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. ИУК4-32Б Петроченков (Петроченков И. А.)
(Подпись) (Ф.И.О.)

Проверил(а): (Пчелинцева Н. И.)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка: 95
- Оценка: отлично

Калуга, 2023 г.

Scanned with AnyScanner

Цели работы:

Целью выполнения лабораторной работы является формирование практических навыков создания алгоритмов обработки графов.

Задачи:

Основными задачами выполнения лабораторной работы являются:

1. Познакомиться со способами представления графов в памяти компьютера.
2. Изучить основные обходы графов.
3. Научиться составлять алгоритмы для нахождения кратчайших путей в графе. Реализовать алгоритм согласно варианту.

Вариант №35

Создать программу согласно варианту, полученному у преподавателя. При выполнении лабораторной работы запрещается использовать сторонние классы и компоненты, реализующие заявленную функциональность. Для заданного графа, используя метод поиска в глубину, определить кратчайшее расстояние из города A в город B.

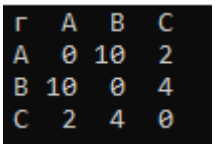

Рис. 1 Печать матрицы смежностей

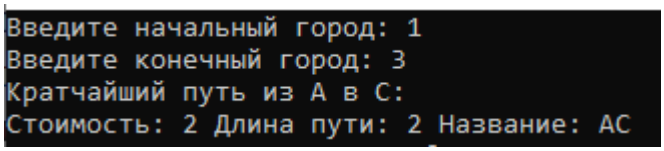

Рис. 2 Выполнение индивидуального задания

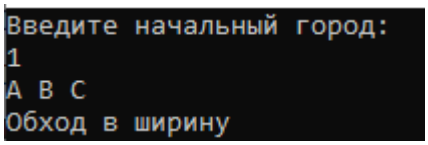

Рис. 3 Демонстрация обхода в ширину

```
Введите начальный город:
1
Маршрут из A в A = 0
Маршрут из A в B = 6
Маршрут из A в C = 2
```

Рис. 4 Алгоритм Деикстры

```
Петроченков И. А. ИУК4-32Б
  [1] Получить размерность матрицы соответствий
  [2] Обновить размерность матрицы соответствий
  [3] Напечатать матрицу соответствий
  [4] Установить ребро
  [5] Обход методом поиска в ширину
  [6] Добавить вершину
  [7] Установить матрицу соответствий
  [8] Поиск наименьшего пути алгоритмом Деикстры
> [9] Поиск наименьшего пути методом поиска в глубину
  [10] Запись в файл матрицы соответствий
  [11] Чтение из файла матрицы соответствий
  [12] Обход графа в глубину
  [0] Выход
```

Рис. 5 Меню программы

Вывод:

Результатами работы являются:

- Программа, реализующая основные действия с графами;
- Подготовленный отчет;

Листинг программы:

Exceptions.cpp

```
#include "Exceptions.h"
```

Graph.cpp

```
#include "Graph.h"
```

lab3.cpp

```
#include <iostream>
#include "Graph.h"
#include <cstdio>
//#include <windows.h>
#include "Addition.h"
#include "Menu.h"
using namespace std;
int main(int argc, char** argv)
{
	setlocale(0, "");
	if (Start(argc, argv))
	{
		Menu menu = Menu("Петроченков И. А. ИУК4-32Б");
		menu.AddItem("Получить размерность матрицы соответствий", GetGraphDim);
		menu.AddItem("Обновить размерность матрицы соответствий", Update);
		menu.AddItem("Напечатать матрицу соответствий", Print);
		menu.AddItem("Установить ребро", SetEdge);
```

```
            menu.AddItem("Обход методом поиска в ширину", BFS);
            menu.AddItem("Добавить вершину", AddVertex);
            menu.AddItem("Установить матрицу соответствий", SetMatrix);
            menu.AddItem("Поиск наименьшего пути алгоритмом Деикстры", Deikstra);
            menu.AddItem("Поиск наименьшего пути методом поиска в глубину", DS);
            menu.AddItem("Запись в файл матрицы соответствий", FileWrite);
            menu.AddItem("Чтение из файла матрицы соответствий", ReadFromFile);
            menu.AddItem("Обход графа в глубину", DFS);
            menu.RunMenu();
      }
}
```

Menu.cpp

MenuItem.cpp

Addition.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <random>
#include <string>
std::string exc = "exception.txt";
//std::string src = "alph.txt";
std::string dst = "destination.txt";
std::string src = "source.txt";
#include <Windows.h>
#include "Graph.h"

