\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

QueueAsArray.h graf

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

template<class T>

class QueueAsArray

{

protected:

T\* array;

int size;

int head;

int tail;

int numberOfElements;

public:

QueueAsArray(int size)

{

this->size = size;

array = new T[size];

head = tail = -1;

numberOfElements = 0;

}

~QueueAsArray()

{

delete[] array;

}

void enqueue(T object)

{

if(numberOfElements==size)

throw "Queue overflow!";

if(++tail==size) tail = 0; // tail = (tail+1)%size;

array[tail] = object;

if(numberOfElements==0) head = tail;

numberOfElements++;

}

T dequeue()

{

if(numberOfElements==0)

throw "Queue underflow!";

T result = array[head];

if(++head==size) head = 0; // head = (head+1)%size;

numberOfElements--;

if(numberOfElements==0) head = tail = -1;

return result;

}

bool isEmpty()

{

return head==-1;

}

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

GraphAsLLists.h 1 2 3 4 5 6 graf

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include "QueueAsArray.h"

#include <iostream>

using namespace std;

template<class T>

class Edge;

template<class T>

class Node

{

public:

T info;

Node<T>\* next;

Edge<T>\* adj;

int inDeg;

int status;

Node(T info)

{

this->info = info;

inDeg = status = 0;

next = NULL;

adj = NULL;

}

};

template<class T>

class Edge

{

public:

Node<T>\* dest;

Edge<T>\* next;

Edge()

{

dest = NULL;

next = NULL;

}

};

template<class T>

class GraphAsLLists

{

private:

Node<T>\* start;

int numberOfNodes;

public:

GraphAsLLists()

{

start = NULL;

numberOfNodes = 0;

}

~GraphAsLLists()

{

Node<T>\* temp1 = start;

Edge<T>\* temp2;

while(temp1!=NULL) // brisanje cvorova

{

temp2 = temp1->adj;

while(temp2!=NULL) // brisanje potega

{

temp1->adj = temp2->next;

delete temp2;

temp2 = temp1->adj;

}

start = temp1->next;

delete temp1;

temp1 = start;

}

}

void printGraph()

{

if(start==NULL)

{

cout << "Graph is empty!" << endl;

return;

}

Node<T>\* temp1 = start;

Edge<T>\* temp2;

while(temp1!=NULL)

{

cout << temp1->info << ": ";

temp2 = temp1->adj;

if(temp2==NULL)

{

cout << "no neighbours" << endl;

temp1 = temp1->next;

continue;

}

while(temp2->next!=NULL)

{

cout << temp2->dest->info << " | ";

temp2 = temp2->next;

}

cout << temp2->dest->info << endl;

temp1 = temp1->next;

}

}

void insertNode(T info)

{

Node<T>\* temp = new Node<T>(info);

temp->next = start;

start = temp;

numberOfNodes++;

}

void insertEdge(T a, T b)

{

Node<T>\* temp1 = findNode(a);

Node<T>\* temp2 = findNode(b);

if(temp1==NULL || temp2==NULL)

return;

Edge<T>\* temp3 = new Edge<T>();

temp3->next = temp1->adj;

temp1->adj = temp3;

temp3->dest = temp2;

}

Node<T>\* findNode(T info)

{

Node<T>\* temp = start;

while(temp!=NULL)

{

if(temp->info==info)

return temp;

temp = temp->next;

}

return NULL;

}

***1g.Projektovati klasu za rad sa orijentisanim grafom predstavljenim listom suseda i***

***implementirati metod za nalaženje čvora sa maksimalnim ulaznim stepenom.***

Node<T>\* maxInDeg()

{

if(start==NULL)

throw "Graph is empty!";

Node<T>\* temp1 = start;

Edge<T>\* temp2;

while(temp1!=NULL)

{

temp1->inDeg = 0;

temp1 = temp1->next;

}

temp1 = start;

while(temp1!=NULL) // postavljanje ulaznih stepena za svaki cvor

{

temp2 = temp1->adj;

while(temp2!=NULL)

{

temp2->dest->inDeg++;

temp2 = temp2->next;

}

temp1 = temp1->next;

