Walchand College of Engineering, Sangli
Computer Science & Engineering
Third Year

Course: Design and analysis of algorithm Lab

Lab course coordinator:
Mrs A M Chimanna- Batch: - T1, T2, T3,T4

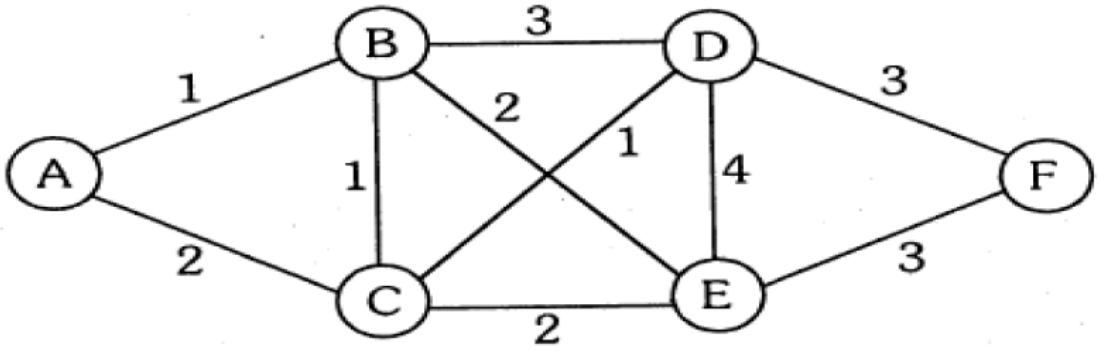
Assignment No 8

Greedy Method

Name: Siddharth Shrinivas Salunkhe.

PRN:21510111.

1) From a given vertex in a weighted connected graph, implement the shortest path finding Dijkstra's algorithm.



Solution:

```
#include<bits/stdc++.h>
using namespace std;
#define V 6
char minDistance(int dist[], bool sptSet[])
{
    int min = INT_MAX;
    char min_index;
    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            {
                min = dist[v];
                min_index = 'A'+v;
            }
    return min_index;
}
void printSolution(int dist[]){
    cout<<"Vertex \t\t Distance from Source"<<endl;
    for (int i = 0; i < V; i++){
        char x=i+'A';
        cout<<x<<"\t\t\t\t"<<dist[i]<<endl;
    }
}
void dijkstra(vector<pair<int,pair<char,char>>> graph, char
src,map<char,vector<pair<char,int>>> graphMap)
{
    int dist[V];
    bool sptSet[V];
    for (int i = 0; i < V; i++)
```

```
        {
            dist[i] = INT_MAX;
            sptSet[i] = false;
        }
    dist[src-'A'] = 0;
    for (int count = 0; count < V - 1; count++) {
        char u = minDistance(dist, sptSet);
        sptSet[u-'A'] = true;
        for (auto x:graphMap[u])
            if (!sptSet[x.first-'A'] && dist[u-'A'] != INT_MAX&&
dist[u-'A'] + x.second < dist[x.first-'A'])
            dist[x.first-'A'] = dist[u-'A'] + x.second;
    }
    printSolution(dist);
}
int main(){
    vector<pair<int,pair<char,char>>>graph;
    map<char,vector<pair<char,int> > > graphMap;
    int numNodes=6,numEdges=10;
    char x;
    graph={
        {1,{'A','B'}},
        {2,{'A','C'}},
        {1,{'B','C'}},
        {3,{'B','D'}},
        {2,{'B','E'}},
        {1,{'C','D'}},
        {2,{'C','E'}},
        {3,{'D','F'}},
        {4,{'D','E'}},
        {3,{'E','F'}}
    };
    for(auto x:graph)
    {
        graphMap[x.second.first].push_back({x.second.second,x.first});
        graphMap[x.second.second].push_back({x.second.first,x.first});
    }
    cout<<"Enter source Node:";
    cin>>x;
    dijkstra(graph, x,graphMap);

    return 0;
}
```

Q) Show that Dijkstra’s algorithm doesn’t work for graphs with negative weight edges

Ans: The problem is that Dijkstra's algorithm relies on the assumption that it always selects the vertex with the smallest tentative distance as the next vertex to explore. However, when there are negative-weight edges, this assumption can break down because it might revisit vertices and potentially find shorter paths after initially selecting a path that appears shorter. This is known as the "relaxation" step in Dijkstra's algorithm.