|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** ***ИУК «Информатика и управление»***

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №3**

**«Реализация основных алгоритмов с графами»**

**ДИСЦИПЛИНА: «Типы и структуры данных»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-42Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Пчелинцева Н.И. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2022

**Цель:** формирование практических навыков создания алгоритмов обработки графов.

**Задание:**

1. Познакомиться со способами представления графов в памяти компьютера.
2. Изучить основные обходы графов.
3. Реализовать алгоритм согласно варианту.

**Вариант 7**

В графе найти максимальное расстояние между всеми парами его вершин.

**Листинг:**

***Graph.h***

#pragma once

#include <vector>

#include <iostream>

#include <iomanip>

#include <fstream>

#include <queue>

#include <string>

namespace DTAS

{

class IncorrectWeightException

{

public:

IncorrectWeightException(std::string error) : \_error(error) {}

std::string GetError() { return \_error; }

private:

std::string \_error{};

};

template <typename T>

class Graph

{

public:

Graph() : \_weights(nullptr), \_size(), \_elements(), \_outputFile("Output.txt") {}

Graph(std::string inputFile) : Graph(inputFile, "Output.txt") {}

Graph(std::string inputFile, std::string outputFile) : \_outputFile(outputFile)

{

std::ifstream file(inputFile);

file >> \_size;

for (int i{}; i < \_size; ++i)

{

T element{};

file >> element;

\_elements.push\_back(element);

}

\_weights = new int\*\* [\_size] {};

for (int i{}; i < \_size; ++i)

{

\_weights[i] = new int\* [\_size] {};

for (int j{}; j < \_size; ++j)

{

std::string weight{};

file >> weight;

if (weight == "-")

\_weights[i][j] = nullptr;

else if (std::find\_if(weight.begin(), weight.end(), [](char c) { return !std::isdigit(c); }) != weight.end())

throw IncorrectWeightException("Incorrect weight");

else

\_weights[i][j] = new int(std::stoi(weight));

}

}

file.close();

}

~Graph()

{

for (int i{}; i < \_size; ++i)

{

for (int j{}; j < \_size; ++j)

if (\_weights[i][j] != nullptr)

delete \_weights[i][j];

delete[] \_weights[i];

}

delete[] \_weights;

}

bool IsEmpty() { return \_weights == nullptr; }

void Clear()

{

if (!IsEmpty())

{

for (int i{}; i < \_size; ++i)

{

for (int j{}; j < \_size; ++j)

if (\_weights[i][j] != nullptr)

delete \_weights[i][j];

delete[] \_weights[i];

}

delete[] \_weights;

\_weights = nullptr;

\_elements.clear();

\_size = 0;

}

}

void Print(int width)

{

if (IsEmpty())

std::cout << "Graph is empty\n";

else

{

std::cout << std::setw(width) << " ";

for (int i{}; i < \_size; ++i)

std::cout << std::setw(width) << \_elements[i];

std::cout << "\n";

for (int i{}; i < \_size; ++i)

{

std::cout << std::setw(width) << \_elements[i];

for (int j{}; j < \_size; ++j)

if (\_weights[i][j] != nullptr)

std::cout << std::setw(width) << \*\_weights[i][j];

else

std::cout << std::setw(width) << "-";

std::cout << "\n";

}

}

}

void Add(T element)

{

int\*\*\* temp = new int\*\* [\_size + 1]{};

for (int j{}; j < \_size + 1; ++j)

temp[j] = new int\* [\_size + 1]{};

for (int j{}; j < \_size; ++j)

for (int k{}; k < \_size; ++k)

if (\_weights[j][k] != nullptr)

temp[j][k] = new int(\*\_weights[j][k]);

for (int i{}; i < \_size; ++i)

{

for (int j{}; j < \_size; ++j)

if (\_weights[i][j] != nullptr)

delete \_weights[i][j];

delete[] \_weights[i];

}

delete[] \_weights;