Graph graph = Graph();

bool InputError()
{
      std::cin.clear();
      std::cin.ignore(INT_MAX, '\n');
      std::cout << "\nОшибка! Введено некорректное значение!\n Попробуйте сно-
ва.\n";
      return false;
}

bool Start(int argc, char* argv[])
{
      std::string message = "Параметры запуска программы:\nsource.txt – обязатель-
ный параметр, место хранения словаря\n";
      if (argc <= 1)
      {
            //std::cout << argc << std::endl;
            std::cout << message;
            return false;
      }
      else
      {
            if (argc == 2)
            {
                  //std::cout << argc << std::endl;
                  dst = argv[1];
                  return true;
            }
            else if (argc > 2)
            {
                  //std::cout << argc << std::endl;
                  std::cout << message << std::endl;
```

```
                return false;
            }
        }
}

void GetGraphDim()
{
        try
        {
            std::cout << "Размерность матрицы соответствий: " << graph.GetDim() <<
std::endl;

        }
        catch (...)
        {
            std::cout << "Произошла ошибка!" << std::endl;
        }
}

void Update()
{
        try
        {
            std::cout << "Введите размерность матрицы: " << std::endl;
            int dim = 0;
            std::cin >> dim;
            graph.Update(dim);
            std::cout << "Таблица соответствий обновлена!" << std::endl;
        }
        catch (...)
        {
            std::cout << "Произошла ошибка обновления матрицы!" << std::endl;
        }
}
void Print()
{
        try
        {
            graph.Print();
        }
        catch (...)
        {
            std::cout << "Матрица пуста!" << std::endl;
        }
}

void SetEdge()
{
        try
        {
            std::cout << "Введите начальный и конечный города и стоимость пути: "
<< std::endl;
            int first = 0, second = 0, val = 0;
            std::cin >> first >> second >> val;
            graph.SetEdge(first, second, val);
            std::cout << "Ребро установлено!" << std::endl;
        }
        catch (...)
        {
            std::cout << "Произошла ошибка установки ребра!" << std::endl;
        }
}
void BFS()
{
        try
```

```
	{
		std::cout << "Введите начальный город: " << std::endl;
		int begin = 0;
		std::cin >> begin;
		graph.BFS(begin);
		std::cout << "Обход в ширину" << std::endl;
	}
	catch (...)
	{
		std::cout << "Произошла ошибка алгоритма поиска в ширину!" <<
std::endl;
	}
}

void AddVertex()
{
	try
	{
		graph.AddVertex();
		std::cout << "Вершина добавлена!" << std::endl;

	}
	catch (...)
	{
		std::cout << "Произошла ошибка добавления вершины!" << std::endl;
	}
}

void SetMatrix()
{
	try
	{
		graph.SetMatrix();
		std::cout << "Матрица установлена!" << std::endl;

	}
	catch (...)
	{
		std::cout << "Произошла ошибка установки матрицы соответствий!" <<
std::endl;
	}
}

void Deikstra()
{
	try
	{
		std::cout << "Введите начальный город: " << std::endl;
		int begin = 0;
		std::cin >> begin;
		graph.Deikstra(begin);

	}
	catch (...)
	{
		std::cout << "Произошла ошибка алгоритмы Деикстры!" << std::endl;
	}
}

void DS()
{
	try
	{
		graph.NewDS();
```

```
	}
	catch (...)
	{
		std::cout << "Произошла ошибка поиска в глубину!" << std::endl;
	}
}

void FileWrite()
{
	try
	{
		std::cout << "Введите путь файла: " << std::endl;
		std::string path = "";
		std::cin >> path;
		graph.WriteInFile(path.c_str());
	}
	catch (...)
	{
		std::cout << "Произошла ошибка записи в файл!" << std::endl;
	}
}

void ReadFromFile()
{
	try
	{
		graph.ReadFromFile(src.c_str());
		std::cout << "Считывание произошло успешно!" << std::endl;
	}
	catch (...)
	{
		std::cout << "Ошибка считывания файла" << std::endl;
	}
}

void DFS()
{
	try
	{
		graph.DoDFS();
		std::cout << "Обход графа в глубину" << std::endl;
	}
	catch (...)
	{
		std::cout << "Ошибка обхода" << std::endl;
	}
}
```

Exceptions.h

```
#pragma once
#include <exception>
#include <iostream>
#include <fstream>
#include <typeinfo>
#include <chrono>
#include <ctime>

#pragma warning(disable : 4996)
class BaseE : std::exception
{
protected:
	const char* message = "";
public:
	BaseE()
```

```
	{
	}
	BaseE(const char* msg)
	{
		try {


			std::ofstream fout;
			fout.open("exception.txt", std::ios_base::app);
			if (fout.is_open()) {
				fout << "\n";
				std::time_t end_time =
std::chrono::system_clock::to_time_t(std::chrono::system_clock::now());
				fout << msg << ' ' << ctime(&end_time);
			}
			fout.close();
		}
		catch (...) {

		}
		message = msg;
	}
	virtual const char* what() const
	{
		return message;
	}
};
class OutOfRangeE : BaseE
{
private:
public:
	OutOfRangeE()
	{
	}
	OutOfRangeE(const char* msg) : BaseE(msg)
	{

