Dijkstra's Algorithm

Dijkstra's Algorithm can be used to find the shortest path from the single given source to the given destination(s).

The time complexity of this algorithm is O(V^2).

```
//Header Section

#include<iostream>
#include<stdio.h>
using namespace std;
#define INFINITY 9999
#define max 5

//Function Declaration

void dijkstra(int G[max][max],int n,int startnode);

//Main Function

int main()
{
  int G[max][max]={{0,1,0,3,10},{1,0,5,0,0},{0,5,0,2,1},{3,0,2,0,6},{10,0,1,6,0}};
  int n=5;
  int u=0;
  dijkstra(G,n,u);
  return 0;
}

//Function Definition

void dijkstra(int G[max][max],int n,int startnode)
{
  int cost[max][max],distance[max],pred[max];
  int visited[max],count,mindistance,nextnode,i,j;
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
  if(G[i][j]==0)
    cost[i][j]=INFINITY;
  else
    cost[i][j]=G[i][j];
  for(i=0;i<n;i++) {
```

```
    distance[i]=cost[startnode][i];
    pred[i]=startnode;
    visited[i]=0;
  }
  distance[startnode]=0;
  visited[startnode]=1;
  count=1;
  while(count<n-1) {
    mindistance=INFINITY;
    for(i=0;i<n;i++)
      if(distance[i]<mindistance&&!visited[i]) {
      mindistance=distance[i];
      nextnode=i;
    }
    visited[nextnode]=1;
    for(i=0;i<n;i++)
      if(!visited[i])
    if(mindistance+cost[nextnode][i]<distance[i]) {
      distance[i]=mindistance+cost[nextnode][i];
      pred[i]=nextnode;
    }
    count++;
  }
  for(i=0;i<n;i++)
  if(i!=startnode) {
    cout<<"\nDistance of node"<<i<<"="<<distance[i];
    cout<<"\nPath="<<i;
    j=i;
    do {
      j=pred[j];
      cout<<"<-"<<j;
    }while(j!=startnode);
  }
}
```