МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФБГОУ ВО «ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра «Программная инженерия»

**Отчёт**

По лабораторной работе №3

**«Представление графов в памяти ЭВМ»**

**Выполнили:**

Студенты группы 92-ПГ

Погосян Ж.В.

**Проверили:**

Ужаринский А.Ю.

Орёл — 2021

**Код программы:**

**Main:**

#include <iostream>

#include <fstream>

#include <sstream>

#include <queue>

#include <string>

#include <vector>

#include <forward\_list>

using namespace std;

class Graph {

public:

Graph() : head(nullptr) {}

~Graph() {};

bool Init(const std::string& path\_to\_file);

bool Insert(int node, std::vector<int> nodes);

bool Delete(int node);

bool FindEdge(int node1, int node2);

void ShowGraph();

private:

class Node {

public:

Node(int \_num, Node\* \_next = nullptr) :

num(\_num), next(\_next) {}

int num;

Node\* next;

std::forward\_list<Node\*> nodes;

};

Node\* head;

std::vector<int> split(std::string& str, char delim);

std::vector<std::vector<int>> BuildMatrix(const std::string& path\_to\_file);

Node\* FindNode(int num);

void clear();

};

//Алгоритм инициализации(чтения из файла)

bool Graph::Init(const string & path\_to\_file) {

clear();

auto matrix = BuildMatrix(path\_to\_file);

if (matrix.empty())

return false;

for (int i = 0; i < matrix.size(); i++) {

try {

head = new Node(i + 1, head);

}

catch (bad\_alloc) {

return false;

}

}

auto riter = matrix.rbegin();

Node\* currentNode = head;

for (; riter != matrix.rend(); ++riter) {

Node\* workPointer = head;

for (auto initer = riter->rbegin(); initer != riter->rend(); ++initer) {

if (\*initer == 1) {

currentNode->nodes.push\_front(workPointer);

}

workPointer = workPointer->next;

}

currentNode = currentNode->next;

}

return true;

}

//Алгоритм добавления узла

bool Graph::Insert(int node, vector<int> nodes) {

if (node < 0 || FindNode(node))

return false;

if (nodes.empty()) {

head = new Node(node, head);

return true;

}

sort(nodes.begin(), nodes.end());

auto last = unique(nodes.begin(), nodes.end());

nodes.erase(last, nodes.end());

vector<Node\*> node\_pointers;

for (auto x : nodes) {

Node\* result = FindNode(x);

if (!result)

return false;

else

node\_pointers.push\_back(result);

}

head = new Node(node, head);

head->nodes.assign(node\_pointers.begin(), node\_pointers.end());

return true;

}

//Алгоритм исключения узла

bool Graph::Delete(int node) {

Node\* to\_delete = FindNode(node);

if (!to\_delete)

return false;

if (to\_delete == head) {

head = head->next;

}

else {

Node\* pre\_delete;

pre\_delete = head;

while (pre\_delete->next != to\_delete)

pre\_delete = pre\_delete->next;

pre\_delete->next = pre\_delete->next->next;

}

Node\* current = head;

while (current) {

current->nodes.remove(to\_delete);

current = current->next;

}

delete to\_delete;

return true;

}

//Алгоритм поиска ребра

bool Graph::FindEdge(int node1, int node2) {

Node\* result1 = FindNode(node1);

Node\* result2 = FindNode(node2);

if (!result1 || !result2)

return false;

if (

find(result1->nodes.begin(), result1->nodes.end(), result2) != result1->nodes.end() ||

find(result2->nodes.begin(), result2->nodes.end(), result1) != result2->nodes.end()

) {

return true;

}

return false;

}

//Алгоритм отображения графа

void Graph::ShowGraph() {

Node\* currentNode = head;

while (currentNode) {

cout << currentNode->num << " - ";

for (auto node : currentNode->nodes)

cout << node->num << " ";

cout << endl;

currentNode = currentNode->next;

}

}

//Алгоритм построения матрицы

vector<vector<int>> Graph::BuildMatrix(const string & path\_to\_file) {

ifstream ifs(path\_to\_file);

string buffer;

vector<vector<int>> result;

if (!ifs.is\_open())

return result;

while (getline(ifs, buffer, '\n'))

result.push\_back(split(buffer, ' '));

ifs.close();

return result;

}

//Алгоритм разбиения строки(для генерации матрицы)

vector<int> Graph::split(string & str, char delim) {

string tmp;

stringstream sstr(str);

vector<int> result;

while (getline(sstr, tmp, ' '))

result.push\_back(stoi(tmp));

return result;

}

//Алгоритм поиска узла

Graph::Node \* Graph::FindNode(int num) {

Node\* current = head;

while (current) {

if (current->num == num)

return current;

current = current->next;

}

return nullptr;

}

//Алгоритм очистки графа

void Graph::clear() {

if (!head)

return;

Node\* to\_delete = head;

while (to\_delete) {

head = head->next;

delete to\_delete;

to\_delete = head;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

Graph graph;

graph.Init("test");

char input;

while (true) {

cout << "1 - добавить новую вершину" << endl;

cout << "2 - исключить вершину" << endl;

cout << "3 - найти ребро" << endl;

cout << "4 - вывести граф" << endl;

cout << "5 - чтение из файла" << endl;

cout << "6 - выйти" << endl;

input = getchar();

switch (input) {

case '1': {

int x, node;

vector<int> nodes;

cout << "Введите номер вершины: ";

cin >> x;

do {

cout << "Добавить дополнительную вершину (y/n): ";

cin >> input;

if (input == 'y') {

cout << "Номер вершины: ";

cin >> input;

nodes.push\_back(static\_cast<int>(input - 48));

}

else {

break;

}

} while (true);

if (graph.Insert(x, nodes)) {

cout << "Вершина успешно добавлена";

}

else {

cout << "Невозможно добавить вершину";

}

cin.ignore();

input = getchar();

break;

}

case '2': {

int x;

cout << "Введите вершину: ";

cin >> x;

if (graph.Delete(x))

cout << "Вершина удалена";

else

cout << "Вершина не найдена";

cin.ignore();

input = getchar();

break;

}

case '3': {

int x1, x2;

cout << "Введите первую вершину: ";

cin >> x1;

cout << "Введите вторую вершину: ";

cin >> x2;

if (graph.FindEdge(x1, x2))

cout << "Ребро найдено"<< endl;

else

cout << "Между вершинами не найдено рёбер"<< endl;

cin.ignore();

input = getchar();

break;

}

case '4': {

graph.ShowGraph();

input = getchar();

break;

}

case '5': {

graph.Init("test");

cout << "Файл перечитан" << endl;

input = getchar();

break;

}

case '6': {

return 0;

}

}

}

}