	}
	virtual const char* what() const override
	{
		return BaseE(message).what();
	};
};
template <class T>
class TypeErrorE : BaseE
{
private:
	const char* targetType = typeid(T).name();
public:
	TypeErrorE()
	{
		targetType = typeid(T).name();
	}
	TypeErrorE(const char* msg) : BaseE(msg)
	{
		targetType = typeid(T).name();
	}
	const char* GetTargetType() const
	{
		return targetType;
	}
	virtual const char* what() const override
	{
		return BaseE(message).what();
	};
```

```
};

class InputErrorE : BaseE
{
private:
public:
	InputErrorE()
	{
	}
	InputErrorE(const char* msg) : BaseE(msg)
	{
	}
	virtual const char* what() const override
	{
		return BaseE(message).what();
	};
};

class NodeE : BaseE
{
private:
public:
	NodeE()
	{
	}
	NodeE(const char* msg) : BaseE(msg)
	{
	}
	virtual const char* what() const override
	{
		return BaseE(message).what();
	};
};
```

Graph.h

```
#pragma once
#include <iostream>
#include <iomanip>
#include <C:\Users\Игорь\Desktop\учеба\LabRab\ЛР ВП 2 Сем\1
ЛР\LR1\LR1\myException.h>
#include "C:\Users\Игорь\Desktop\учеба\LabRab\ЛР ВП 2 Сем\1 ЛР\LR1\LR1\myVector.h"
#include "Exceptions.h"
#include <fstream>

struct RouteParams
{
public:
	int cost = 0;
	int len = 0;
	//char* pathName = nullptr;
	PIA::myVector<char> name = PIA::myVector<char>();
	RouteParams()
	{
		cost = 0;
		len = 0;
		name.clear();
	}

	RouteParams(int cost, int len, PIA::myVector<char> name)
	{
		this->cost = cost;
		this->len = len;
		this->name = name;
	}
```

```
        RouteParams(const RouteParams& other)
        {
                this->cost = other.cost;
                this->len = other.len;
                this->name = other.name;
        }
        RouteParams& operator=(const RouteParams& other)
        {
                this->cost = other.cost;
                this->len = other.len;
                this->name = other.name;
                return *this;
        }

        RouteParams Copy()
        {
                RouteParams result = RouteParams();
                result.cost = this->cost;
                result.len = this->len;
                result.name = this->name;
                return result;
        }
        /*~RouteParams()
        {
                delete[] pathName;
        }*/
};

struct Routes
{
public:
        RouteParams* routes = nullptr;
        int len = 0;
        Routes()
        {
                len = 0;
                routes = nullptr;
        }
        Routes& operator=(const Routes& other)
        {
                this->len = other.len;
                if (this->routes) delete[] this->routes;
                this->routes = new RouteParams[this->len];
                for (int i = 0; i < other.len; i++)
                {
                        this->routes[i] = other.routes[i];
                }
                return *this;
        }
        ~Routes()
        {
                if(this->routes) delete[] routes;
        }
};

class Graph
{
private:
        int** adjacency = nullptr;
        int dim = 0;
public:
        Graph()
        {
                this->adjacency = 0;
```

```
		this->dim = 0;
	}
	Graph(int dim)
	{
		this->dim = dim;
		if (this->adjacency)
		{
			for (int i = 0; i < dim; i++)
			{
				if (adjacency[i]) delete[] adjacency[i];
			}
		}
		this->adjacency = new int* [dim];
		for (int i = 0; i < dim; i++)
		{
			this->adjacency[i] = new int[dim];
		}
		for (int i = 0; i < dim; i++)
		{
			for (int j = 0; j < dim; j++)
			{
				this->adjacency[i][j] = 0;
			}
		}
	}

	int GetDim() const
	{
		return this->dim;
	}

	void Update(int dim)
	{
		if (dim <= 0) throw new OutOfRangeE("Matrix dimension can't be lower
than one");
		if (this->adjacency)
		{
			for (int i = 0; i < this->dim; i++)
			{
				if (adjacency[i]) delete[] adjacency[i];
			}
		}
		this->dim = dim;
		this->adjacency = new int* [this->dim];
		for (int i = 0; i < this->dim; i++)
		{
			this->adjacency[i] = new int[this->dim];
		}
		for (int i = 0; i < this->dim; i++)
		{
			for (int j = 0; j < this->dim; j++)
			{
				this->adjacency[i][j] = 0;
			}
		}
	}

	int& operator()(int i, int j)
	{
		if (i < 0 || j < 0) throw new OutOfRangeE("Index can't be lower than
zero");
		if (i > this->dim - 1 || j > this->dim - 1) throw new OutOfRang-
eE("Index can't be greater than matrix dimension");
		return this->adjacency[i][j];
	}
```

