**CS323-22: Project 6 (CPP)**

**Yida Tao**

**Due date: Nov. 07, 2017**

Algorithm Steps:

step 0: open input and output files

numNodes 🡨 get from input

sourceNode 🡨 get from input

Allocate and initialize all members in the DijkstraSSS class accordingly

step 1: loadMatrix

step 2: setBestCostAry (sourceNode)

setFatherAry (sourceNode)

setToDoAry (sourceNode )

step 3: minNode 🡨 findMinNode()

markMinNode(minNode)

call debugPrint

step 4: currentNode 🡨 1

step 5: if toDoAry[currentNode] equal to 1

newCost 🡨 computeCost(minNode, currentNode)

if newCost < bestCostAry [currentNode]

changeCost (currentNode, newCost)

changeFather (currentNode, minNode)

call debugPrint

step 6: currentNode ++

step 7: repeat step 5-step 6 while currentNode <= numNodes

step 8: repeat step 3 to step 7 until all nodes in toDoAry are 0

step 9: // finding the shortest paths

currentNode 🡨 1

step 10: printShortestPath (currentNode)

// trace from currentNode back to sourceNode (via fatherAry), and print the shortest path from //sourceNode to currentNode with the cost to OutFile1

step 11: currentNode ++

step 12: repeat step 10 and step 11 while currentNode <= numNodes

step 13: close all files.

**Source Code**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class DijktraSSS{

int numNodes;

int sourceNode;

int minNode;

int currentNode;

int newCost;

int \*\*costMatrix;

int \*fatherAry;

int \*toDoAry;

int \*bestCostAry;

int count=0;

public:

~DijktraSSS(){

delete[] costMatrix;

delete[] fatherAry;

delete[] toDoAry;

delete[] bestCostAry;

}

void loadCostMatrix(string in){

ifstream infile;

infile.open(in);

//load # of node and source node

infile >> numNodes;

infile >> sourceNode;

//initialize arrays

fatherAry = new int[numNodes + 1];

toDoAry = new int[numNodes + 1];

bestCostAry = new int[numNodes + 1];

costMatrix = new int\*[numNodes+1];

for(int i = 0; i < numNodes + 1; i++){

fatherAry[i] = i;

toDoAry[i] = 1;

bestCostAry[i] = 9999;

costMatrix[i] = new int[numNodes+1];

for(int j = 0; j < numNodes + 1; j++){

costMatrix[i][j] = 9999;

}

costMatrix[i][i] = 0;

}

//load date to costmatrix

int start = 0;

int end = 0;

int distance = 0;

while(infile >> start) {

infile >> end;

infile >>distance;

costMatrix[start][end] = distance;

}

infile.close();

cout<<"costMatrix:"<<endl;

for(int i = 1; i < numNodes + 1; i++){

for(int j = 1; j < numNodes + 1; j++){

cout<<costMatrix[i][j]<<" ";

}

cout<<endl;

}

}

void setBestCostAry(){

for(int i = 1; i < numNodes + 1; i++){

bestCostAry[i] = costMatrix[sourceNode][i];

}

}

void setToDoAry(){

toDoAry[sourceNode] = 0;

}

void setFatherAry(){

for(int i = 1; i < numNodes + 1; i++){

fatherAry[i] = sourceNode;

}

}

void findMinNode(){

int min = 99999;

for(int i = 1; i < numNodes + 1; i++){

if(toDoAry[i] == 1 && bestCostAry[i] < min){

minNode = i;

min = bestCostAry[i];

}

}

}

bool isAllZero(){

for(int i = 1; i< numNodes + 1; i++){

if(toDoAry[i] == 1){

return false;

}

}

return true;

}

void computeCost(){

newCost = bestCostAry[minNode] + costMatrix[minNode][currentNode];

}

void markMinNode(){

toDoAry[minNode] = 0;

}

void changeFather(){

fatherAry[currentNode] = minNode;

}

void changeCost(){

bestCostAry[currentNode] = newCost;

}

void Dijkstras(){

for(currentNode = 1; currentNode< numNodes + 1; currentNode++){

if(toDoAry[currentNode] == 1){

computeCost();

if(newCost < bestCostAry[currentNode]){

changeCost();

changeFather();

debugPrint();

