My Peer-graded Assignment 2

- Aim :- **implement Dijkstra's Algorithm in C++**

Dijkstra’s Algorithm allows you to calculate the shortest path between one node and every other node in a graph.

- **Algorithm Execution**

Here’s how the algorithm is implemented:

1. Mark all nodes as *unvisited*.
2. Mark the initially selected node with the *current* distance of 00 and the rest with *infinity*.
3. Set the *initial node* as the *current node*.
4. For the current node, consider all of its unvisited neighbors and calculate their distances by adding the *current* distance of the current node to the *weight* of the *edge* that connects the current node to the neighboring node.
5. Compare the newly calculated distance to the current distance assigned to the neighboring node. If it is smaller, set it as the new current distance of the *neighboring node* otherwise, keep the previous weight.
6. When you’re done considering all of the unvisited neighbors of the current node, mark the current node as visited.
7. Select the unvisited node that is marked with the smallest distance, set it as the new *current node*, and go back to step 4.

Now repeat this process, until all the nodes are marked as visited.

Demo

Let's take a look at an illustration implementing Dijkstra's Algorithm to understand it better!

Implementation

In the code below, an adjacency matrix is used for an **undirected** graph.

A 6x6 matrix is used for the above case, but you can change it per your need.

The vertex with the minimum distance, which is not included in the **Tset**, is searched in the `minimumDist()` method.

Remember: In C++, **INT_MAX** is a default large number as a replacement for infinity in the above algorithm.

```
#include<iostream>
#include<climits>
using namespace std;

// this method returns a minimum distance for the
// vertex which is not included in Tset.
int minimumDist(int dist[], bool Tset[])
{
    int min=INT_MAX,index;

    for(int i=0;i<6;i++)
    {
        if(Tset[i]==false && dist[i]<=min)
        {
            min=dist[i];
            index=i;
        }
    }
    return index;
}

void Dijkstra(int graph[6][6],int src) // adjacency matrix used is 6x6
```

```
{
    int dist[6]; // integer array to calculate minimum distance for each node.

    bool Tset[6];// boolean array to mark visted/unvisted for each node.

    // set the nodes with infinity distance
    // except for the initial node and mark
    // them unvisited.
    for(int i = 0; i<6; i++)
    {
        dist[i] = INT_MAX;
        Tset[i] = false;
    }

    dist[src] = 0;   // Source vertex distance is set to zero.

    for(int i = 0; i<6; i++)
    {
        int m=minimumDist(dist,Tset); // vertex not yet included.
        Tset[m]=true;// m with minimum distance included in Tset.
        for(int i = 0; i<6; i++)
        {
            // Updating the minimum distance for the particular node.
            if(!Tset[i] && graph[m][i] && dist[m]!=INT_MAX && dist[m]+graph[m]
[i]<dist[i])
                dist[i]=dist[m]+graph[m][i];
        }
    }
    cout<<"Vertex\t\tDistance from source"<<endl;
    for(int i = 0; i<6; i++)
    { //Printing
char str=65+i; // Ascii values for pritning A,B,C..
        cout<<str<<"\t\t\t"<<dist[i]<<endl;
    }
}
int main()
{
    int graph[6][6]={
        {0, 10, 20, 0, 0, 0},
        {10, 0, 0, 50, 10, 0},
        {20, 0, 0, 20, 33, 0},
        {0, 50, 20, 0, 20, 2},
        {0, 10, 33, 20, 0, 1},
        {0, 0, 0, 2, 1, 0}};
    Dijkstra(graph,0);
    return 0;
}
```

Output
0.433s

```
Vertex              Distance from source
A                   0
B                   10
C                   20
D                   23
E                   20
F                   21
```