

**matrix.txt**

A B C D E

A 0 1 1 0 0

B 0 0 1 0 1

C 0 0 0 1 0

D 0 1 0 0 1

E 0 0 0 0 0

(входные данные из задания на защите)

Предыдущее содержимое файла:

A B C D

A 0 1 0 0

B 1 0 0 1

C 0 0 0 1

D 0 0 0 1

Пояснение:

A <-> B

v

C > D<-.

`--`

**LAB3.h**

#pragma once

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

template <typename T>

struct Node;

struct Vertex;

typedef Node<Node<Vertex>\*> VertexLink;//Узел списка рёбер, указывающий на узел списка вершин, содержащий вершину

template <typename T>

struct Node {

T value;

Node<T>\* next = nullptr;

};

struct Vertex {

char name;

VertexLink\* link\_to = nullptr;

VertexLink\* link\_from = nullptr;

~Vertex();

};

class Graph {

private:

Node<Vertex>\* first = nullptr;

public:

void clear();

void from\_file(ifstream& file);

void add\_vertex(char name);

void remove\_vertex(char name);

void connect(char v1, char v2);

bool edge\_exist(char v1, char v2);

//Просмотр графа в виде матрицы смежности

void show\_as\_matrix(ostream& destination);

};

**AISD\_LAB3.cpp**

#include "LAB3.h"

Graph graph;

char input\_vertex() {

char vertex;

cout << "Введите название вершины >> ";

cin >> vertex;

graph.add\_vertex(vertex);

return vertex;

}

void input\_link(char v1) {

cout << "Добавить вершине " << v1 << " связи с вершинами...\n";

cin.ignore();

while (true) {

string inp;

cout << "... >> ";

getline(cin, inp);

if (inp == "\0") break;

if (inp.length() > 1) {

cout << "Введите одну латинскую букву!" << endl;

continue;

}

graph.connect(v1, inp[0]);

cout << "Успех: Связь добавлена в граф." << endl;

}

}

int main() {

setlocale(LC\_ALL, "ru");

while (true) {

cout << "\033[0m";//Сделать текст стандартным

cout <<

"\

Список команд:\n\

1. Добавить вершину.\n\

2. Добавить связи от вершины.\n\

3. Удалить вершину.\n\

4. Проверить наличие введённого ребра.\n\

5. Отобразить граф в виде матрицы смежности.\n\

\n\

6. Построение графа по данным из файла.\n\

0. Выход\n\

";

cout << "\033[1m";//Сделать текст жирным

string cmd;

cout << "Номер команды >> ";

cin >> cmd;

string answer = "";

try {

if (cmd == "1") {

char vertex = input\_vertex();

answer = "Вершина добавлена.";

}

else if (cmd == "2") {

char vertex;

cout << "Введите название вершины >> ";

cin >> vertex;

input\_link(vertex);

answer = "Все рёбра добавлены.";

}

else if (cmd == "3") {

char vertex;

cout << "Введите название вершины >> ";

cin >> vertex;

graph.remove\_vertex(vertex);

answer = "Вершина успешно удалена.";

}

else if (cmd == "4") {

char v1;

cout << "Введите название первой вершины >> ";

cin >> v1;

char v2;

cout << "Введите название второй вершины >> ";

cin >> v2;

answer = "Ребро " + string(1, v1) + string(1, v2) + " ";

answer += graph.edge\_exist(v1, v2) ? "найдено." : "не найдено.";

}

else if (cmd == "5") {

graph.show\_as\_matrix(cout);

}

else if (cmd == "6") {

ifstream file = ifstream("matrix.txt", ios::in);

graph.from\_file(file);

file.close();

answer = "Граф построен.";

}

else if (cmd == "0") {

cout << "Выход и программы.";

graph.clear();

exit(0);

}

else {

answer = "Команда не распознана.";

}

cout << answer << endl;

}

catch (exception e) {

cout << "Ошибка: " << e.what() << endl;

}

}

return 0;

}

**module.cpp**

#include "LAB3.h"

Vertex::~Vertex() {

//В этом деструкторе удаляются все рёбра, связанные с вершиной

VertexLink\* link;

//Очистка to у других, всех from у удаляемого

link = link\_from;

while (link != nullptr) {

VertexLink\* link\_to = link->value->value.link\_to;

//Первый же указывает на удаляемый

if (&(link\_to->value->value) == this) {

link->value->value.link\_to = link\_to->next;

delete link\_to;

}

//Ищем какой указывает на удаляемый

else {

while (&(link\_to->next->value->value) != this)

link\_to = link\_to->next;

VertexLink\* to\_delete = link\_to->next;

link\_to->next = link\_to->next->next;

delete to\_delete;

}

VertexLink\* next\_link = link->next;

delete link;

link = next\_link;

}

//Очистка from у других, всех to у удаляемого

link = link\_to;

while (link != nullptr) {

VertexLink\* link\_from = link->value->value.link\_from;

//Первый же указывает на удаляемый

if (&(link\_from->value->value) == this) {

link->value->value.link\_from = link\_from->next;

delete link\_from;

}

//Ищем какой указывает на удаляемый

else {

while (&(link\_from->next->value->value) != this)

link\_from = link\_from->next;

VertexLink\* to\_delete = link\_from->next;

link\_from->next = link\_from->next->next;

delete to\_delete;

}

VertexLink\* next\_link = link->next;

delete link;

link = next\_link;

}

}

void Graph::clear() {

Node<Vertex>\* vertex = first;

while (vertex) {

Node<Vertex>\* next = vertex->next;

delete vertex;

vertex = next;

}

first = nullptr;

}

void Graph::add\_vertex(char name) {

if (first == nullptr) {

first = new Node<Vertex>;

first->value.name = name;

