#include<bits/stdc++.h>

using namespace std;

template<typename T>

class graph {

public:

map<T, list<pair<T,int> > > adjlist;

void addEdge(T u, T v, int dist,bool bidirec = 1)

{

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

if (bidirec)

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

}

void print\_adj()

{

for (auto n : adjlist)

{

cout << n.first<<" : ";

for (auto a : n.second )

{

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

}

cout << endl;

}

}

void bfs(T u)

{

map< T, bool> visited;

queue<T> q;

q.push(u);

while (!q.empty())

{

T front\_element = q.front();

q.pop();

if (!visited[front\_element]) {

cout << front\_element << " -> ";

visited[front\_element] = true;

}

for (auto a : (adjlist[front\_element]))

{

if(!visited[a.first])

q.push(a.first);

}

}

}

void dfs\_helper(T src,map<T,bool> &visited)

{

visited[src] = true;

cout << src << " -> ";

for (auto a : adjlist[src])

{

if (!visited[a.first])

{

dfs\_helper(a.first, visited);

}

}

}

void dfs\_utility(T src) //calling function of dfs helper

{

map < T, bool> visited;

dfs\_helper(src, visited);

}

void bfs\_sssp(T src)

{

map<T, int> distance;

map<T, T> parent;

for (auto i : adjlist)

{

distance[i.first] = INT\_MAX;

}

queue<T> q;

q.push(src);

distance[src] = 0;

parent[src]=src;

while (!q.empty())

{

T front\_element = q.front();

q.pop();

for (auto neighbours : (adjlist[front\_element]))

{

if (distance[neighbours.first] == INT\_MAX)

{

q.push(neighbours.first);

distance[neighbours.first] = distance[front\_element] + 1;

parent[neighbours.first] = front\_element;

}

}

}

for (auto i : adjlist)

{

cout << "distance to " << i.first << " from " << src << " is " << distance[i.first] << endl;

}

//suppose u want to know the hortest path beeen 2 nodes.

//let destion be == g

T temp = 'g';

while (parent[temp] != temp)

{

cout << temp << "<--";

temp = parent[temp];

}cout << src << endl;

}

void dijkstra(T src)

{

map<T, int> dist;

for (auto i : adjlist)

{

dist[i.first] = INT\_MAX;

}

set<pair<int, T> > s;

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

dist[src] = 0;

while (!s.empty())

{

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

int nodedist = p.first;// sorce node ka distance

T temp = p.second;

s.erase(s.begin());

for (auto neigh : adjlist[temp])

{

if (nodedist + neigh.second < dist[neigh.first])//neigh.second distance h adjlist me

{

//purane wale ko delete karo and naye wale ko dalo

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

if (f != s.end())

{

s.erase(f);

}

dist[neigh.first] = nodedist + neigh.second;

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

}

}

}

for (auto d : dist)

cout << d.first << " and " << d.second << endl;

}

};

int main()

{

graph<char> g;

g.addEdge('0', '1',4, 0);

g.addEdge('0', '7',8, 0);

g.addEdge('1', '7',11, 0);

g.addEdge('1', '2',8, 0);

g.addEdge('7', '8',7, 0);

g.addEdge('2', '8',2, 0);

g.addEdge('8', '6',6, 0);

g.addEdge('2', '5',4, 0);

g.addEdge('6', '5',2, 0);

g.addEdge('2', '3',7, 0);

g.addEdge('3', '3',14, 0);

g.addEdge('3', '4',9, 0);

g.addEdge('5', '4',10, 0);

g.addEdge('7', '6',1, 0);

g.print\_adj();

return 0;

}