Институт приборостроения, автоматизации и информационных технологий

Кафедра информационных систем и цифровых технологий

Дисциплина «Алгоритмы и структуры данных»

Отчет к лабораторной работе № 3

«Представление графов в памяти ЭВМ»

Выполнил:

Василения Иван Валерьевич

Мельниченко Артём Олегович

Принял:

Рыженков Д.В.

Орёл, 2024г

Листинг файла main.cpp:

#include <iostream>

#include <fstream>

#include <string>

#include <sstream>

#include <iomanip>

#include "Graph.h"

using namespace std;

void showGraph(Graph\* graph) {

    vertex\* vert = graph->head;

    while (vert != nullptr) {

        cout << vert->name << " -> {";

        for (auto v : vert->links) {

            cout << v->name << ", ";

        }

        cout << "}" << endl;

        vert = vert->next;

    }

    cout << endl;

}

void readFromFile(Graph\* graph) {

    graph->clear();

    ifstream inFile("data.txt");

    if (!inFile) {

        cerr << "Не удалось открыть файл для чтения" << endl;

        return;

    }

    string line, st;

    getline(inFile, line);

    stringstream ss(line);

    while (getline(ss, st, ',')) {

        graph->addVertex((char)st[0]);

    }

    vertex\* currVert = graph->head;

    while (getline(inFile, line)) {

        vertex\* link = graph->head;

        stringstream sstream(line);

        string el;

        while (getline(sstream, el, ',')) {

            char connectValue = el[0];

            if (connectValue != '0') {

                currVert->links.push\_back(link);

            }

            link = link->next;

        }

        currVert = currVert->next;

    }

    cout << "Чтение успешно" << endl;

}

void addVertex(Graph\* graph) {

    cout << "Введите новую вершину: ";

    char vert;

    cin >> vert;

    if (!graph->addVertex(vert)) cout << "Вершина уже существует" << endl;

}

void delVertex(Graph\* graph) {

    cout << "Введите вершину, которую нужно удалить: ";

    char vert;

    cin >> vert;

    if (!graph->delVertex(vert)) cout << "Такой вершины не существует" << endl;

}

void addLink(Graph\* graph) {

    char vert1, vert2;

    cout << "Введите вершину 1: ";

    cin >> vert1;

    cout << endl;

    cout << "Введите вершину 2: ";

    cin >> vert2;

    cout << endl;

    int resp = graph->addLink(vert1, vert2);

    if (resp == -1) {

        cout << "ну нет короче вершины";

    }

    else if (resp == -2) {

        cout << "связь уже существует";

    }

    else {

        cout << "Связь добавлена";

    }

    cout << endl;

}

void findLink(Graph\* graph) {

    char vert1, vert2;

    cout << "Введите вершину 1: ";

    cin >> vert1;

    cout << endl;

    cout << "Введите вершину 2: ";

    cin >> vert2;

    cout << endl;

    if (graph->findLink(vert1, vert2)) {

        cout << "Связь найдена" << endl;

    }

    else {

        cout << "Связь не найдена" << endl;

    }

}

void showMenu() {

    cout<< "1. Показать\n"

        << "2. Считать из файла\n"

        << "3. Добавить вершину\n"

        << "4. Удалить вершину\n"

        << "5. Добавить ребро\n"

        << "6. Поиск ребра\n"

        << endl

        << "0. ВЫХОД\n"

        ;

}

int main() {

    setlocale(LC\_ALL, "Russian");

    char fun = '0';

    Graph graph;

    graph.addVertex('a');

    graph.addVertex('b');

    graph.addVertex('c');

    graph.addVertex('d');

    while (1) {

        system("cls");

        showMenu();

        cin >> fun;

        system("cls");

        switch (fun)

        {

        case '1':

            showGraph(&graph);

            break;

        case '2':

            readFromFile(&graph);

            break;

        case '3':

            addVertex(&graph);

            break;

        case '4':

            delVertex(&graph);

            break;

        case '5':

            addLink(&graph);

            break;

        case '6':

            findLink(&graph);

            break;

        case '0':

            return 0;

        default:

            cout << "Выбрана неверная функция" << endl;

            break;

        }

        system("pause");

    }

}

Листинг файла Graph.h:

#pragma once

#include <vector>

struct vertex {

    char name;

    std::vector<vertex\*> links;

    vertex\* next = nullptr;

    vertex(char name) {

        this->name = name;

    }

    ~vertex() {

        next = nullptr;

        links.clear();

    }

};

class Graph {

private:

    vertex\* head = nullptr;

    vertex\* tail = nullptr;

    size\_t size = 0;

public:

    friend void showGraph(Graph\* graph);

    friend void readFromFile(Graph\* graph);

    bool isEmpty() {

        return head == nullptr;

    }

    vertex\* findVertex(char name, vertex\*\* vertPrev = nullptr) {

        if (isEmpty()) return nullptr;

        vertex\* curVert = head;

        while (curVert != nullptr) {

            if (curVert->name == name) return curVert;

            if (vertPrev) {

                \*vertPrev = curVert;

            }

            curVert = curVert->next;

        }

        return nullptr;

    }

    bool findLink(vertex\* vert, vertex\* vertTarget) {

        auto links = vert->links;

        for (auto l : links) {

            if (l == vertTarget) return true;

        }

        return false;

    }

    bool findLink(char vert, char vertTarget) {

        vertex\* vertLink = findVertex(vert);

        for (auto l : vertLink->links) {

            if (l->name == vertTarget) return true;

        }

        return false;

    }

    bool addVertex(char name) {

        if (findVertex(name) != nullptr) return false;

        vertex\* newVertex = new vertex(name);

        if (isEmpty()) {

            head = newVertex;

            tail = newVertex;

        }

        else {

            tail->next = newVertex;

            tail = newVertex;

        }

        size++;

        return true;

    }

    bool delVertex(char name) {

        vertex\* vertPrev = nullptr;

        vertex\* vert = findVertex(name, &vertPrev);

        if (vert == nullptr) return false;

        if (vert == head) {

            head = vert->next;

        }

        else if (vert == tail) {

            tail = vertPrev;

            tail->next = nullptr;

        }

        else {

            vertPrev->next = vert->next;

        }

        vertex\* curVert = head;

        while (curVert != nullptr) {

            for (int i = 0; i < curVert->links.size(); i++) {

                if (curVert->links[i] == vert) {

                    curVert->links.erase(curVert->links.begin() + i); //erase link on deleted vertex

                }

            }

            curVert = curVert->next;

        }

        delete vert;

        size--;

        return true;

    }

    int addLink(char name1, char name2) {

        vertex\* vert1 = findVertex(name1);

        vertex\* vert2 = findVertex(name2);

        if (vert1 == nullptr || vert2 == nullptr) return -1; // if no vertex

        if (findLink(vert1, vert2)) return -2; // if link alredy exists

        vert1->links.push\_back(vert2);

        return 1;

    }

    void clear() {

        vertex\* currVert = head;

        while (currVert != nullptr) {

            vertex\* next = currVert->next;

            delete currVert;

            currVert = next;

        }

        head = nullptr;

        tail = nullptr;

        size = 0;

    }

};