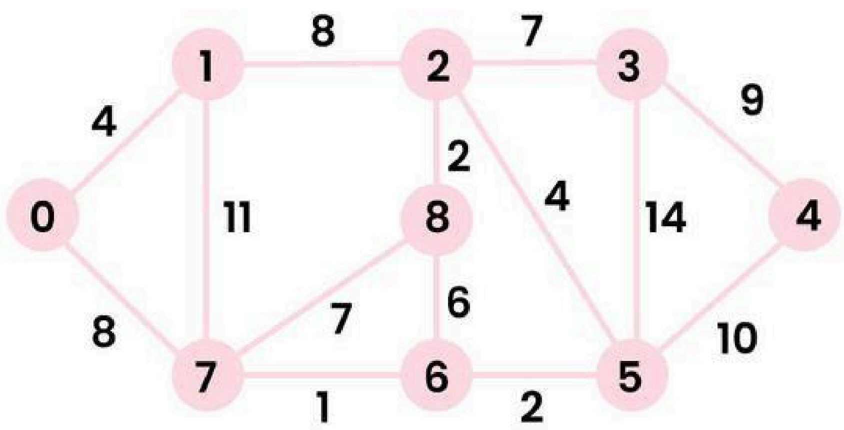
AOA EXPERIMENT 4

Name: Mohammed Arhaan Ali
Div: S11
Roll No: 02

Aim: To implement Dijkstra's Algorithm using Greedy approach.

Theory:
Given a weighted graph and a source vertex in the graph, find the shortest paths from the source to all the other vertices in the given graph.
Note: The given graph does not contain any negative edge.
Examples:
Input: src = 0, the graph is shown below.



Output: 0 4 12 19 21 11 9 8 14
Explanation: The distance from 0 to 1 = 4.
The minimum distance from 0 to 2 = 12. 0->1->2
The minimum distance from 0 to 3 = 19. 0->1->2->3
The minimum distance from 0 to 4 = 21. 0->7->6->5->4
The minimum distance from 0 to 5 = 11. 0->7->6->5
The minimum distance from 0 to 6 = 9. 0->7->6
The minimum distance from 0 to 7 = 8. 0->7
The minimum distance from 0 to 8 = 14. 0->1->2->8

Dijkstra's Algorithm using Adjacency Matrix:

The idea is to generate a SPT (shortest path tree) with a given source as a root. Maintain an Adjacency Matrix with two sets,

- one set contains vertices included in the shortest-path tree,
- other set includes vertices not yet included in the shortest-path tree.

At every step of the algorithm, find a vertex that is in the other set (set not yet included) and has a minimum distance from the source.
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.

Note: We use a boolean array sptSet[] to represent the set of vertices included in SPT. If a value sptSet[v] is true, then vertex v is included in SPT, otherwise not. Array dist[] is used to store the shortest distance values of all vertices.

Code:

```
#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;
}
```

Output:

```
PS C:\Users\arhaa\OneDrive\Desktop\AOA> cd "c:\Users\arhaa\OneDriv
ra.c -o dijkstra } ; if ($?) { .\dijkstra }
Vertex           Distance from Source
0                                0
1                                4
2                                12
3                                19
4                                21
5                                11
6                                9
7                                8
8                                14
PS C:\Users\arhaa\OneDrive\Desktop\AOA>
```

Conclusion: Thus we have successfully implemented Dijkstra's algorithm using Greedy approach.