(4, vector<int>(4, 0));
    vector<int> dist;

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (adj_mat[i][j] != 0)
            {
                temp[0] = i;
                temp[1] = j;
                temp[2] = adj_mat[i][j];
                edge_mat.push_back(temp);
            }
        }
    }

    dist = bellmanFord(edge_mat, adj_mat);

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (adj_mat[i][j] != 0)

```

```

    {
        mod_adj_mat[i][j] = adj_mat[i][j] + dist[i] - dist[j];
    }
    else
    {
        mod_adj_mat[i][j] = -1;
    }
}
}

for (int i = 0; i < 4; i++)
{
    cout << "SOURCE VERTEX " << i << endl;
    dijkstraToAll(mod_adj_mat, i);
    cout << endl;
}
}

int main()
{
    int adj_mat[][4] = {{0, -5, 2, 3},
                        {0, 0, 4, 0},
                        {0, 0, 0, 1},
                        {0, 0, 0, 0}};

    johnson(adj_mat);
    return 0;
}

```

b. Screenshots

```
zsh x
niraj ~/Desktop/IT-Labs/DAA-Lab/Lab5 → g++-11 prog1.cpp
niraj ~/Desktop/IT-Labs/DAA-Lab/Lab5 → ./a.out
SOURCE VERTEX 0
Shortest path from vertex 0 to 1 is: 0 -> 1
Shortest path from vertex 0 to 2 is: 0 -> 1 -> 2
Shortest path from vertex 0 to 3 is: 0 -> 1 -> 2 -> 3

SOURCE VERTEX 1
Shortest path from vertex 1 to 0 is not possible to be found.
Shortest path from vertex 1 to 2 is: 1 -> 2
Shortest path from vertex 1 to 3 is: 1 -> 2 -> 3

SOURCE VERTEX 2
Shortest path from vertex 2 to 0 is not possible to be found.
Shortest path from vertex 2 to 1 is not possible to be found.
Shortest path from vertex 2 to 3 is: 2 -> 3

SOURCE VERTEX 3
Shortest path from vertex 3 to 0 is not possible to be found.
Shortest path from vertex 3 to 1 is not possible to be found.
Shortest path from vertex 3 to 2 is not possible to be found.

niraj ~/Desktop/IT-Labs/DAA-Lab/Lab5 →
```

2. Program 2 – Floyd-Warshall Algorithm

a. Code and Screenshots

```
#include <iostream>
#include <limits.h>
#include <math.h>
#include <stdio.h>
#include <vector>

using namespace std;

void floydWarshall(int no_V, vector<vector<int>> &dist, vector<vector<int>> &next)
{
    for (int k = 0; k < no_V; k++)
    {
        if (k == 1 || k == 3 || k == 5)
        {
            cout << "\n\nValues of dist(i,j) at k = " << k << ": " << endl;
            for (int m = 0; m < no_V; m++)
            {
                for (int n = 0; n < no_V; n++)
                {
                    cout << dist[m][n] << " ";
                }
                cout << endl;
            }
        }

        for (int i = 0; i < no_V; i++)
        {
            for (int j = 0; j < no_V; j++)
            {
                if (dist[i][k] == INT_MAX || dist[k][j] == INT_MAX)
                {
                    continue;
                }

                if (dist[i][j] > dist[i][k] + dist[k][j])
                {
                    dist[i][j] = dist[i][k] + dist[k][j];
                    next[i][j] = next[i][k];
                }
            }
        }
    }
}

vector<int> constrPath(int u, int v, vector<vector<int>> next)
{
    vector<int> result;

    if (next[u][v] == -1)
```

```

{
    return {};
}
result.push_back(u);

while (u != v)
{
    u = next[u][v];
    result.push_back(u);
}

return result;
}

int main()
{
    int no_V, no_E, source, dest, wt, n;
    vector<int> path;

    cout << "Enter number of vertices: ";
    cin >> no_V;
    cout << "Enter number of edges: ";
    cin >> no_E;

    vector<vector<int>> adj_mat(no_V, vector<int>(no_V, INT_MAX));
    vector<vector<int>> dist(no_V, vector<int>(no_V));
    vector<vector<int>> next(no_V, vector<int>(no_V));

    for (int i = 1; i <= no_E; i++)
    {
        cout << "\nEdge number " << i;
        cout << "\nEnter source vertex: ";
        cin >> source;
        cout << "Enter destination vertex: ";
        cin >> dest;
        cout << "Enter weight: ";
        cin >> wt;
        adj_mat[source][dest] = wt;
        adj_mat[dest][source] = wt;
    }

    for (int i = 0; i < no_V; i++)
    {
        for (int j = 0; j < no_V; j++)
        {
            if (i == j)
                adj_mat[i][j] = 0;
        }
    }

    cout << "\n\nInitial values of dist(i,j): " << endl;
    for (int i = 0; i < no_V; i++)
    {

```

```

for (int j = 0; j < no_V; j++)
{
    dist[i][j] = adj_mat[i][j];
    cout << dist[i][j] << " ";
    if (adj_mat[i][j] == INT_MAX)
    {
        next[i][j] = -1;
    }
    else
    {
        next[i][j] = j;
    }
}
cout << endl;
}

floydWarshall(no_V, dist, next);

cout << "\n\nFinal value of dist(i,j): " << endl;
for (int m = 0; m < no_V; m++)
{
    for (int n = 0; n < no_V; n++)
    {
        cout << dist[m][n] << " ";
    }
    cout << endl;
}

cout << "\n\nEnter source vertex: ";
cin >> source;
cout << "Enter destination vertex: ";
cin >> dest;
path = constrPath(source, dest, next);
cout << "Shortest path from vertex " << source << " to " << dest << " is: " << path[0];
for (int i = 1; i < path.size(); i++)
{
    cout << " -> " << path[i];
}
cout << endl;

return 0;
}

```



```

zsh
niraj ~/Desktop/IT-Labs/DAA-Lab/Lab5 → g++-11 prog2.cpp
niraj ~/Desktop/IT-Labs/DAA-Lab/Lab5 → ./a.out
Enter number of vertices: 9
Enter number of edges: 16

