#ifndef GRAPH\_H

#define GRAPH\_h

using namespace std;

#include <iostream>

#include <vector>

#include <queue>

#include <map>

#include "node.h"

class Graph {

private:

vector<Node\*>\* vectNodes;

int iSize = 0;

public:

Graph() {

vectNodes = new vector<Node\*>();

}

void addNode(int iKeyNew) {

Node\* tmp = new Node(iKeyNew);

vectNodes->push\_back(tmp);

iSize++;

}

void addEdge(int iKeyFrom,int iKeyTo, int iWeight) {

Node\* nodeFrom = getNode(iKeyFrom);

Node\* nodeTo = getNode(iKeyTo);

if (nodeTo == nullptr || nodeFrom == nullptr) return;

nodeFrom->addEdge(nodeTo, iWeight);

}

void showGraph() {

if (iSize == 0) return;

Node\* nextNode = vectNodes->at(0);

queue<Edge\*>\* qEdges = new queue<Edge\*>();

int\* visitedNodes = new int[iSize];

for (int i = 0; i < iSize; i++) visitedNodes[i] = -1;

cout << "\n" << nextNode->getKey();

visitedNodes[nextNode->getKey()] = 1;

vector<Edge\*>\* baseEdges = nextNode->getEdges();

if (baseEdges->size() == 0) return;

for (int i = 0; i < baseEdges->size(); i++) qEdges->push(baseEdges->at(i));

while (!qEdges->empty()) {

Edge\* nextEdge = qEdges->front();

qEdges->pop();

Node\* prevNode = getNode(nextEdge->getNodeFrom());

nextNode = getNode(nextEdge->getNodeTo());

baseEdges = nextNode->getEdges();

cout << "\n" << prevNode->getKey();

visitedNodes[prevNode->getKey()] = 1;

cout << " - " << nextNode->getKey();

if (visitedNodes[nextNode->getKey()] == -1) {

visitedNodes[nextNode->getKey()] = 1;

for (int i = 0; i < baseEdges->size(); i++)

{

if (visitedNodes[baseEdges->at(i)->getNodeTo()] == -1) {

qEdges->push(baseEdges->at(i));

}

}

}

}

}

void wayFinder(map<int,pair<int,int>>\* wayWeighted, Node\* root, int iKeyResult, vector<int>\* alreadyVisited) {

vector<Edge\*>\* baseEdges = root->getEdges();

for (int i = 0; i < baseEdges->size(); i++) {

if ((\*alreadyVisited)[baseEdges->at(i)->getNodeTo()] == 1) continue;

if ((\*wayWeighted)[baseEdges->at(i)->getNodeTo()].first == -1) {

(\*wayWeighted)[baseEdges->at(i)->getNodeTo()] = make\_pair(baseEdges->at(i)->getWeight() + (((\*wayWeighted)[baseEdges->at(i)->getNodeFrom()].first != -1)?(\*wayWeighted)[baseEdges->at(i)->getNodeFrom()].first : 0), baseEdges->at(i)->getNodeFrom());

}

else {

if ((\*wayWeighted)[baseEdges->at(i)->getNodeTo()].first > (\*wayWeighted)[baseEdges->at(i)->getNodeFrom()].first + baseEdges->at(i)->getWeight()) {

(\*wayWeighted)[baseEdges->at(i)->getNodeTo()] = make\_pair((\*wayWeighted)[baseEdges->at(i)->getNodeFrom()].first + baseEdges->at(i)->getWeight(), baseEdges->at(i)->getNodeFrom());

}

}

}

(\*alreadyVisited)[root->getKey()] = 1;

int minNodeIndex = -1;

for (int i = 0; i < baseEdges->size(); i++) {

if ((\*alreadyVisited)[baseEdges->at(i)->getNodeTo()] == 1) continue;

if (baseEdges->at(i)->getNodeTo() != iKeyResult and ( minNodeIndex == -1 || (\*wayWeighted)[baseEdges->at(minNodeIndex)->getNodeTo()].first == -1 || (\*wayWeighted)[baseEdges->at(minNodeIndex)->getNodeTo()].first > (\*wayWeighted)[baseEdges->at(i)->getNodeTo()].first))

minNodeIndex = i;

}

if (minNodeIndex != -1)

wayFinder(wayWeighted, getNode(baseEdges->at(minNodeIndex)->getNodeTo()), iKeyResult, alreadyVisited);

}

Node\* getNode(int iKey) {

for (int i = 0; i < iSize; i++) {

if (vectNodes->at(i)->getKey() == iKey) return vectNodes->at(i);

}

return nullptr;

}

void clear() {

vectNodes->clear();

iSize = 0;

}

int getSize() {

return iSize;

}

};

#endif