}

}

}

}

void printShortestPath(){

int temp = 0;

cout<<"===================================="<<endl;

cout<<"There are "<<numNodes<<" nodes in the input graph."<<endl;

cout<<"===================================="<<endl;

cout<<"Source node = "<<sourceNode<<endl;

cout<<"The shortest paths from the source node "<<sourceNode<<" are:"<<endl;

for(currentNode = 1; currentNode< numNodes + 1; currentNode++){

cout<<"The path from "<<sourceNode<<" to "<<currentNode<<" : "<<currentNode;

temp = fatherAry[currentNode];

while(temp != sourceNode){

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

temp = fatherAry[temp];

}

cout<<"<--"<<temp<<" Cost: "<< bestCostAry[currentNode]<<endl;

}

}

void debugPrint(){

cout<<"===================================="<<endl;

cout<<"loop "<<count<<": "<<endl;

cout<<"the sourceNode is: "<<sourceNode<<endl;

cout<<"fatherAry: ";

for(int i = 1; i < numNodes + 1; i++){

cout<<fatherAry[i]<<" ";

}

cout<<endl;

cout<<"bestCostAry: ";

for(int i = 1; i < numNodes + 1; i++){

cout<<bestCostAry[i]<<" ";

}

cout<<endl;

cout<<"toDoAry: ";

for(int i = 1; i < numNodes + 1; i++){

cout<<toDoAry[i]<<" ";

}

cout<<endl;

count++;

}

};

int main(int argc, char \*argv[]) {

if(argv[1]==NULL || argv[2]==NULL || argv[3]==NULL) {

cout<<"no parameter"<<endl;

return 0;

}

ifstream infile;

infile.open(argv[1]);

if(!infile.is\_open()){

cout<<"cant find file"<<endl;

return 0;

}

infile.close();

streambuf \*console = cout.rdbuf();

ofstream out1;

out1.open(argv[2]);

ofstream out2;

out2.open(argv[3]);

//set output to outfile 1

cout.rdbuf(out2.rdbuf());

string inputFile = argv[1];

DijktraSSS \*sss = new DijktraSSS();

//step 0,1

sss->loadCostMatrix(inputFile);

//sss->debugPrint();

//step 2

sss->setBestCostAry();

sss->setFatherAry();

sss->setToDoAry();

while(!sss->isAllZero()){

//step 3

sss->findMinNode();

sss->markMinNode();

sss->debugPrint();

//step 4,5,6,7

sss->Dijkstras();

}

//output to output1

cout.rdbuf(out1.rdbuf());

//step 9,10,11,12

sss->printShortestPath();

//step 13 close all files, delete objects

cout.rdbuf(console);

cout<<"input: "<<argv[1]<<endl;

cout<<"output1: "<<argv[2]<<endl;

cout<<"output2: "<<argv[3]<<endl;

cout<<"done"<<endl;

out1.close();

out2.close();

delete sss;

return 0;

}

**Input**

8

1

1 2 10

2 8 2

3 2 2

6 7 2

1 5 30

8 7 2

1 4 20

3 8 10

1 3 5

8 1 6

3 7 30

4 6 3

3 4 5

4 5 5

5 6 15

7 4 4

7 6 3

6 1 5

8 6 7

1 8 20

7 1 40

5 2 10

7 5 30

5 8 3

2 7 40

2 3 35

2 1 25

6 2 5

6 3 20

4 7 25

4 8 20

**Output 1**

====================================

There are 8 nodes in the input graph.

====================================

Source node = 1

The shortest paths from the source node 1 are:

The path from 1 to 1 : 1<--1 Cost: 0

The path from 1 to 2 : 2 <-- 3<--1 Cost: 7

The path from 1 to 3 : 3<--1 Cost: 5

The path from 1 to 4 : 4 <-- 3<--1 Cost: 10

The path from 1 to 5 : 5 <-- 4 <-- 3<--1 Cost: 15

The path from 1 to 6 : 6 <-- 4 <-- 3<--1 Cost: 13

The path from 1 to 7 : 7 <-- 8 <-- 2 <-- 3<--1 Cost: 11

The path from 1 to 8 : 8 <-- 2 <-- 3<--1 Cost: 9

====================================

There are 8 nodes in the input graph.

====================================

Source node = 2

The shortest paths from the source node 2 are:

The path from 2 to 1 : 1 <-- 8<--2 Cost: 8

The path from 2 to 2 : 2<--2 Cost: 0

The path from 2 to 3 : 3 <-- 1 <-- 8<--2 Cost: 13

The path from 2 to 4 : 4 <-- 7 <-- 8<--2 Cost: 8

The path from 2 to 5 : 5 <-- 4 <-- 7 <-- 8<--2 Cost: 13

The path from 2 to 6 : 6 <-- 7 <-- 8<--2 Cost: 7

The path from 2 to 7 : 7 <-- 8<--2 Cost: 4

The path from 2 to 8 : 8<--2 Cost: 2

====================================

There are 8 nodes in the input graph.