\_weights = temp;

\_elements.push\_back(element);

++\_size;

}

void Connect(T source, T destination, std::string weight)

{

int s;

for (s = 0; s < \_size; ++s)

if (\_elements[s] == source)

break;

int d;

if (source == destination)

d = s;

else

{

for (d = 0; d < \_size; ++d)

if (\_elements[d] == destination)

break;

}

if (s < \_size && d < \_size)

{

delete \_weights[s][d];

if (weight == "-")

\_weights[s][d] = nullptr;

else if (std::find\_if(weight.begin(), weight.end(), [](char c) { return !std::isdigit(c); }) != weight.end())

throw IncorrectWeightException("Incorrect weight");

else

\_weights[s][d] = new int(std::stoi(weight));

}

}

void Remove(T element)

{

int i;

for (i = 0; i < \_size; ++i)

if (\_elements[i] == element)

break;

if (i < \_size)

{

int\*\*\* temp = nullptr;

if (\_size > 1)

{

temp = new int\*\* [\_size - 1]{};

for (int j{}; j < \_size - 1; ++j)

temp[j] = new int\* [\_size - 1]{};

for (int j{}; j < i; ++j)

{

for (int k{}; k < i; ++k)

if (\_weights[j][k] != nullptr)

temp[j][k] = new int(\*\_weights[j][k]);

for (int k = i + 1; k < \_size; ++k)

if (\_weights[j][k] != nullptr)

temp[j][k - 1] = new int(\*\_weights[j][k]);

}

for (int j = i + 1; j < \_size; ++j)

{

for (int k{}; k < i; ++k)

if (\_weights[j][k] != nullptr)

temp[j - 1][k] = new int(\*\_weights[j][k]);

for (int k = i + 1; k < \_size; ++k)

if (\_weights[j][k] != nullptr)

temp[j - 1][k - 1] = new int(\*\_weights[j][k]);

}

}

for (int j{}; j < \_size; ++j)

{

for (int k{}; k < \_size; ++k)

if (\_weights[j][k] != nullptr)

delete \_weights[j][k];

delete[] \_weights[j];

}

delete[] \_weights;

\_weights = temp;

\_elements.erase(\_elements.begin() + i);

--\_size;

}

}

void DFS(T start)

{

if (IsEmpty())

std::cout << "Graph is empty\n";

else

{

int i;

for (i = 0; i < \_size; ++i)

if (\_elements[i] == start)

break;

if (i < \_size)

{

bool\* visited = new bool[\_size] {};

DFS(i, visited);

delete[] visited;

}

}

}

void BFS(T start)

{

if (IsEmpty())

std::cout << "Graph is empty\n";

else

{

int index;

for (index = 0; index < \_size; ++index)

if (\_elements[index] == start)

break;

if (index < \_size)

{

bool\* visited = new bool[\_size] {};

std::queue<int> waiting{};

waiting.push(index);

visited[index] = true;

while (!waiting.empty())

{

index = waiting.front();

waiting.pop();

std::cout << \_elements[index] << " ";

std::vector<int> unsorted{};

for (int i{}; i < \_size; ++i)

if ((\_weights[index][i] != nullptr) && (!visited[i]))

{

unsorted.push\_back(i);

visited[i] = true;

}

std::sort(unsorted.begin(), unsorted.end(), [&](int a, int b) { return \*\_weights[index][a] < \*\_weights[index][b]; });

for (int i{}; i < unsorted.size(); ++i)

waiting.push(unsorted[i]);

}

delete[] visited;

}

}

}

void PrintMaxDistances(int width)

{

std::ofstream file(\_outputFile);

if (IsEmpty())

{

std::cout << "Graph is empty\n";

file << "Graph is empty";

}

else

{

int\*\*\* maxDistances = new int\*\* [\_size] {};

for (int i{}; i < \_size; ++i)

maxDistances[i] = new int\* [\_size] {};

for (int i{}; i < \_size; ++i)

for (int j{}; j < \_size; ++j)