```
	void Print() const
	{
		if (!this->adjacency) throw new BaseE("The graph is empty");
		for (int i = 0; i < dim + 1; i++)
		{
			for (int j = 0; j < dim + 1; j++)
			{
				if (i == 0 && j == 0)
				{
					std::cout << std::setw(3) << 'r';
				}
				else if (i == 0 || j == 0)
				{
					std::cout << std::setw(3) << (char)('A' - 1 + i +
j);
				}
				else
				{
					std::cout << std::setw(3) << adjacency[i-1][j-1];
				}
			}
			std::cout << std::endl;
		}
	}

	void SetEdge(int first, int second, int value)
	{
		if(first < 0 || second < 0) throw new OutOfRangeE("Index can't be lower
than zero");
		if(first > this->dim-1 || second > this->dim - 1) throw new OutOfRang-
eE("Index can't be greater than matrix dimension");
		if (first == second) throw new BaseE("Can't change distance");
		if(first < this->dim && second < this->dim)
		this->adjacency[first - 1][second - 1] = value;
		this->adjacency[second - 1][first - 1] = value;
	}


void BFS(int begin) const
{
	if(begin < 1) throw new OutOfRangeE("Index can't be lower than one");
	if (begin > this->dim) throw new OutOfRangeE("Index can't be greater than ma-
trix dimension");
	int unit = begin - 1;
	bool* visited = new bool[this->dim];
	for (int i = 0; i < this->dim; i++)
	{
		visited[i] = false;
	}
	int* queue = new int[this->dim];
	int count = 0;
	int head = 0;
	for (int i = 0; i < this->dim; i++)
	{
		queue[i] = 0;
	}
	queue[count++] = unit;
	visited[unit] = true;
	while (head < count)
	{
		unit = queue[head++];
		std::cout << (char)(unit + 'A') << ' ';
		for (int i = 0; i < this->dim; i++)
		{
```

```
                        if (adjacency[unit][i] && !visited[i])
                        {
                                queue[count++] = i;
                                visited[i] = true;
                        }
                }
        }
        std::cout << std::endl;
        delete[] queue;
}

void AddVertex()
{
        int** temp = new int* [this->dim+1];
        for (int i = 0; i< this->dim +1; i++)
        {
                temp[i] = new int[this->dim +1];
        }
        for (int i = 0; i < this->dim; i++)
        {
                for (int j = 0; j < this->dim; j++)
                {
                        temp[i][j] = adjacency[i][j];
                }
        }
        //Update(this->dim + 1);
        this->dim += 1;
        for (int i = 0; i < dim - 1; i++)
        {
                std::cout << "Введите вес между городами " << (char)(dim - 1 + 'A') <<
" и " << (char)(i + 'A') << ": ";
                int val = 0;
                while (!(std::cin >> val))
                {
                        std::cin.clear();
                        std::cin.ignore(INT_MAX, '\n');
                        std::cout << "\nОшибка! Введено некорректное значение!\n Попро-
буйте снова.\n";
                }
                temp[dim - 1][i] = val;
                temp[i][dim - 1] = val;
                //temp[dim][i] = val;
                //temp[i][dim] = val;
        }
        temp[dim - 1][dim-1] = 0;

        if (this->adjacency)
        {
                for (int i = 0; i < this->dim - 1; i++)
                {
                        if (this->adjacency[i]) delete[] this->adjacency[i];
                }
        }
        this->adjacency = temp;
}

void SetMatrix()
{
        int bias = 0;
        for (int i = 0; i < dim; i++)
        {
                adjacency[i][i] = 0;
        }
        for (int i = 0; i < dim - 1; i++)
        {
```

```
            for (int j = i + 1; j < dim; j++)
            {
                  std::cout << "Введите вес между городами " << (char)(i + 'A') <<
" и " << (char)(j + 'A') << ": ";
                  int val = 0;
                  while (!(std::cin >> val))
                  {
                        std::cin.clear();
                        std::cin.ignore(INT_MAX, '\n');
                        std::cout << "\nОшибка! Введено некорректное значение!\n
Попробуйте снова.\n";
                  }
                  adjacency[i][j] = val;
                  adjacency[j][i] = val;
            }
      }
}