====================================

Source node = 3

The shortest paths from the source node 3 are:

The path from 3 to 1 : 1 <-- 8 <-- 2<--3 Cost: 10

The path from 3 to 2 : 2<--3 Cost: 2

The path from 3 to 3 : 3<--3 Cost: 0

The path from 3 to 4 : 4<--3 Cost: 5

The path from 3 to 5 : 5 <-- 4<--3 Cost: 10

The path from 3 to 6 : 6 <-- 4<--3 Cost: 8

The path from 3 to 7 : 7 <-- 8 <-- 2<--3 Cost: 6

The path from 3 to 8 : 8 <-- 2<--3 Cost: 4

====================================

There are 8 nodes in the input graph.

====================================

Source node = 4

The shortest paths from the source node 4 are:

The path from 4 to 1 : 1 <-- 6<--4 Cost: 8

The path from 4 to 2 : 2 <-- 6<--4 Cost: 8

The path from 4 to 3 : 3 <-- 1 <-- 6<--4 Cost: 13

The path from 4 to 4 : 4<--4 Cost: 0

The path from 4 to 5 : 5<--4 Cost: 5

The path from 4 to 6 : 6<--4 Cost: 3

The path from 4 to 7 : 7 <-- 6<--4 Cost: 5

The path from 4 to 8 : 8 <-- 5<--4 Cost: 8

====================================

There are 8 nodes in the input graph.

====================================

Source node = 5

The shortest paths from the source node 5 are:

The path from 5 to 1 : 1 <-- 8<--5 Cost: 9

The path from 5 to 2 : 2<--5 Cost: 10

The path from 5 to 3 : 3 <-- 1 <-- 8<--5 Cost: 14

The path from 5 to 4 : 4 <-- 7 <-- 8<--5 Cost: 9

The path from 5 to 5 : 5<--5 Cost: 0

The path from 5 to 6 : 6 <-- 7 <-- 8<--5 Cost: 8

The path from 5 to 7 : 7 <-- 8<--5 Cost: 5

The path from 5 to 8 : 8<--5 Cost: 3

====================================

There are 8 nodes in the input graph.

====================================

Source node = 6

The shortest paths from the source node 6 are:

The path from 6 to 1 : 1<--6 Cost: 5

The path from 6 to 2 : 2<--6 Cost: 5

The path from 6 to 3 : 3 <-- 1<--6 Cost: 10

The path from 6 to 4 : 4 <-- 7<--6 Cost: 6

The path from 6 to 5 : 5 <-- 4 <-- 7<--6 Cost: 11

The path from 6 to 6 : 6<--6 Cost: 0

The path from 6 to 7 : 7<--6 Cost: 2

The path from 6 to 8 : 8 <-- 2<--6 Cost: 7

====================================

There are 8 nodes in the input graph.

====================================

Source node = 7

The shortest paths from the source node 7 are:

The path from 7 to 1 : 1 <-- 6<--7 Cost: 8

The path from 7 to 2 : 2 <-- 6<--7 Cost: 8

The path from 7 to 3 : 3 <-- 1 <-- 6<--7 Cost: 13

The path from 7 to 4 : 4<--7 Cost: 4

The path from 7 to 5 : 5 <-- 4<--7 Cost: 9

The path from 7 to 6 : 6<--7 Cost: 3

The path from 7 to 7 : 7<--7 Cost: 0

The path from 7 to 8 : 8 <-- 2 <-- 6<--7 Cost: 10

====================================

There are 8 nodes in the input graph.

====================================

Source node = 8

The shortest paths from the source node 8 are:

The path from 8 to 1 : 1<--8 Cost: 6

The path from 8 to 2 : 2 <-- 6 <-- 7<--8 Cost: 10

The path from 8 to 3 : 3 <-- 1<--8 Cost: 11

The path from 8 to 4 : 4 <-- 7<--8 Cost: 6

The path from 8 to 5 : 5 <-- 4 <-- 7<--8 Cost: 11

The path from 8 to 6 : 6 <-- 7<--8 Cost: 5

The path from 8 to 7 : 7<--8 Cost: 2

The path from 8 to 8 : 8<--8 Cost: 0