{

std::vector<Pair> passed{};

maxDistances[i][j] = FindMaxDistance(i, j, passed);

}

std::cout << std::setw(width) << " ";

file << std::setw(width) << " ";

for (int i{}; i < \_size; ++i)

{

std::cout << std::setw(width) << \_elements[i];

file << std::setw(width) << \_elements[i];

}

std::cout << "\n";

file << "\n";

for (int i{}; i < \_size; ++i)

{

std::cout << std::setw(width) << \_elements[i];

file << std::setw(width) << \_elements[i];

for (int j{}; j < \_size; ++j)

if (maxDistances[i][j] != nullptr)

{

std::cout << std::setw(width) << \*maxDistances[i][j];

file << std::setw(width) << \*maxDistances[i][j];

}

else

{

std::cout << std::setw(width) << "-";

file << std::setw(width) << "-";

}

std::cout << "\n";

file << "\n";

}

for (int i{}; i < \_size; ++i)

{

for (int j{}; j < \_size; ++j)

if (maxDistances[i][j] != nullptr)

delete maxDistances[i][j];

delete[] maxDistances[i];

}

delete[] maxDistances;

}

file.close();

}

private:

void DFS(int index, bool\* visited)

{

std::cout << \_elements[index] << " ";

visited[index] = true;

for (int i{}; i < \_size; ++i)

if ((\_weights[index][i] != nullptr) && (!visited[i]))

DFS(i, visited);

}

struct Pair

{

int i{};

int j{};

};

int\* FindMaxDistance(int source, int destination, std::vector<Pair> passed)

{

if (source == destination)

return new int(0);

int\* maxDistance = nullptr;

for (int i{}; i < \_size; ++i)

{

Pair pair{ source, i };

if (\_weights[source][i] != nullptr && std::find\_if(passed.begin(), passed.end(),

[&](Pair a) { return a.i == pair.i && a.j == pair.j || a.i == pair.j && a.j == pair.i; }) == passed.end())

{

passed.push\_back(pair);

int\* distance = FindMaxDistance(i, destination, passed);

passed.pop\_back();

if (distance != nullptr)

{

\*distance += \*\_weights[source][i];

if (maxDistance == nullptr)

maxDistance = distance;

else if (\*distance > \*maxDistance)

{

delete maxDistance;

maxDistance = distance;

}

}

}

}

return maxDistance;

}

int\*\*\* \_weights{};

uint16\_t \_size{};

std::vector<T> \_elements{};

std::string \_outputFile{};

};

}

***Menu.h***

#pragma once

#include "Graph.h"

namespace DTAS

{

class Menu

{

public:

Menu();

Menu(std::string inputFile);

Menu(std::string inputFile, std::string outputFile);

Menu(std::string inputFile, std::string outputFile, std::string logFile);

~Menu();

void Run();

private:

void Pause();

Graph<std::string> \_graph{};

std::string \_logFile{};

};

}

***Menu.cpp***

#include "Menu.h"

namespace DTAS

{

Menu::Menu() : \_graph() {}

Menu::Menu(std::string inputFile) : \_graph(inputFile), \_logFile("Log.txt") {}

Menu::Menu(std::string inputFile, std::string outputFile) : Menu(inputFile, outputFile, "Log.txt") {}

Menu::Menu(std::string inputFile, std::string outputFile, std::string logFile) : \_graph(inputFile, outputFile), \_logFile(logFile) {}

Menu::~Menu() { remove("pauseTemp"); }

void Menu::Run()

{

int command = 1;

while (command != 0)

{

system("cls");

\_graph.Print(3);

std::cout << "\n1. Add element\n";

std::cout << "2. Change connection\n";

std::cout << "3. Remove element\n";

std::cout << "4. Clear\n";

std::cout << "\n5. Depth-first search\n";

std::cout << "6. Breadth-first search\n";

std::cout << "7. Show max distances\n";

std::cout << "\n0. Exit\n";

std::cout << ">>> ";

std::cin >> command;

std::cout << "\n";

std::string input{};

std::string source{};

std::string destination{};

switch (command)