}

Node<T>\* nodeMax = start; // cvor sa maksimalnim ulaznim stepenom

int max = 0; // maksimalni ulazni stepen

temp1 = start;

while(temp1!=NULL)

{

if(temp1->inDeg > max)

{

max = temp1->inDeg;

nodeMax = temp1;

}

temp1 = temp1->next;

}

return nodeMax;

}

***2g. Projektovati klasu za rad sa orijentisanim grafom predstavljenim listom suseda i implementirati metod za nalaženje čvora sa maksimalnim izlaznim stepenom.***

Node<T>\* maxOutDeg()

{

if(start==NULL)

throw "Graph is empty!";

Node<T>\* nodeMax = start; // cvor sa maksimalnim izlaznim stepenom

Node<T>\* temp1 = start;

Edge<T>\* temp2;

int max = 0; // maksimalni izlazni stepen

int t;

while(temp1!=NULL)

{

temp2 = temp1->adj;

t = 0;

while(temp2!=NULL)

{

t++;

temp2 = temp2->next;

}

if(t>max)

{

max = t;

nodeMax = temp1;

}

temp1 = temp1->next;

}

return nodeMax;

}

***3g. Projektovati klasu za rad sa orijentisanim grafom predstavljenim listom suseda i implementirati metod za nalaženje čvorova sa izlaznim stepenom 0.***

void zeroOutDegNodes()

{

if(start==NULL)

throw "Graph is empty!";

Node<T>\* temp = start;

int n = 0; // broj trazenih cvorova

while(temp!=NULL)

{

if(temp->adj==NULL)

n++;

temp = temp->next;

}

if(n==0)

{

cout << "No such nodes!" << endl;

return;

}

Node<T>\*\* nodes = new Node<T>\*[n];

temp = start;

int i = 0;

while(temp!=NULL)

{

if(temp->adj==NULL)

nodes[i++] = temp;

temp = temp->next;

}

cout << "Nodes(node) with zero out degree are(is): ";

for(i=0; i<n-1; i++)

cout << nodes[i]->info << ", ";

cout << nodes[n-1]->info << "." << endl;

delete[] nodes;

}

***4g. Projektovati klasu za rad sa orijentisanim grafom predstavljenim listom suseda i implementirati metod za nalaženje čvorova sa ulaznim stepenom 0.***

void zeroInDegNodes()

{

if(start==NULL)

throw "Graph is empty!";

Node<T>\* temp1 = start;

Edge<T>\* temp2;

while(temp1!=NULL)

{

temp1->inDeg = 0;

temp1 = temp1->next;

}

temp1 = start;

while(temp1!=NULL) // postavljanje ulaznih stepena za svaki cvor

{

temp2 = temp1->adj;

while(temp2!=NULL)

{

temp2->dest->inDeg++;

temp2 = temp2->next;

}

temp1 = temp1->next;

}

temp1 = start;

int n = 0; // broj trazenih cvorova

while(temp1!=NULL)

{

if(temp1->inDeg==0)

n++;

temp1 = temp1->next;

}

if(n==0)

{

cout << "No such nodes!" << endl;

return;

}

Node<T>\*\* nodes = new Node<T>\*[n];

temp1 = start;

int i = 0;

while(temp1!=NULL)

{

if(temp1->inDeg==0)

nodes[i++] = temp1;

temp1 = temp1->next;

}

cout << "Nodes(node) with zero in degree are(is): ";

for(i=0; i<n-1; i++)

cout << nodes[i]->info << ", ";

cout << nodes[n-1]->info << "." << endl;

delete[] nodes;

}

***5g. Projektovati klasu za rad sa orijentisanim grafom predstavljenim listom suseda i implementirati metod za utvrđivanje da li je graf jako povezan (postoji put između svaka dva čvora u grafu).***

void breadthFirst(Node<T>\* node)

{

Node<T>\* temp1 = start;

Node<T>\* temp2;

Edge<T>\* temp3;

while(temp1!=NULL)

{

temp1->status = 0;

temp1 = temp1->next;

}

temp1 = node;

QueueAsArray<Node<T>\*> queue(numberOfNodes);

queue.enqueue(temp1);

while(!queue.isEmpty())