void Deikstra(int start) const
{
      if(start < 0) throw new OutOfRangeE("Index can't be lower than zero");
      if(start > this->dim) throw new OutOfRangeE("Index can't be greater than ma-
trix dimension");
      int begin = start - 1;
      PIA::myVector<int> distance = PIA::myVector<int>();
      int count = 0, index = 0, u = 0;// , m = begin + 1;
      bool* visited = new bool[this->dim];
      for (int i = 0; i < this->dim; i++)
      {
            distance.pushBack(INT_MAX);
            visited[i] = false;
      }
      distance[begin] = 0;
      for (count = 0; count < this->dim - 1; count++)
      {
            int min = INT_MAX;
            for (int i = 0; i < this->dim; i++)
            {
                  if (!visited[i] && distance[i] <= min)
                  {
                        min = distance[i];
                        index = i;
                  }
            }
            u = index;
            visited[u] = true;
            for (int i = 0; i < this->dim; i++)
            {
                  if (!visited[i] && this->adjacency[u][i] && distance[u] !=
INT_MAX &&
                        distance[u] + adjacency[u][i] < distance[i])
                        distance[i] = distance[u] + adjacency[u][i];
            }
      }
      for (int i = 0; i < this->dim; i++)
      {
            if (distance[i] != INT_MAX)
                  std::cout << "Маршрут из " << (char)(begin + 'A') << " в " <<
(char)('A' + (i) % (this->dim)) << " = " << distance[i] << std::endl;
            else std::cout << "Маршрут из " << (char)(begin + 'A') << " в " <<
(char)('A' + (i ) % (this->dim)) << " = " << "маршрут недоступен" << std::endl;
      }
      delete[] visited;
}
void NewDS() const
```

```
{
	int begin = 0;
	std::cout << "Введите начальный город: ";
	while (!(std::cin >> begin) || begin < 0 || begin > this->dim)
	{
		std::cin.clear();
		std::cin.ignore(INT_MAX, '\n');
		std::cout << "\nОшибка! Введено некорректное значение!\n Попробуйте
снова.\n";
	}
	int end = 0;
	std::cout << "Введите конечный город: ";
	while (!(std::cin >> end) || end < 0 || end > this->dim)
	{
		std::cin.clear();
		std::cin.ignore(INT_MAX, '\n');
		std::cout << "\nОшибка! Введено некорректное значение!\n Попробуйте
снова.\n";
	}
	PIA::myVector<bool> visited = PIA::myVector<bool>();
	for (int i = 0; i < this->dim; i++)
	{
		visited.pushBack(false);
	}
	PIA::myVector<RouteParams> params = PIA::myVector<RouteParams>();
	RouteParams base = RouteParams(0, 0, PIA::myVector<char>());
	NewDepthSearch(begin - 1, end - 1, visited, params, base);
	auto shortest = RouteParams(INT_MAX, INT_MAX, PIA::myVector<char>());
	for (auto el : params)
	{
		/*std::cout << "cost: " << el.cost << " len: " << el.len << " name: ";
		for (auto& e : el.name)
		{
			std::cout << e;
		}
		std::cout << std::endl;*/
		if (el.name[el.len - 1] == (char)('A' + end - 1))
		{
			if (el.cost < shortest.cost)
			{
				shortest = el;
			}
		}
	}
	if (shortest.cost != INT_MAX && shortest.len != INT_MAX)
	{
		std::cout << "Кратчайший путь из " << (char)('A' + begin - 1) << " в "
<< (char)('A' + end - 1) << ": " << std::endl;
		std::cout << "Стоимость: " << shortest.cost << " Длина пути: " <<
shortest.len << " Название: ";
		for (auto& e : shortest.name)
		{
			std::cout << e;
		}
		std::cout << std::endl;
	}
	else
	{
		std::cout << "Из города " << (char)('A' + begin - 1) << " в " <<
(char)('A' + end - 1) << " добраться нельзя" << std::endl;
	}
}
void NewDepthSearch(int start, int end, PIA::myVector<bool> visited,
PIA::myVector<RouteParams>& params, RouteParams base) const
```

```
{
	try
	{
		visited[start] = true;
		if (start == end)
		{
			auto name = PIA::myVector<char>(base.name);
			name.pushBack((char)('A' + start));
			RouteParams nbase = RouteParams(base.cost + adjacen-
cy[start][start], base.len + 1, name);
			params.pushBack(nbase);
			return;
		}
		else
		{
			for (int i = 0; i < this->dim; i++)
			{
				if (adjacency[start][i] != 0 && !visited[i])
				{
					auto name = PIA::myVector<char>(base.name);
					name.pushBack((char)('A' + start));
					RouteParams nbase = RouteParams(base.cost + adjacen-
cy[start][i], base.len + 1, name);//base.name + (char)('A' + start));
					params.pushBack(nbase);
					NewDepthSearch(i, end, visited, params, nbase);
				}
			}
		}
	}
	catch (std::exception& ex)
	{
		std::cout << ex.what() << std::endl;
	}
}

void WriteInFile(const char* path) const
{
	try
	{
		std::ofstream fout;
		fout.open(path);
		if (!this->adjacency) throw new BaseE("The graph is empty");
		for (int i = 0; i < dim + 1; i++)
		{
			for (int j = 0; j < dim + 1; j++)
			{
				if (i == 0 && j == 0)
				{
					fout << std::setw(3) << 'r';
				}
				else if (i == 0 || j == 0)
				{
					fout << std::setw(3) << (char)('A' - 1 + i + j);
				}
				else
				{
					fout << std::setw(3) << adjacency[i - 1][j - 1];
				}
			}
			fout << std::endl;
		}
		fout.close();
	}
	catch (std::exception& ex)
	{
```

