Vashisth Zatakia
60003200163 B3

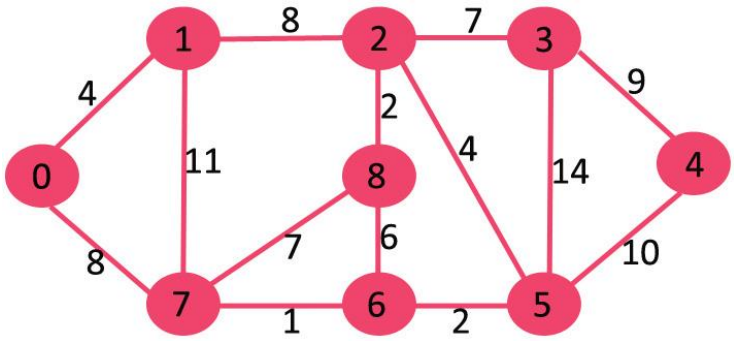
Aim: Implementation of shortest path with Dijkstra's Algorithm.

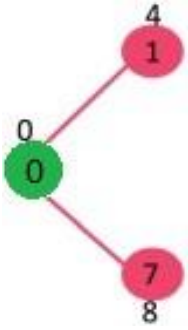
Theory:

Given a graph and a source vertex in the graph, find the shortest paths from the source to all vertices in the given graph.
Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a *SPT (shortest path tree)* with a given source as a root. We maintain 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, we find a vertex that is in the other set (set of not yet included) and has a minimum distance from the source.
Below are the detailed steps used in Dijkstra's algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.

Algorithm
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.

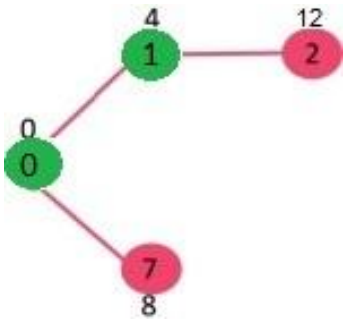
Let us understand with the following example:



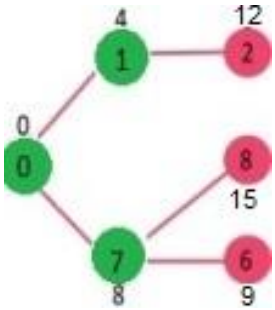


The set *sptSet* is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with a minimum distance value. The vertex 0 is picked, include it in *sptSet*. So *sptSet* becomes {0}. After including 0 to *sptSet*, update 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,

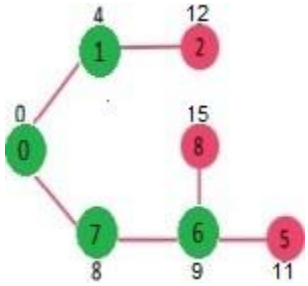
only the vertices with finite distance values are shown. The vertices included in SPT are shown in green colour.



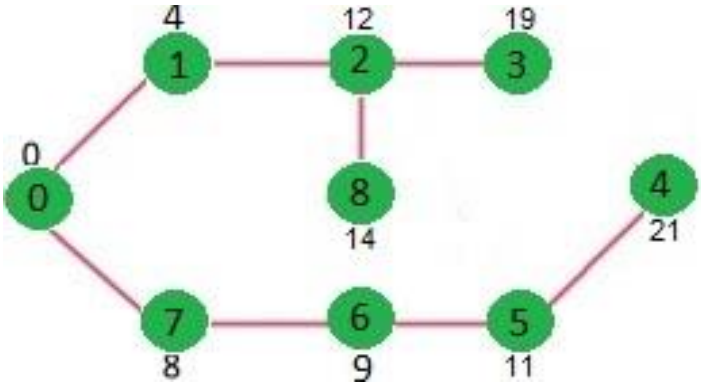
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 is picked and added to sptSet. So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.



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).



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).



Time Complexity:

The given graph G=(V, E) is represented as an adjacency matrix. Here w[u, v] stores the weight of edge (u, v). The priority queue Q is represented as an unordered list. Let |E| and |V| be the number of edges and vertices in the graph, respectively. Then the time complexity is calculated:

Adding all |V| vertices to Q takes O(|V|) time.

Vashisth Zatakia
60003200163 B3

Removing the node with minimal dist takes O(|V|) time, and we only need O(1) to recalculate dist[u] and update Q. Since we use an adjacency matrix here, we'll need to loop for |V| vertices to update the dist array.

The time taken for each iteration of the loop is O(|V|), as one vertex is deleted from Q per loop.

Thus, total time complexity becomes O(|V|) + O(|V|) \times O(|V|) = O(|V|^2).

Source Code:

```
#include <stdio.h>
#include<stdbool.h>
#define MAX 999
int v;

int mindistance(int distance[],bool vertset[])
{
   int minimum=MAX;
   int index;
   for(int i=0;i<v;i++)
   {
      if(vertset[i]==false && distance[i]<=minimum)
      {
         minimum=distance[i];
         index=i;
      }
   }
   return index;
}
void djkastra(int graph[v][v],int source)
{
   int distance[v];
   bool vertset[v];
```

```
    for(int i=0; i<v;i++)
    {
        distance[i]=MAX;
        vertset[i]=false;
    }
    distance[source]=0;
    for(int i=0;i<v-1;i++)
    {
        int u=mindistance(distance,vertset);
        vertset[u]=true;
        for(int j=0;j<v;j++)
        {
            if(!vertset[v] && graph[u][j]!=0 && distance[u]!=MAX && distance[u]+graph[u][j]<distance[j])
            {
                distance[j]=distance[u]+graph[u][j];
            }
        }
    }
    printf("Vertex list \t Distance");
    for(int i;i<v;i++)
    {
        printf("\n%d \t\t %d",i,distance[i]);
    }
}
int main()
{
    printf("Enter number of vertices:");
    scanf("%d",&v);
    int graph[v][v];
```

```
    for(int i=0; i<v;i++)
    {
        printf("Enter edges for vertex %d: ",i);
        for(int j=0;j<v;j++)
        {
            scanf("%d",&graph[i][j]);
        }
    }
    djkastra(graph,0);
    return 0;
}
```

Output:

```
Enter number of vertices:9
Enter edges for vertex 0: 0 4 0 0 0 0 0 8 0
Enter edges for vertex 1:  4 0 8 0 0 0 0 11 0
Enter edges for vertex 2:  0 8 0 7 0 4 0 0 2
Enter edges for vertex 3: 0 0 7 0 9 14 0 0 0
Enter edges for vertex 4: 0 0 0 9 0 10 0 0 0
Enter edges for vertex 5:  0 0 4 14 10 0 2 0 0
Enter edges for vertex 6: 0 0 0 0 0 2 0 1 6
Enter edges for vertex 7: 8 11 0 0 0 0 1 0 7
Enter edges for vertex 8: 0 0 2 0 0 0 6 7 0
Vertex list      Distance
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.
```

Conclusion: Thus, we have implemented shortest path problem using DjKastra's Algorithm and the time complexity of the program is O(|V|^2) where V represents number of vertices.