{

temp2 = queue.dequeue();

if(temp2->status==1)

continue;

temp2->status = 1;

temp3 = temp2->adj;

while(temp3!=NULL)

{

queue.enqueue(temp3->dest);

temp3 = temp3->next;

}

}

}

bool isStronglyConnected()

{

if(start==NULL)

throw "Graph is empty!";

bool u = true;

Node<T>\* temp1 = start;

Node<T>\* temp2;

while(temp1!=NULL && u)

{

breadthFirst(temp1);

temp2 = start;

while(temp2!=NULL && u)

{

if(temp2->status==0)

u = false;

temp2 = temp2->next;

}

temp1 = temp1->next;

}

return u;

}

**6g. Projektovati klasu za rad sa orijentisanim grafom predstavljenim listom suseda i implementirati metod za formiranje matrice susedstva.**

void adjecencyMatrix()

{

if(start==NULL)

throw "Graph is empty!";

int\*\* result = new int\*[numberOfNodes];

for(int i=0; i<numberOfNodes; i++)

result[i] = new int[numberOfNodes];

for(int i=0; i<numberOfNodes; i++)

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

result[i][j] = 0;

Node<T>\* temp1 = start;

Edge<T>\* temp2;

int i = 0;

while(temp1!=NULL)

{

temp1->status = i++;

temp1 = temp1->next;

}

temp1 = start;

while(temp1!=NULL)

{

temp2 = temp1->adj;

while(temp2!=NULL)

{

result[temp1->status][temp2->dest->status] = 1;

temp2 = temp2->next;

}

temp1 = temp1->next;

}

for(int i=0; i<numberOfNodes; i++)

{

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

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

cout << endl;

}

for(int i=0; i<numberOfNodes; i++)

delete result[i];

delete[] result;

}

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

GraphAsArrays.h 7 8 9 10 graf

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# include <iostream>

using namespace std;

template<class Node>

class GraphAsArrays

{

private:

Node\* nodes;

int\*\* adjecencyMatrix;

int numberOfNodes;

int maxNumberOfNodes;

public:

GraphAsArrays(int n)

{

nodes = new Node[n];

adjecencyMatrix = new int\*[n];

for(int i=0; i<n; i++)

adjecencyMatrix[i] = new int[n];

for(int i=0; i<n; i++)

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

adjecencyMatrix[i][j] = 0;

maxNumberOfNodes = n;

numberOfNodes = 0;

}

~GraphAsArrays()

{

if(nodes!=NULL)

delete[] nodes;

if(adjecencyMatrix!=NULL)

{

for(int i=0; i<numberOfNodes; i++)

delete[] adjecencyMatrix[i];

delete[] adjecencyMatrix;

}

}

void printGraph()

{

if(numberOfNodes==0)

{

cout << "Graph is empty!" << endl;

return;

}

for(int i=0; i<numberOfNodes; i++)

{

cout << nodes[i] << ": ";

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

if(adjecencyMatrix[i][j]==1)

cout << nodes[j] << " ";

cout << endl;

}

}

void insertNode(Node n)

{

if(numberOfNodes==maxNumberOfNodes)

throw "Graph is full!";

nodes[numberOfNodes++] = n;

}

void insertEdge(Node n1, Node n2)

{

int temp1 = findNode(n1);

int temp2 = findNode(n2);

if(temp1==-1 || temp2==-1)

return;

adjecencyMatrix[temp1][temp2] = 1;

}

int findNode(Node n)

{

int i=0;

while(i<numberOfNodes)

{

if(nodes[i]==n)

return i;

i++;

}

return -1;

}

***7g. Projektovati klasu za rad sa orijentisanim grafom predstavljenim matricom susedstva i implementirati metod za nalaženje čvora sa maksimalnim ulaznim stepenom.***

Node maxInDeg()

{

if(numberOfNodes==0)

throw "Graph is empty!";

int max = 0;

int jmax;

int s;

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

{

s = 0;

for(int i=0; i<numberOfNodes; i++)

if(adjecencyMatrix[i][j]==1)

s++;

if(s>max)

{

max = s;

jmax = j;

}

}

return nodes[jmax];

}

***8g. Projektovati klasu za rad sa orijentisanim grafom predstavljenim matricom susedstva i implementirati metod za nalaženje čvora sa maksimalnim izlaznim stepenom.***

