Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai) Munshi Nagar, Andheri (W), Mumbai – 400 058.

"

Name: Prakhar

UID: 202130040

Subject: DAA

Experiment No: 6

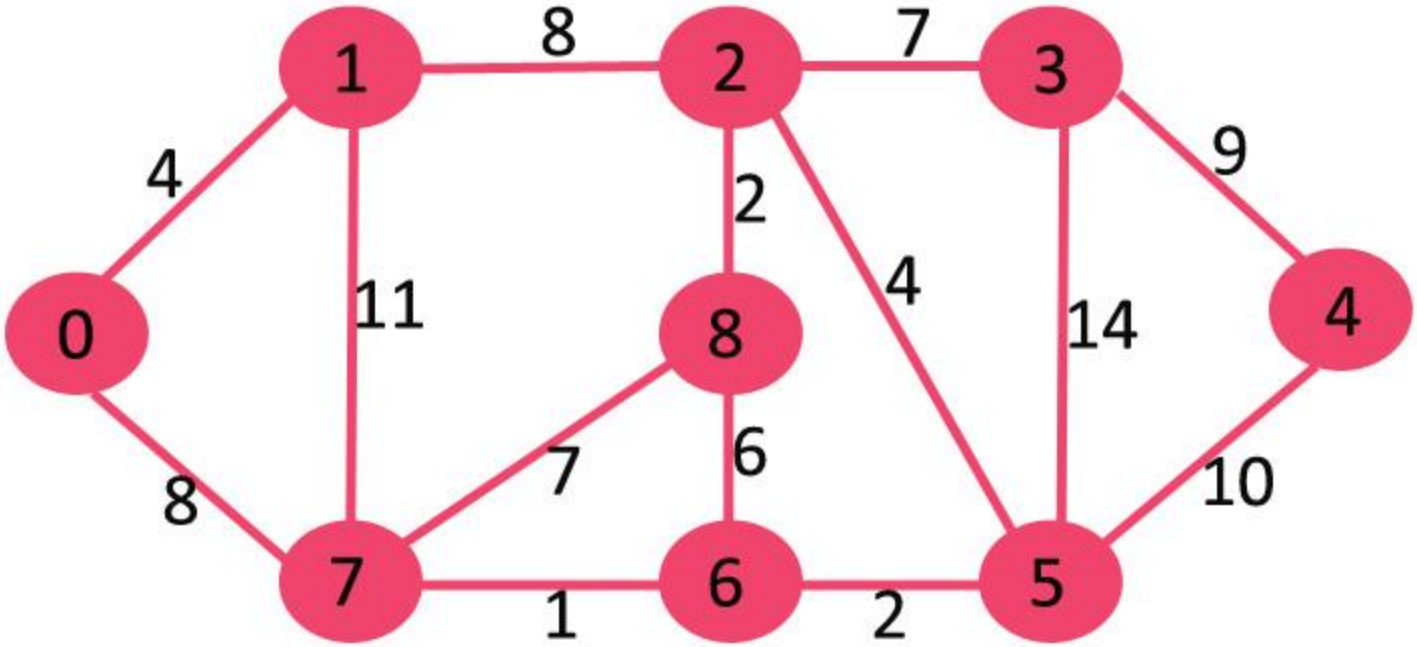
Aim – Experiment Dijkstra's algorithm

Objective: Apply the concept of greedy to get the shortest path to every node of the graph.

Theory:

To understand the Dijkstra's Algorithm lets take a graph and find the shortest path from source to all nodes.

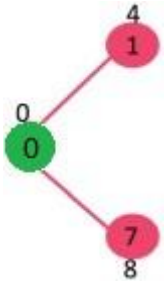
*Consider below graph and **src = 0***



Step 1:

- *The set **sptSet** is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where **INF** indicates infinite.*
- *Now pick the vertex with a minimum distance value. The vertex 0 is picked, include it in sptSet. So sptSet becomes {0}. After including 0 to sptSet, update distance values of its adjacent vertices.*
- *Adjacent vertices of 0 are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8.*