Edge number 1
Enter source vertex: 0
Enter destination vertex: 1
Enter weight: 5

Edge number 2
Enter source vertex: 1
Enter destination vertex: 3
Enter weight: 8

Edge number 3
Enter source vertex: 3
Enter destination vertex: 6
Enter weight: 2

Edge number 4
Enter source vertex: 6
Enter destination vertex: 7
Enter weight: 3

Edge number 5
Enter source vertex: 7
Enter destination vertex: 8
Enter weight: 5

Edge number 6
Enter source vertex: 8
Enter destination vertex: 5
Enter weight: 4

Edge number 7
Enter source vertex: 5
Enter destination vertex: 2
Enter weight: 5

Edge number 8
Enter source vertex: 2
Enter destination vertex: 0
Enter weight: 2

Edge number 9
Enter source vertex: 0
Enter destination vertex: 3
Enter weight: 4

Edge number 10
Enter source vertex: 0
Enter destination vertex: 4
Enter weight: 10

Edge number 11
Enter source vertex: 2
Enter destination vertex: 4
Enter weight: 7

Edge number 12
Enter source vertex: 4
Enter destination vertex: 5
Enter weight: 3

Edge number 13
Enter source vertex: 3
Enter destination vertex: 4
Enter weight: 6

Edge number 14
Enter source vertex: 4
Enter destination vertex: 6
Enter weight: 2

```

```

Edge number 15
Enter source vertex: 4
Enter destination vertex: 7
Enter weight: 3

Edge number 16
Enter source vertex: 5
Enter destination vertex: 7
Enter weight: 2

Initial values of dist(i,j):
0 5 2 4 10 2147483647 2147483647 2147483647 2147483647
5 0 2147483647 8 2147483647 2147483647 2147483647 2147483647
2 2147483647 0 2147483647 7 5 2147483647 2147483647 2147483647
4 8 2147483647 0 6 2147483647 2 2147483647 2147483647
10 2147483647 7 6 0 3 2 3 2147483647
2147483647 2147483647 5 2147483647 3 0 2147483647 2 4
2147483647 2147483647 2147483647 2 2 2147483647 0 3 2147483647
2147483647 2147483647 2147483647 2147483647 3 2 3 0 5
2147483647 2147483647 2147483647 2147483647 2147483647 4 2147483647 5 0

Values of dist(i,j) at k = 1:
0 5 2 4 10 2147483647 2147483647 2147483647 2147483647
5 0 7 8 15 2147483647 2147483647 2147483647 2147483647
2 7 0 6 7 5 2147483647 2147483647 2147483647
4 8 6 0 6 2147483647 2 2147483647 2147483647
10 15 7 6 0 3 2 3 2147483647
2147483647 2147483647 5 2147483647 3 0 2147483647 2 4
2147483647 2147483647 2147483647 2 2 2147483647 0 3 2147483647
2147483647 2147483647 2147483647 2147483647 3 2 3 0 5
2147483647 2147483647 2147483647 2147483647 2147483647 4 2147483647 5 0

Values of dist(i,j) at k = 3:
0 5 2 4 9 7 2147483647 2147483647 2147483647
5 0 7 8 14 12 2147483647 2147483647 2147483647
2 7 0 6 7 5 2147483647 2147483647 2147483647
4 8 6 0 6 11 2 2147483647 2147483647
9 14 7 6 0 3 2 3 2147483647
7 12 5 11 3 0 2147483647 2 4
2147483647 2147483647 2147483647 2 2 2147483647 0 3 2147483647
2147483647 2147483647 2147483647 2147483647 3 2 3 0 5
2147483647 2147483647 2147483647 2147483647 2147483647 4 2147483647 5 0

Values of dist(i,j) at k = 5:
0 5 2 4 9 7 6 12 2147483647
5 0 7 8 14 12 10 17 2147483647
2 7 0 6 7 5 8 10 2147483647
4 8 6 0 6 9 2 9 2147483647
9 14 7 6 0 3 2 3 2147483647
7 12 5 9 3 0 5 2 4
6 10 8 2 2 5 0 3 2147483647
12 17 10 9 3 2 3 0 5
2147483647 2147483647 2147483647 2147483647 2147483647 4 2147483647 5 0

Final value of dist(i,j):
0 5 2 4 8 7 6 9 11
5 0 7 8 12 12 10 13 16
2 7 0 6 7 5 8 7 9
4 8 6 0 4 7 2 5 10
8 12 7 4 0 3 2 3 7
7 12 5 7 3 0 5 2 4
6 10 8 2 2 5 0 3 8
9 13 7 5 3 2 3 0 5
11 16 9 10 7 4 8 5 0

Enter source vertex: 0
Enter destination vertex: 7
Shortest path from vertex 0 to 7 is: 0 -> 2 -> 5 -> 7
niraj ~/Desktop/IT-Labs/DAA-Lab/Lab5 →

```

b. How can the output of the Floyd-Warshall algorithm be used to detect the presence of a negative-weight cycle? Explain your answer.

Ans. The output of Floyd-Warshall algorithm can be used to check if a negative-weight cycle is present or not. All we have to do is check the diagonal entries of $\text{dist}(i,j)$ matrix. If the value of the diagonal entries is negative, then there exists a negative weight cycle. This means that there is a path from the vertex to itself, hence it has negative weight.