NAME:	SANIYA BANGARE
UID NO:	2021300009
BATCH	A
EXP NO:	6th

AIM:	(Graph Algorithm - Single source shortest path algorithm. Dijkstra Algorithms
THEORY:	The Single-Source Shortest Path (SSSP) problem consists of finding the shortest paths between a given 1. vertex v and all other vertices in the graph 2. Bellman Ford algorithm works by overestimating the length of the path from the starting vertex to all other vertices. 3. Then it iteratively relaxes those estimates by finding new paths that are shorter than the previously overestimated paths. 4. Dijkstra algorithm is a single-source shortest path algorithm. Here, single-source means that only one source is given, and we have to find the shortest path from the source to all the nodes. **5. ALGORITHM :** **1. Bellman–Ford :** 1. function bellmanFordAlgorithm(G, s) //G is the graph and s is the source vertex 2. for each vertex V in G 3. dist[V] <- infinite // dist is distance 4. prev[V] <- NULL // prev is previous 5. dist[s] <- 0 6. for each vertex V in G 7. for each edge (u,v) in G 8. temporaryDist <- dist[u] + edgeweight(u, v) 9. if temporaryDist < dist[v] 10. dist[v] <- temporaryDist 11. prev[v] <- u 12. for each edge (U,V) in G 13. If dist[U] + edgeweight(U, V) < dist[V} 14. Error: Negative Cycle Exists 15. return dist[], previ[]

2. Dijkstra Algorithms :

1. Mark the source node with a current distance of 0 and the rest with infinity.
2. Set the non-visited node with the smallest current distance as the current node.
3. For each neighbor, N of the current node adds the current distance of the adjacent node with the weight of the edge connecting 0->1. If it is smaller than the current distance of Node, set it as the new current distance of N.
4. Mark the current node 1 as visited.
5. Go to step 2 if there are any nodes are unvisited.

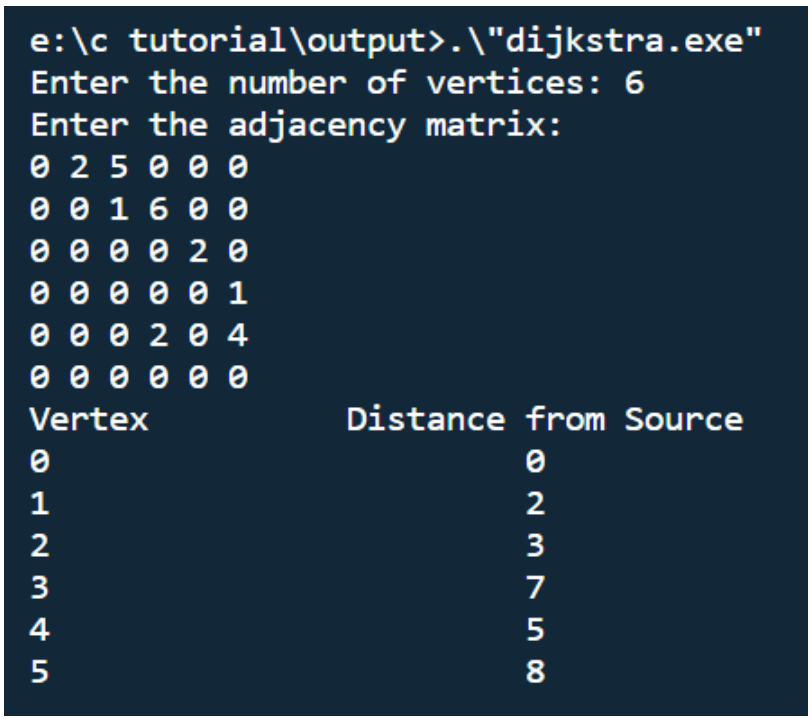
7. Time Complexity :

1. Bellman–Ford
1. O(V * E)
2. Dijkstra Algorithms
1. O((V+E)LogV)

PROGRAM:

```c
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#include<stdbool.h>

int minDistance(int dist[], bool minSet[], int V)
{
   int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++)
     if (minSet[v] == false && dist[v] <= min)
         min = dist[v], min_index = v;

   return min_index;
}

void printSolution(int dist[], int V)
{
   printf("Vertex\t\tDistance from Source \n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t\t %d\n", i, dist[i]);
}

void dijkstra( int v, int graph[v][v], int source){
```

```
    int dist[v];
    bool minSet[v];
    for(int i=0;i<v;i++){
        dist[i]=INT_MAX;
        minSet[i]=false;
    }
    dist[source]=0;
    for(int count=0;count<v-1;count++){
        int u = minDistance(dist,minSet,v);
        minSet[u]=true;
        for(int i=0;i<v;i++){
            if(!minSet[i] && graph[u][i] && dist[u]!=INT_MAX &&
dist[u]+graph[u][i]<dist[i]){
                dist[i]=dist[u]+graph[u][i];
            }
        }
    }
    printSolution(dist,v);
}

int main(){
    int v;
    printf("Enter the number of vertices: ");
    scanf("%d",&v);
    int graph[v][v];
    printf("Enter the adjacency matrix: ");
    for(int i=0;i<v;i++){
        for(int j=0;j<v;j++){
            scanf("%d",&graph[i][j]);
        }
    }
    dijkstra(v,graph,0);
    return 0;

}
```

RESULT:	
CONCLUSION:	From this experiment, i have successfully understood how to perform - Single source shortest path algorithm.