dijkshtras\_07.cpp

**Compile:** g++ dijkshtras\_07.cpp -o dijkshtras\_07

**Run:** ./dijkshtras\_07

**Program:**

#include<iostream>

#define INFINITY 999

using namespace std;

class Dijkstra

{

private:

intadjMatrix[15][15];

int predecessor[15],distance[15];

bool mark[15];

int source;

intnumOfVertices;

public:

void read();

void initialize();

intgetClosestUnmarkedNode();

voidcalculateDistance();

void output();

voidprintPath(int);

};

void Dijkstra::read()

{

cout<<"Enter the number of vertices of the graph(should be > 0)\n";

cin>>numOfVertices;

while(numOfVertices<= 0)

{

cout<<"Enter the number of vertices of the graph(should be > 0)\n";

cin>>numOfVertices;

}

cout<<"Enter the adjacency matrix for the graph\n";

cout<<"To enter infinity enter "<<INFINITY<<endl;

for(inti=0;i<numOfVertices;i++)

{

cout<<"Enter the (+ve)weights for the row "<<i<<endl;

for(int j=0;j<numOfVertices;j++)

{

cin>>adjMatrix[i][j];

while(adjMatrix[i][j]<0)

{

cout<<"Weights should be +ve. Enter the weight again\n";

cin>>adjMatrix[i][j];

}

}

}

cout<<"Enter the source vertex\n";

cin>>source;

while((source<0) && (source>numOfVertices-1))

{

cout<<"Source vertex should be between 0 and"<<numOfVertices-1<<endl;

cout<<"Enter the source vertex again\n";

cin>>source;

}

}

void Dijkstra::initialize()

{

for(inti=0;i<numOfVertices;i++)

{

mark[i] = false;

predecessor[i] = -1;

distance[i] = INFINITY;

}

distance[source]= 0;

}

int Dijkstra::getClosestUnmarkedNode()

{

intminDistance = INFINITY;

intclosestUnmarkedNode;

for(inti=0;i<numOfVertices;i++)

{

if((!mark[i]) && ( minDistance>= distance[i]))

{

minDistance = distance[i];

closestUnmarkedNode = i;

}

}

returnclosestUnmarkedNode;

}

void Dijkstra::calculateDistance()

{

initialize();

intminDistance = INFINITY;

intclosestUnmarkedNode;

int count = 0;

while(count <numOfVertices)

{

closestUnmarkedNode = getClosestUnmarkedNode();

mark[closestUnmarkedNode] = true;

for(inti=0;i<numOfVertices;i++)

{

if((!mark[i]) && (adjMatrix[closestUnmarkedNode][i]>0) )

{

if(distance[i] > distance[closestUnmarkedNode]+adjMatrix[closestUnmarkedNode][i])

{

distance[i] = distance[closestUnmarkedNode]+adjMatrix[closestUnmarkedNode][i];

predecessor[i] = closestUnmarkedNode;

}

}

}

count++;

}

}

void Dijkstra::printPath(int node)

{

if(node == source)

cout<<(char)(node + 97)<<"..";

else if(predecessor[node] == -1)

cout<<"No path from “<<source<<”to "<<(char)(node + 97)<<endl;

else

{

printPath(predecessor[node]);

cout<<(char) (node + 97)<<"..";

}

}

void Dijkstra::output()

{

for(inti=0;i<numOfVertices;i++)

{

if(i == source)

cout<<(char)(source + 97)<<".."<<source;

else

printPath(i);

cout<<"->"<<distance[i]<<endl;

}

}

int main()

{

Dijkstra G;

G.read();

G.calculateDistance();

G.output();

return 0;

}

**Output:**

