**Name : Lakhan Kumawat**

**Roll No : 1906055**

**Branch : CSE-1**

**Course : CSL4403**

**Lab Experiment 07**

**Write a program to implement Single Shortest path ( Dijkstra’s ) algorithm.**

**Dijkstra's algorithm** (or **Dijkstra's** Shortest Path First **algorithm**, SPF **algorithm**) is an **algorithm** for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. ... For a given source node in the graph, the **algorithm** finds the shortest path between that node and every other.

**Program Code : C++**

#include<bits/stdc++.h>

#include<iostream>

#include<unordered\_map>

using namespace std;

//making a template to store data of different datatypes in hashmap

template<typename T>

class Graph{

unordered\_map<T,list<pair<T,int>>> m;

public:

void addEdge(T u,T v,int dist,bool bidir=true){

//Lets push the edge in map using its predefined functions

m[u].push\_back(make\_pair(v,dist));

//If an edge is bidirectional

if(bidir){

m[v].push\_back(make\_pair(u,dist));

}

}

void printAdj(){

cout<<"Adj List of Given Graph : "<<endl<<endl;

//Lets print the adj list

//I will use foreach loop to iterate over each object in map using first and second

//First iterate over hashmap

for(auto j:m){

cout<<j.first<<"->";

//Second iterate over list

for(auto l:j.second){

cout<<"("<<l.first<<","<<l.second<<")";

}

cout<<endl;

}

}

void dijsktraSSSp(T src){

//Will make an Unordered map t store distance with city

map<T,int> dist;

//Set all the distance to infinity initially

for(auto j:m){ //Note here we are using m

dist[j.first]=INT\_MAX;

}

//Make a set to find out node with minimum distance and set sets data in sorted order

set<pair<int,T>> s; //Every set will be having a pair

//dist of src=0

dist[src]=0;

s.insert(make\_pair(0,src));

while(!s.empty()){

//This will point to the begin node always of the set

auto p= \*(s.begin());

T Setnode = p.second; //Node name will always be second one as defined set earlier

int setnodeDist = p.first;

s.erase(s.begin());

//Iterator over neighbours/children of current node which is just removed

//Childpair will give pairs of a particular node

for(auto childPair:m[Setnode]){

if(setnodeDist+childPair.second< dist[childPair.first]){ //infinity we stored basically

//In the set updation of a praticular node is not possible

//We have to remove the old pair and insert a new pair to simulate updation

T dest = childPair.first;

auto f = s.find(make\_pair(dist[dest],dest));

if(f!=s.end()){

s.erase(f);

}

//Insert the new pair

dist[dest]=setnodeDist+childPair.second;

s.insert(make\_pair(dist[dest],dest));

}

}

}

cout<<"After Traversal : "<<endl;

//Lets print distance to all other nodes from source

for(auto d:dist){

cout<<d.first<<" is located at "<<d.second<<endl;

}

}

};

int main(){

**//Input from USER**

/\*lets make a Graph

Graph<int> g;

int edges,a,b,c;

cin>>edges;

while(edges>0){

cin>>a>>b>>c;

g.addEdge(a,b,c); //public function is defined above

edges--;

}

g.printAdj(); //To print

g.dijsktraSSSp(0);\*/

**//Input already provided**

Graph<string> city;

city.addEdge("Amritsar","Delhi",1);

city.addEdge("Amritsar","Jaipur",4);

city.addEdge("Amritsar","Mumbai",7);

city.addEdge("Delhi","Jaipur",1);

city.addEdge("Jaipur","Mumbai",4);

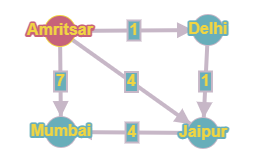
city.printAdj(); //To print

city.dijsktraSSSp("Amritsar");