```
			std::cout << ex.what() << std::endl;
			std::cout << "Произошла ошибка записи в файл!" << std::endl;
		}
}
void Clear()
{
		if (this->adjacency)
		{
			for (int i = 0; i < this->dim; i++)
			{
				if (this->adjacency[i]) delete adjacency[i];
			}
		}
		this->dim = 0;
}

void ReadFromFile(const char* path)
{
		try
		{
			this->Clear();
			std::ifstream fin;
			fin.open(path);
			fin >> this->dim;
			this->Update(this->dim);
			for (int i = 0; i < this->dim; i++)
			{
				for (int j = 0; j < this->dim; j++)
				{
					fin >> this->adjacency[i][j];
				}
			}
			fin.close();
		}
		catch (...)
		{
			std::cout << "Произошла ошибка считывания файла" << std::endl;
		}
}

void DoDFS() const
{
		std::cout << "Введите начальный город" << std::endl;
		int begin = 0;
		while (!(std::cin >> begin) || begin < 0 || begin > this->dim)
		{
			std::cin.clear();
			std::cin.ignore(INT_MAX, '\n');
			std::cout << "\nОшибка! Введено некорректное значение!\n Попробуйте
снова.\n";
		}
		bool* visited = new bool[this->dim];
		for (int i = 0; i < this->dim; i++)
		{
			visited[i] = false;
		}
		DFS(visited, begin - 1);
		delete[] visited;
		std::cout << std::endl;
}

void DFS(bool* visited, int begin) const
{
		std::cout << (char)('A' + begin);
		visited[begin] = true;
```

```
	for (int i = 0; i < this->dim; i++)
	{
		if (this->adjacency[begin][i] != 0 && !visited[i])
		{
			DFS(visited, i);
		}
	}
}
/*void PrintList(bool* visited)
{
	for (int i = 0; i < dim; i++)
	{
		std::cout << visited[i] << ' ';
	}
	std::cout << std::endl;
}*/
/*void DoDS()
{
	bool* visited = new bool[dim] {};
	int begin = 0;
	std::cout << "Введите начальный город: ";
	std::cin >> begin;
	int end = 0;
	std::cout << "Введите конечный город: ";
	std::cin >> end;
	bool flag = false;
	int* summ = new int[dim];
	DepthSearch(begin - 1, end - 1, visited, flag, summ);
	int len = 0;
	for (int i = 0; i < dim; i++)
	{
		if (visited[i] == true) len++;
	}
	if (len == 1)
	{
		std::cout << "Нет способа добраться из вершины " << begin << " до вер-
шины " << end << std::endl;
	}
	else
	{
		std::cout << std::endl << "Стоимость: " << summ << std::endl << "Длина:
" << len << std::endl;
	}
	delete[] visited;
}*/
/*void DepthSearch(int begin, int end, bool* visited, bool& flag, int* summ)
{
	if (!flag)
	{
		std::cout << begin + 1 << ' ';
		visited[begin] = true;
		if (begin == end)
		{
			flag = true;
			return;
		};
		for (int i = 0; i < this->dim; i++)
		{
			if (!flag && adjacency[begin][i] != 0 && !visited[i])
			{
				DepthSearch(i, end, visited, flag, summ);
				summ[begin] += adjacency[begin][i];
			}
		}
```

```
	}
}*/

//void DoDS_()
//{
//	bool** visited = new bool*[dim];
//	for (int i = 0; i < dim; i++)
//	{
//		visited[i] = new bool[dim];
//	}
//	for (int i = 0; i < dim; i++)
//	{
//		for (int j = 0; j < dim; j++)
//		{
//			visited[i][j] = false;
//		}
//	}
//	int begin = 0;
//	std::cout << "Введите начальный город: ";
//	std::cin >> begin;
//	int end = 0;
//	std::cout << "Введите конечный город: ";
//	std::cin >> end;
//	Routes* params = new Routes();
//	//DepthSearch_(begin-1, end-1, visited, params, 0);
//	bool* visited_ = new bool[dim];
//	for (int i = 0; i < dim; i++)
//	{
//		visited_[i] = false;
//	}
//	RouteParams temp = RouteParams();
//	DepthSearch__(begin - 1, end - 1, visited_, params, temp);
//	for (int i = 0; i < params->len; i++)
//	{
//		std::cout << i << ": " << std::endl;
//		//std::cout << "cost: " << params->routes[i].cost << " len: " <<
params->routes[i].len << " name: " << params->routes[i].name << std::endl;
//	}
//	for (int i = 0; i < dim; i++)
//	{
//		delete[] visited[i];
//	}
//}



