Name of the Algorithm : **Djikstra’s Algorithm**

**Description** : This is a single source shortest path algorithm . This algorithm helps in finding the shortest distance from one node to other nodes present in the graph. Generally we use it for finding the shortest distance of one node from all other node rather than finding the shortest distance between any two nodes . It is an extended application of the djikstra’s algorithm.

This is one of the most reliable and fastest algorithm which helps in finding the shortest path.

We also use this algorithm for finding the path which we followed to obtain the lowest cost.

I have Implemented this Algorithm on 3 different platforms

**PSEUDOCODE:**

**WORKING OF CODE :**

here initially I will get the number of the vertices and edges as input. Then after getting the input of all the edges along with their respective weight , i’ll call the djikstra function.

*Now in DJIKSTRA’s function :*

1)- I have one array named VIS , which if set true means the vertex with current index has been visited and vice-versa .

2)-Then I have an array named DIST , which will give the shortest path of a node From the chosen node at the index equals to the respective node’s number.

3)-Then I have used minimum priority queue which will pop-out the neighbour edge having the minimum weight.

4)-Initially we will push the vertex in the queue from which we want the shortest distance of all the nodes.

5)-Then on every iteration we will push the unvisited neighbours of the node on the top of the priority queue with side by side updation of distances stored in the DIST.

6)- Once the queue is empty we will stop the loop and will print the DIST array with the node number as the respective index.

EXAMPLE GRAPH :

1. **XCODE (using CPP)**

CODE :

#include<bits/stdc++.h>

#include <fstream>

#define all(x) x.begin(),x.end()

#define rall(x) x.rbegin(),x.rend()

#define FILL(a,b) memset((a),(b),sizeof((a)))

#define countr(v,a) (int)count(v.begin(),v.end(),a)

#define err(v) v.erase(v.begin(),v.end());

#define fast ios\_base::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);

#define ll long long

#define long\_vec vector<ll>

#define nl cout<<endl;

#define out cout<<

#define print(v) repl(0,v.size()){out v[i]<<" ";}

#define rep(i,a,n) for(int i=a;i<n;i++)

#define repl(a,b) for(ll i=a;i<b;i++)

#define ret0 return 0;

#define sortv(v) sort(v.begin(),v.end())

#define start int main(){fast str s;int inp;ll n,inpl,a,b,t,q=0,k;long\_vec v;char c;ifstream in ("/Users/kashishmiglani/Desktop/iCloud Drive (Archive) - 1/Desktop/practice/inputfile") ;//ofstream Output\_in\_file("/Users/kashishmiglani/Desktop/op1.txt");

#define str string

#define pb push\_back

#define pll pair<ll,ll>

#define vec vector<int>

#define mp(a,b) make\_pair(a,b)

#define vecp vector<pair<ll,ll>>

#define fi(it,a) for(auto it=a.begin();it!=a.end();it++)

#define MOD 1000000007

#define MAX 100000

using namespace std;

vector<pair<ll,ll>> vv[100000];

void djikstra(int n)

{

bool vis[n+1];

ll dist[n+1];

rep(i, 0,n+1)

{

vis[i]=false;dist[i]=INT\_MAX;

}

priority\_queue<pll,vector<pll>,greater<pll>> q;

q.push(mp(0,1));

dist[1]=0;

while(!q.empty())

{

ll w1=q.top().first;

ll e1=q.top().second;

q.pop();

if(vis[e1])

continue;

vis[e1]=true;

rep(i, 0,vv[e1].size())

{

if(dist[vv[e1][i].second] > dist[e1] + vv[e1][i].first )

{

dist[vv[e1][i].second] =dist[e1] + vv[e1][i].first;

if(!vis[vv[e1][i].second])

q.push(vv[e1][i]);

}

}

}

out "vertex distance";nl

rep(i, 1,n+1)

{

out i<<" "<<dist[i];nl

}

}

start

ll m;

in>>m>>n;

rep(i,0, n)

{

in>>a>>b>>k;

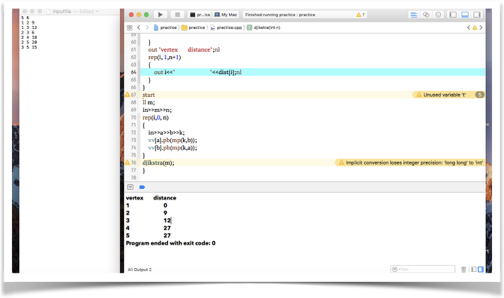
vv[a].pb(mp(k,b));

vv[b].pb(mp(k,a));

}

djikstra(m);

}

**OUTPUT :**

1. MATLAB

CODE :

OUTPUT :

C) SCILAB

CODE :

OUTPUT :