}

else {

Node<Vertex>\* vertex = first;

if (vertex->value.name == name) throw exception("Вершина уже существует!");

while (vertex->next != nullptr) {

vertex = vertex->next;

if (vertex->value.name == name) throw exception("Вершина уже существует!");

}

vertex->next = new Node<Vertex>;

vertex->next->value.name = name;

}

}

void Graph::remove\_vertex(char name) {

if (first == nullptr) throw exception("Вершины не существует!");

Node<Vertex>\* vertex;

//Если первый элемент - искомая вершина

if (first->value.name == name) {

vertex = first;

first = vertex->next;

}

//Если первый элемент - не искомая вершина

else {

Node<Vertex>\* vertex\_prev = first;

while (vertex\_prev->next != nullptr) {

if (vertex\_prev->next->value.name == name) break;

vertex\_prev = vertex\_prev->next;

}

if (vertex\_prev->next == nullptr) throw exception("Вершины не существует!");

vertex = vertex\_prev->next;

vertex\_prev->next = vertex->next;

}

delete vertex;

}

void Graph::connect(char from, char to) {

Node<Vertex>\* vertex\_from = nullptr;

Node<Vertex>\* vertex\_to = nullptr;

Node<Vertex>\* vertex = first;

while (vertex != nullptr) {

if (vertex->value.name == from)

vertex\_from = vertex;

if (vertex->value.name == to)

vertex\_to = vertex;

if (vertex\_from != nullptr && vertex\_to != nullptr) break;

vertex = vertex->next;

}

if(vertex == nullptr) throw exception("Не существует такой пары вершин!");

if (vertex\_from->value.link\_to == nullptr) {

vertex\_from->value.link\_to = new VertexLink;

vertex\_from->value.link\_to->value = vertex\_to;

}

else {

VertexLink\* link = vertex\_from->value.link\_to;

while (true) {

if (link->value->value.name == to) throw exception("Такое ребро уже существует!");

if (link->next != nullptr)

link = link->next;

else

break;

}

link->next = new VertexLink;

link->next->value = vertex\_to;

}

if (vertex\_to->value.link\_from == nullptr) {

vertex\_to->value.link\_from = new VertexLink;

vertex\_to->value.link\_from->value = vertex\_from;

}

else {

VertexLink\* link = vertex\_to->value.link\_from;

while (true) {

if (link->value->value.name == from) throw exception("Такое ребро уже существует!");

if (link->next != nullptr)

link = link->next;

else

break;

}

link->next = new VertexLink;

link->next->value = vertex\_from;

}

}

bool Graph::edge\_exist(char from, char to) {

Node<Vertex>\* vertex = first;

while (vertex != nullptr) {

if (vertex->value.name == from) {

VertexLink\* link = vertex->value.link\_to;

while (link != nullptr) {

if (link->value->value.name == to) return true;

link = link->next;

}

return false;

}

if (vertex->value.name == to) {

VertexLink\* link = vertex->value.link\_from;

while (link != nullptr) {

if (link->value->value.name == from) return true;

link = link->next;

}

return false;

}

vertex = vertex->next;

}

throw exception("Не существует такой пары вершин!");

}

void Graph::from\_file(ifstream& file) {

clear();

string row;

getline(file, row);

if (row == "\0") throw exception("Неожиданный конец файла!");

row += " ";//Применить данные в конце строки, не отделённые пробелом

string vertex\_name = "";

for (char c : row) {

if (c == ' ') {

if (vertex\_name != "") {

if (vertex\_name.length() > 1) throw exception("Вершина должна содержать одну латинскую букву!");

add\_vertex(vertex\_name[0]);

vertex\_name = "";

}

}

else

vertex\_name += c;

}

if (first == nullptr) return;

Node<Vertex>\* v1 = first;

while(v1) {

getline(file, row);

if (row == "\0") throw exception("Неожиданный конец файла!");

row += " ";//Применить данные в конце строки, не отделённые пробелом

string data = "";

Node<Vertex>\* v2 = first;

bool reading\_v1\_name = true;

for (char c : row) {

if (c == ' ') {

if (data == "") continue;

if (reading\_v1\_name) {

if (data != string(1, v1->value.name)) throw exception("Неверный заголовок строки!");

reading\_v1\_name = false;

data = "";

}

else {

if (data == "1")

connect(v1->value.name, v2->value.name);

else if (data != "0")

throw exception("Неверное заполнение таблицы!");

v2 = v2->next;

data = "";

}

}

else data += c;

}

v1 = v1->next;

}

}

//Функция вывода матрицы смежности

void Graph::show\_as\_matrix(ostream& destination) {

//Подсчёт размера

int size = 0;

Node<Vertex>\* vertex = first;

while (vertex) {

size++;

vertex = vertex->next;

}

//Первая строка таблицы и матрица смежности

string table\_header = " ";

bool\* matrix = new bool[size \* size];

//Заполнение матрицы данными о рёбрах

Node<Vertex>\* v1 = first;

Node<Vertex>\* v2 = first;

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

table\_header += string(1, v1->value.name) + " ";

v2 = first;

for (int j = 0; j < size; j++) {

matrix[i \* size + j] = edge\_exist(v1->value.name, v2->value.name);

v2 = v2->next;

}

v1 = v1->next;

}

//Заполнение table текстовыми данными

string table = table\_header + "\n";

int i = 0;

vertex = first;

while (vertex) {

table += string(1, vertex->value.name);

for (int j = 0; j < size; j++) {

table += matrix[i \* size + j] ? " 1" : " 0";

}

table += "\n";

vertex = vertex->next;

i++;

}

delete[] matrix;

destination << table;

}