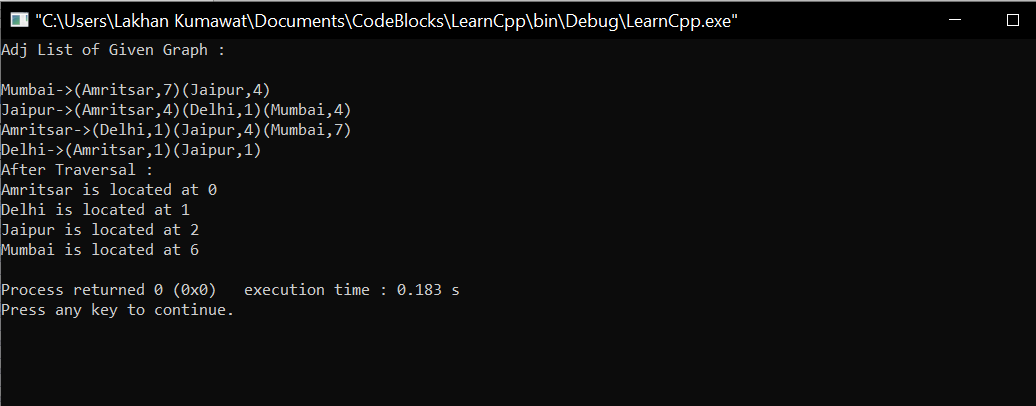
return 0;

}

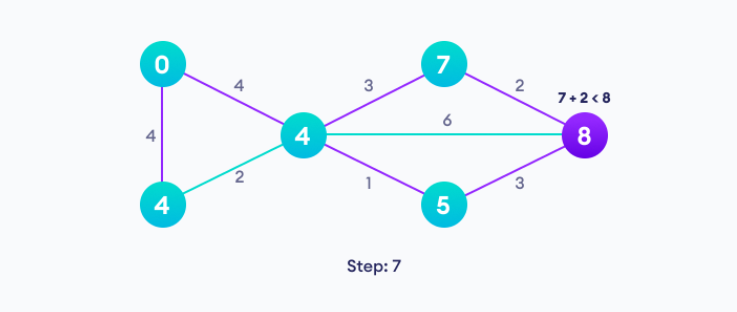
**Input Graph :**

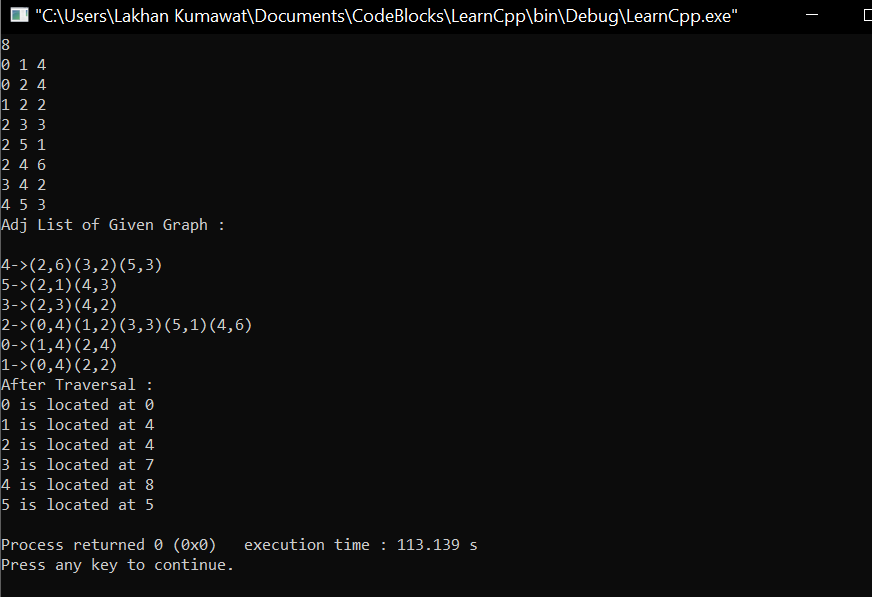
****

**Output :**

****

**Input Graph :**

****

****