{

case 1:

std::cout << "Input new element: ";

std::cin >> input;

\_graph.Add(input);

break;

case 2:

std::cout << "Input source: ";

std::cin >> source;

std::cout << "Input destination: ";

std::cin >> destination;

std::cout << "Input new weight or '-' to break the connection: ";

std::cin >> input;

try

{

\_graph.Connect(source, destination, input);

}

catch (IncorrectWeightException& error)

{

std::ofstream file(\_logFile, std::ios\_base::app);

std::cout << error.GetError() << "\n";

file << error.GetError() << "\n";

file.close();

Pause();

}

break;

case 3:

std::cout << "Input element to remove: ";

std::cin >> input;

\_graph.Remove(input);

break;

case 4:

\_graph.Clear();

break;

case 5:

std::cout << "Input start element: ";

std::cin >> input;

std::cout << "DFS: ";

\_graph.DFS(input);

Pause();

break;

case 6:

std::cout << "Input start element: ";

std::cin >> input;

std::cout << "BFS: ";

\_graph.BFS(input);

Pause();

break;

case 7:

std::cout << "Max distances:\n";

\_graph.PrintMaxDistances(3);

Pause();

break;

}

}

}

void Menu::Pause()

{

system("pause>pauseTemp");

remove("pauseTemp");

}

}

***Main.cpp***

#include "Menu.h"

int main(int argc, char\* argv[])

{

if (argc == 1)

{

DTAS::Menu menu{};

menu.Run();

}

else

{

try

{

if (argc == 2)

{

DTAS::Menu menu(argv[1]);

menu.Run();

}

else if (argc == 3)

{

DTAS::Menu menu(argv[1], argv[2]);

menu.Run();

}

else if (argc == 4)

{

DTAS::Menu menu(argv[1], argv[2], argv[3]);

menu.Run();

}

}

catch (DTAS::IncorrectWeightException& error)

{

std::ofstream file("Log.txt", std::ios\_base::app);

std::cout << error.GetError() << "\n";

file << error.GetError() << "\n";

file.close();

}

}

return 0;

}

**Результат:**

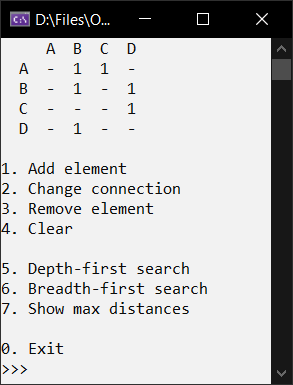


Рис. 1. Основное меню

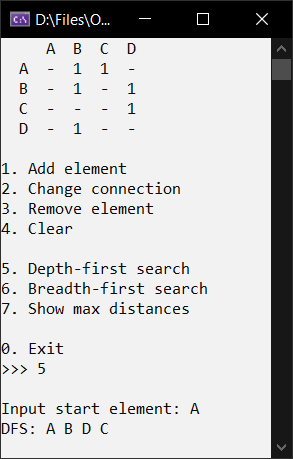
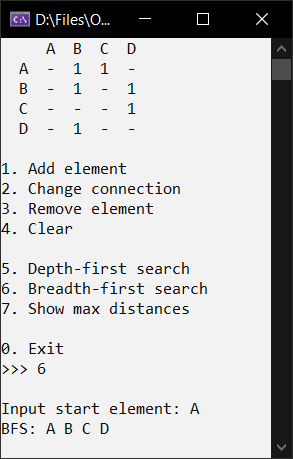
 