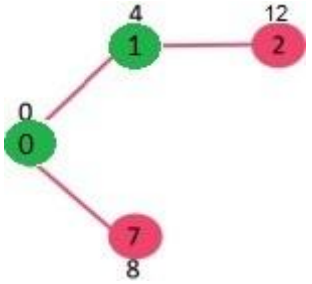
*The following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in **green** colour.*



Step 2:

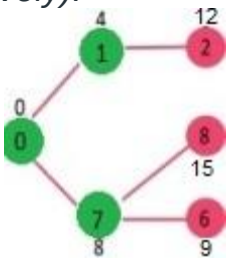
- *Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 is picked and added to sptSet.*

- *So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1.*
- *The distance value of vertex 2 becomes **12**.*



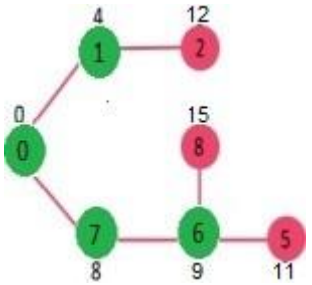
Step 3:

- *Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes **{0, 1, 7}.***
- *Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (**15 and 9** respectively).*

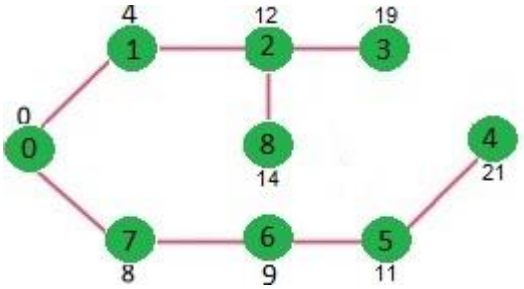


Step 4:

- *Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 6 is picked. So sptSet now becomes **{0, 1, 7, 6}**.*
- *Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.*



*We repeat the above steps until sptSet **includes** all vertices of the given graph. Finally, we get the following Shortest Path Tree (SPT).*



Algorithm:

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

- While **sptSet** doesn't include all vertices
    - Pick a vertex **u** that is not there in **sptSet** and has a minimum distance value.
    - Include u to **sptSet**.
    - Then update the 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 the 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.

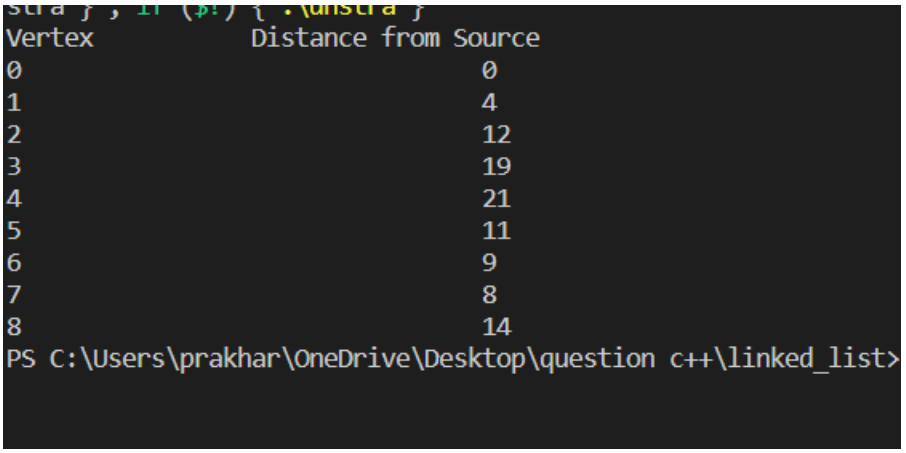
Program:

```c
// C program for Dijkstra's single source shortest path
// algorithm. The program is for adjacency matrix
// representation of the graph

#include <limits.h>
#include <stdbool.h>
#include <stdio.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[])
{
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t\t\t %d\n", i, dist[i]);
}

// 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's code
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 } };

    // Function call
    dijkstra(graph, 0);

    return 0;
```

```
}
```

Result:



Observation:

The greedy algorithm can easily solve Dijkstra's Algorithms in o(ElogV)

CONCLUSION:

After conducting this experiment, I have learnt how to use greedy approach to solve Dijkstra algorithm.