//void DepthSearch_(int start, int end, bool**& visited, Routes*& params, int coun-
ter = 0)
//{
//	if (start == end)
//	{
//		return;
//	}
//	for (int i = 0; i < this->dim; i++)
//	{
//		if (adjacency[start][i] != 0 && !visited[start][i])
//		{
//			//Routes* temp = new Routes[params->len+1];
//			////temp->routes = new RouteParams[params->len + 1];
//			//for (int i = 0; i < params->len; i++)
//			//{
//			//	temp->routes[i] = params->routes[i];
//			//}
```

```
//                  //temp->len = params->len;
//                  ///*if (temp->len - 1 >= 0)
//                  //    temp->routes[temp->len] = temp->routes[temp->len - 1];*/
//                  //
//                  //temp->routes[temp->len].cost += adjacency[start][i];
//                  //temp->routes[temp->len].len += 1;
//                  //temp->routes[temp->len].name+=(char)('A' - 1 + i);
//                  //temp->len++;
//                  //delete[] params;
//                  //params = temp;
//                  //visited[start][i] = true;
//                  Routes* temp = new Routes;
//                  temp->routes = new RouteParams[params->len + 1];
//                  for (int i = 0; i < params->len; i++)
//                  {
//                        temp->routes[i] = params->routes[i];
//                  }
//                  temp->len = params->len;
//                  if (temp->len - 1 >= 0)
//                        //temp->routes[temp->len] = temp->routes[temp->len - 1];
//                        temp->routes[temp->len] = temp->routes[counter];
//                  else
//                        temp->routes[temp->len] = RouteParams();
//                  temp->routes[temp->len].cost += adjacency[start][i];
//                  temp->routes[temp->len].len++;
//                  //temp->routes[temp->len].name += (char)('A' + i);
//                  temp->len++;
//                  if(params) delete params;
//                  params = temp;
//                  visited[start][i] = true;
//                  visited[i][start] = true;
//                  counter++;
//                  DepthSearch_(i, end, visited, params, counter);
//            }
//      }
//}

//Routes* GetNewRoute(int start, int i, Routes*& params, RouteParams& previous)
//{
//      Routes* temp = new Routes;
//      temp->routes = new RouteParams[params->len + 1];
//      for (int i = 0; i < params->len; i++)
//      {
//            temp->routes[i] = params->routes[i];
//      }
//      temp->len = params->len;
//      temp->routes[temp->len] = previous.Copy();
//      temp->routes[temp->len].cost += adjacency[start][i];
//      temp->routes[temp->len].len++;
//      //temp->routes[temp->len].name += (char)('A' + i);
//      temp->len++;
//      return temp;
//}

/*bool* SaveVisitedPoint(bool*& visited)
{
      bool* visited_copy = new bool[dim];
      for (int i = 0; i < dim; i++)
      {
            visited_copy[i] = visited[i];
      }
      return visited_copy;
}*/
```

```
	//void DepthSearch__(int start, int end, bool*& visited, Routes*& params,
RouteParams previous)
	//{
	//	std::cout << start << std::endl<<std::endl;
	//	PrintList(visited);
	//	Print();
	//	if (start != end)
	//	{

	//		for (int i = 0; i < this->dim; i++)
	//		{
	//			if (adjacency[start][i] != 0 && !visited[start])
	//			{
	//
	//				/*this->Print();
	//				for (int i = 0; i < dim; i++)
	//				{
	//					std::cout << visited[i] << ' ';
	//				}
	//
	//				std::cout << std::endl;
	//				std::cout << previous.name << std::endl;*/
	//				//if (temp->len - 1 >= 0)
	//				//	//temp->routes[temp->len] = temp-
>routes[temp->len - 1];
	//				//	temp->routes[temp->len] = temp->routes[i];
	//				//else
	//				//	temp->routes[temp->len] = RouteParams();
	//				//
	//				//
	//				Routes* temp = new Routes;
	//				temp->routes = new RouteParams[params->len + 1];
	//				for (int i = 0; i < params->len; i++)
	//				{
	//					temp->routes[i] = params->routes[i];
	//				}
	//				temp->len = params->len;
	//				temp->routes[temp->len] = previous.Copy();
	//				temp->routes[temp->len].cost += adjacency[start][i];
	//				temp->routes[temp->len].len++;
	//				//temp->routes[temp->len].name += (char)('A' + i);
	//				temp->len++;
	//				//auto temp = GetNewRoute(start, i, params, previ-
ous);
	//				if (params) delete params;
	//				params = temp;
	//				visited[start] = true;
	//				adjacency[start][i] = 0;
	//				//auto visited_copy = SaveVisitedPoint(visited);
	//				bool* visited_copy = new bool[dim];
	//				for (int i = 0; i < dim; i++)
	//				{
	//					visited_copy[i] = visited[i];
	//				}
	//				DepthSearch__(i, end, visited, params, temp-
>routes[temp->len-1]);
	//				delete[] visited;
	//				visited = visited_copy;
	//			}
	//		}
	//	}
	//}