Рис. 2. Обходы

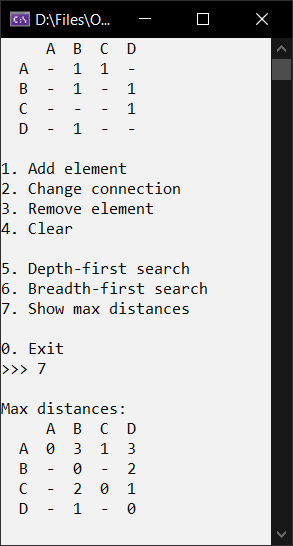


Рис. 3. Наибольшие расстояния

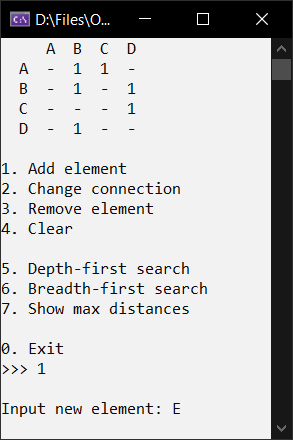
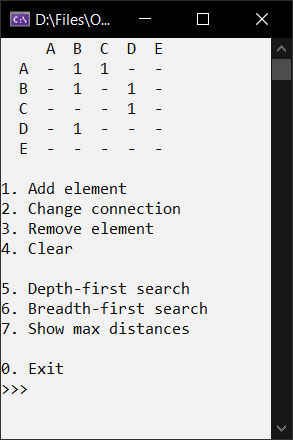
 

Рис. 4. Вставка элемента

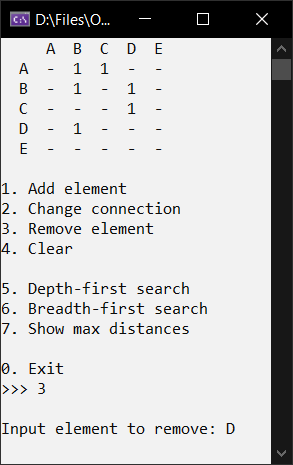
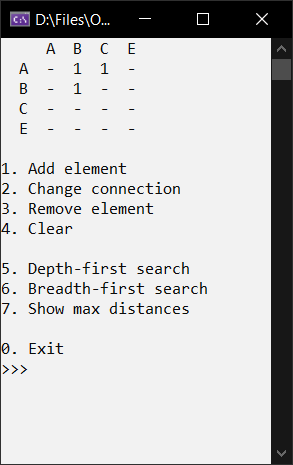
 

Рис. 5. Удаление элемента

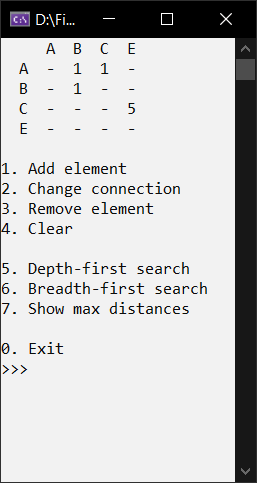
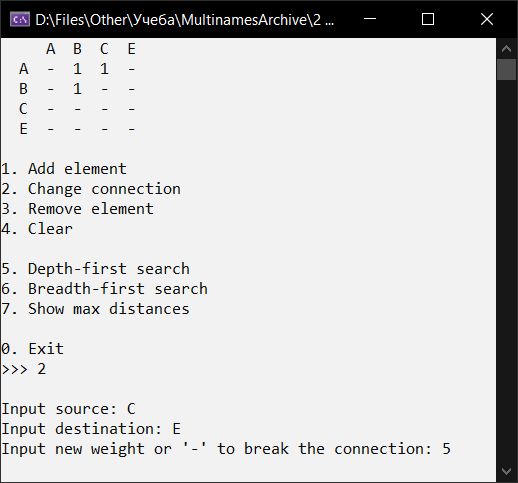


Рис. 6. Изменение связи

**Вывод:** в ходе выполнения лабораторной работы были получены навыки создания графа, реализации добавления и удаления элемента, изменения связей, обходов в глубину и ширину, нахождения наибольших расстояний.