Node maxOutDeg()

{

if(numberOfNodes==0)

throw "Graph is empty!";

int max = 0;

int imax;

int s;

for(int i=0; i<numberOfNodes; i++)

{

s = 0;

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

if(adjecencyMatrix[i][j]==1)

s++;

if(s>max)

{

max = s;

imax = i;

}

}

return nodes[imax];

}

***9g. Projektovati klasu za rad sa orijentisanim grafom predstavljenim matricom susedstva i implementirati metod za nalaženje čvorova sa izlaznim stepenom 0.***

void zeroOutDegNodes()

{

int n = 0;

bool u;

for(int i=0; i<numberOfNodes; i++)

{

u = true;

for(int j=0; j<numberOfNodes && u; j++)

if(adjecencyMatrix[i][j]==1)

u = false;

if(u)

n++;

}

if(n==0)

{

cout << "No such nodes!" << endl;

return;

}

cout << "Nodes(node) with zero out degree are(is): ";

for(int i=0; i<numberOfNodes; i++)

{

u = true;

for(int j=0; j<numberOfNodes && u; j++)

if(adjecencyMatrix[i][j]==1)

u = false;

if(u)

cout << nodes[i] << " ";

}

cout << endl;

}

***10g. Projektovati klasu za rad sa orijentisanim grafom predstavljenim matricom susedstva i implementirati metod za nalaženje čvorova sa ulaznim stepenom 0.***

void zeroInDegNodes()

{

int n = 0;

bool u;

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

{

u = true;

for(int i=0; i<numberOfNodes && u; i++)

if(adjecencyMatrix[i][j]==1)

u = false;

if(u)

n++;

}

if(n==0)

{

cout << "No such nodes!" << endl;

return;

}

cout << "Nodes(node) with zero in degree are(is): ";

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

{

u = true;

for(int i=0; i<numberOfNodes && u; i++)

if(adjecencyMatrix[i][j]==1)

u = false;

if(u)

cout << nodes[j] << " ";

}

cout << endl;

}

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

main.cpp graf

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#pragma once

#include "GraphAsArrays.h"

#include "GraphAsLLists.h"

int main()

{

GraphAsLLists<int> graph1;

graph1.insertNode(4);

graph1.insertNode(3);

graph1.insertNode(2);

graph1.insertNode(1);

graph1.insertEdge(1, 4);

graph1.insertEdge(2, 1);

graph1.insertEdge(3, 2);

graph1.insertEdge(3, 1);

graph1.insertEdge(4, 3);

graph1.insertEdge(4, 2);

graph1.insertEdge(4, 1);

graph1.printGraph();

try

{

cout << "Node with max in degree is: " << graph1.maxInDeg()->info << endl;

cout << "Node with max out degree is: " << graph1.maxOutDeg()->info << endl;

graph1.zeroOutDegNodes();

graph1.zeroInDegNodes();

if(graph1.isStronglyConnected())

cout << "Graph is strongly connected!" << endl;

else cout << "Graph isn't strongly connected!" << endl;

graph1.adjecencyMatrix();

}

catch(const char\* exception)

{

cout << exception << endl;

}

GraphAsArrays<char> graph2(10);

graph2.insertNode('A');

graph2.insertNode('B');

graph2.insertNode('C');

graph2.insertNode('D');

graph2.insertNode('E');

graph2.insertEdge('A', 'B');

graph2.insertEdge('A', 'D');

graph2.insertEdge('B', 'C');

graph2.insertEdge('B', 'E');

graph2.insertEdge('C', 'A');

graph2.insertEdge('D', 'B');

graph2.insertEdge('D', 'C');

graph2.insertEdge('E', 'A');

graph2.insertEdge('E', 'B');

graph2.insertEdge('E', 'D');

graph2.printGraph();

try {

cout << "Node with max in degree is: " << graph2.maxInDeg() << endl;

cout << "Node with max out degree is: " << graph2.maxOutDeg() << endl;

graph2.zeroOutDegNodes();

graph2.zeroInDegNodes();

}

catch(const char\* exception)

{

cout << exception << endl;

}

return 0;}