	/*void MinDistance(int fSity, int sSity)
	{
```

```
	}*/
};
```

Menu.h

```
#pragma once
#include "Addition.h"
#pragma once
#include "MenuItem.h"
#include <Windows.h>
#include <conio.h>

HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
void SetCursor(short x, short y)
{
	SetConsoleCursorPosition(hStdOut, { x, y });
}
class Menu {
protected:
	string name;
	MenuItem* items = nullptr;
	int countOfItem = 0;
public:
	Menu(string name, int countOfItems, MenuItem* items[]);
	void PrintMenu();
	void RunMenu();
	void AddItem(string name, void (*func)())
	{
		countOfItem++;
		MenuItem* temp = new MenuItem[countOfItem];
		for (int i = 0; i < countOfItem - 1; i++)
		{
			temp[i] = items[i];
		}
		temp[countOfItem - 1].Set(name, func);
		items = temp;
	}
	Menu()
	{
		items = nullptr;
		countOfItem = 0;
	}
	Menu(std::string name)
	{
		this->name = name;
		items = nullptr;
		countOfItem = 0;
	}

};

Menu::Menu(string name, int countOfItems, MenuItem* items[]) {
	this->name = name;
	this->countOfItem = countOfItems;
	for (int i = 0; i < countOfItem; i++) {
		this->items[i] = *items[i];
	}
}

void Menu::PrintMenu() {
	cout << name << "\n";
	for (int i = 0; i < countOfItem; i++) {
```

```
			cout << "  [" << i + 1 << "] " << items[i].Name();
			//items[i].PrintItem();
			cout << "\n";
		}
		cout << "  [0] Выход\n";
}
void Menu::RunMenu() {
		int choice = 1;
		bool input = false;
		bool call = false;
		this->PrintMenu();
		SetCursor(0, 1);
		std::cout << ">";
		do
		{
			char ch = _getch();
			switch (ch)
			{
			case 13:
				call = true;
				input = true;
				break;
			/*case 27:
				choice = 0;
				break;*/
			case 80:
				if (choice <= this->countOfItem)
				{
					input = true;
					choice++;
				}
				else
				{
					choice = 1;
					input = true;
				}
				break;
			case 72:
				if (choice - 1 >= 1)
				{
					input = true;
					choice--;
				}
				else
				{
					choice = this->countOfItem + 1;
					input = true;
				}
				break;
			case -32:
			default:
				input = false;
				break;
			}
			if (input)
			{
				if (call)
				{
					if (choice <= countOfItem && choice >= 0)
					{
						if (choice != 0)
						{
							system("cls");
							items[choice - 1].RunFunction();
							system("pause");
```

```
                                        system("cls");
                                        this->PrintMenu();
                                        SetCursor(0, choice);
                                        std::cout << '>';
                                  }
                            }
                            else if (choice == countOfItem + 1)
                            {
                                  break;
                            }
                      }
                      else
                      {
                            for (int i = 0; i < this->countOfItem + 1; i++)
                            {
                                  SetCursor(0, 1 + i);
                                  std::cout << ' ';
                            }
                            SetCursor(0, choice);
                            std::cout << '>';
                      }
                      call = false;
                }
                input = false;
          } while (choice != 0);
}
```

MenuItem.h

```
#pragma once
#include <iostream>
#include <string>
using namespace std;
class MenuItem {
protected:
      string name;
      void (*func)();
public:
      MenuItem() {};
      MenuItem(string name, void (*func)());
      void RunFunction();
      void PrintItem();
      void Set(string name, void (*func)())
      {
            this->name = name;
            this->func = func;
      }
      std::string& Name()
      {
            return this->name;
      }
};
MenuItem::MenuItem(string name, void(*func)()) {
      this->name = name;
      this->func = func;
}
void MenuItem::RunFunction() {
      this->func();
}
void MenuItem::PrintItem() {
      cout << name;
}
```

Список литературы

1. Алексеев В.Е. Графы и алгоритмы. Структуры данных. Модели вычислений [Электронный ресурс]/ В.Е. Алексеев, В.А. Таланов. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 153 с. — Режим доступа: http://www.iprbookshop.ru/52186.html
2. Вирт Никлаус. Алгоритмы и структуры данных [Электронный ресурс]/ Никлаус Вирт— Электрон. текстовые данные. — Саратов: Профобразование, 2017. — 272 с.— Режим доступа: http://www.iprbookshop.ru/63821.html
3. 3Самуйлов С.В. Алгоритмы и структуры обработки данных [Электронный ресурс]: учебное пособие/ С.В. Самуйлов. — Электрон. текстовые данные. — Саратов: Вузовское образование, 2016. — 132 с.— Режим доступа: http://www.iprbookshop.ru/47275.html