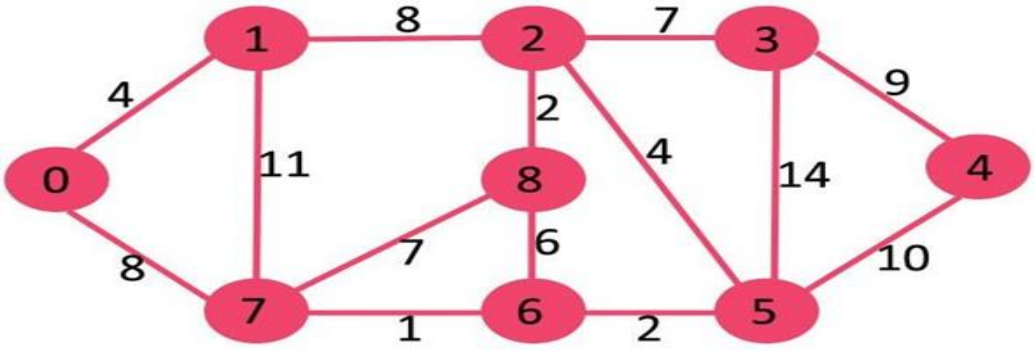
# Experiment-5

**Student Name:** Toshiba Ansari
**UID:** 20BCS6671
**Branch:** CSE_AIML
**Section/Group:** AIML_4_B
**Semester:** 5th
**Date of Performance:** 12/10/2022
**Subject Name:** Adv. Programming Lab
**Subject Code:** 20CSP-334

## 1. Aim/Overview of the practical:

**(A)** From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.




**(B)** Compute the transitive closure of a given directed graph using Warshall's algorithm.

## 2. Task to be done:

**(A)** To find the shortest paths to other vertices using Dijkstra's algorithm.

**(B)** To find the transitive closure of a directed graph.

## 3. Algorithm/ Flowchart:

**(A)** 1) Create a set sptSet (shortest path tree set) that keeps track of vertices included in the shortest-path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.

2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

3) While sptSet doesn't include all vertices

a) Pick a vertex u which is not there in sptSet and has a minimum distance value.

b) Include u to sptSet.

c) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if the sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

**(B)** Warshall(A[1...n, 1...n]) // A is the adjacency matrix
$\mathbf{R}^{(0)} \leftarrow A$
**for** k ← 1 **to** n **do**
**for** i ← 1 **to** n **do**
**for** j ← **to** n **do**
$R^{(k)}[i, j] \leftarrow R^{(k-1)}[i, j]$ **or** ($R^{(k-1)}[i, k]$ **and** $R^{(k-1)}[k, j]$)
**return** $R^{(n)}$

## 4. Code:

**(A) Dijkstra Algorithm**

```
#include <iostream>

using namespace std;

#include <limits.h>
```

```
   // Number of vertices in the graph
    #define V 9
   // A utility function to find the vertex with minimum distance value, from
   // the set of vertices not yet included in shortest path tree
   int minDistance(int dist[], bool sptSet[])
{
   // Initialize min value
   int min = INT_MAX, min_index;
   for (int v = 0; v < V; v++)
      if (sptSet[v] == false && dist[v] <= min)
         min = dist[v], min_index = v;
   return min_index;
}
// A utility function to print the constructed distance array
void printSolution(int dist[])
{
   cout <<"Vertex \t Distance from Source" << endl;
   for (int i = 0; i < V; i++)
      cout  << i << " \t\t"<<dist[i]<< endl;
}
// Function that implements Dijkstra's single source shortest path algorithm
```

```
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
   int dist[V]; // The output array.  dist[i] will hold the shortest
   // distance from src to i
   bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
   // path tree or shortest distance from src to i is finalized
   // Initialize all distances as INFINITE and stpSet[] as false
   for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = false;
   // Distance of source vertex from itself is always 0
   dist[src] = 0;
   // Find shortest path for all vertices
   for (int count = 0; count < V - 1; count++) {
      // Pick the minimum distance vertex from the set of vertices not
      // yet processed. u is always equal to src in the first iteration.
      int u = minDistance(dist, sptSet);
      // Mark the picked vertex as processed
      sptSet[u] = true;
      // Update dist value of the adjacent vertices of the picked vertex.
      for (int v = 0; v < V; v++)
```

```
// Update dist[v] only if is not in sptSet, there is an edge from u to v, and total
weight of path from src to  v through u is smaller than current value of dist[v]

        if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX

           && dist[u] + graph[u][v] < dist[v])

           dist[v] = dist[u] + graph[u][v];

   }

   // print the constructed distance array

   printSolution(dist);

}

// driver program to test above function

int main()

{

   /* Let us create the example graph discussed above */

   int graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},

                      {4, 0, 8, 0, 0, 0, 0, 11, 0},

                      {0, 8, 0, 7, 0, 4, 0, 0, 2},

                      {0, 0, 7, 0, 9, 14, 0, 0, 0},

                      {0, 0, 0, 9, 0, 10, 0, 0, 0},

                      {0, 0, 4, 14, 10, 0, 2, 0, 0},

                      {0, 0, 0, 0, 0, 2, 0, 1, 6},

                      {8, 11, 0, 0, 0, 0, 1, 0, 7},

                      {0, 0, 2, 0, 0, 0, 6, 7, 0}};
```

```
    dijkstra(graph, 0);
    return 0;
}
```

**(B) Warshall Algorithm**

```
#include<iostream>
using namespace std;
const int n_nodes = 20;
int main() {
  int n_nodes, k, n;
  char i, j, res, c;
  int adj[10][10], path[10][10];
  cout << "\n\tMaximum number of nodes in the graph :";
  cin >>n;
  n_nodes = n;
  cout << "\nEnter 'y' for 'YES' and 'n' for 'NO' \n";
  for (i = 97; i < 97 + n_nodes; i++)
    for (j = 97; j < 97 + n_nodes; j++) {
      cout << "\n\tIs there an edge from " << i << " to " << j << " ? ";
      cin >>res;
      if (res == 'y')
```

```
            adj[i - 97][j - 97] = 1;
         else
            adj[i - 97][j - 97] = 0;
      }
      cout << "\nTransitive Closure of the Graph:\n";
      cout << "\n\t\t\t ";
      for (i = 0; i < n_nodes; i++) {
        c = 97 + i;
        cout << c << " ";
      }
      cout << "\n\n";
      for (int i = 0; i < n_nodes; i++) {
        c = 97 + i;
        cout << "\t\t\t" << c << " ";
        for (int j = 0; j < n_nodes; j++)
           cout << adj[i][j] << " ";
           cout << "\n";
      }
      return 0;
   }
```

## 5. Output:

```
Vertex   Distance from Source
0                0
1                4
2                12
3                19
4                21
5                11
6                9
7                8
8                14


...Program finished with exit code 0
Press ENTER to exit console.
```

**Output: Dijkstra**

```
        Maximum number of nodes in the graph :3

Enter 'y' for 'YES' and 'n' for 'NO'

        Is there an edge from a to a ? y

        Is there an edge from a to b ? n

        Is there an edge from a to c ? y

        Is there an edge from b to a ? y

        Is there an edge from b to b ? y

        Is there an edge from b to c ? n

        Is there an edge from c to a ? n

        Is there an edge from c to b ? n

        Is there an edge from c to c ? y

Transitive Closure of the Graph:

                        a b c

                    a 1 0 1
                    b 1 1 0
                    c 0 0 1
```

**Output: Warshall**

**Learning outcomes (What I have learnt):**

- Learnt about Dijkstra algorithm
- Learnt about Warshall algorithm
- Learnt about transitive closure
- Learnt how to code both of them

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
|---|---|---|---|
| 1. | | | |
| 2. | | | |
| 3. | | | |
| | | | |