|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет  имени Н. Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н. Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | «Информатика и системы управления» (ИУ) |

|  |  |
| --- | --- |
| КАФЕДРА | «Информационная безопасность» (ИУ8) |

Домашнее задание № 3

ПО КУРСУ

«Алгоритмические языки»

на тему «Объектно ориентированное программирование в языке Си++. Алгоритмы на графах»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ИУ8-23 |  |  |  | В. С. Ажгирей  М. Ю. Григорьева |
|  | (Группа) |  |  |  | (И. О. Фамилия) |
|  |  |  |  |  |  |
| Преподаватель: |  |  |  |  | М. В. Малахов |
|  |  |  |  |  | (И.О. Фамилия) |

2024

Введение

**Вариант 2.** Разработайте приложение для поиска кратчайшего пути из одной вершины в другую в ориентированном взвешенном и невзвешенном (в рамках данной задачи будем считать, что вес всех ребер в невзвешенном графе равен единице) графах. Графы читаются из файла, в приложении должен быть функционал по изменению считанных графов (удалению/добавлению вершин и рёбер, изменению веса ребер для взвешенного графа).

В разработанных классах должны быть конструкторы и операторы перемещения и присваивания, а также как минимум одна переопределенная операция, помимо присваивания.

В приложениях должен присутствовать код для обработки возможных исключительных ситуаций (например, ошибка при чтении данных из файла из­-за некорректности данных).

**Цели и задачи:**

* изучить поставленную задачу
* изучить основы теории графов
* подобрать необходимый алгоритм на графах для текущей задачи
* написать код программы на языке С++;
* разработать тестовые примеры и отладить программу;
* рассмотреть исключительные ситуации и разработать методы их обработки
* подготовить отчет по лабораторной работе.

**Основная часть**

**Исходный текст программы:**

# Файл заголовка mainwindow.h:

#pragma once

#include <vector>

#include <string>

#include <algorithm>

#include <fstream>

#include <iostream>

class Graph {

private:

std::vector<std::vector<int>> graph;

size\_t size;

bool oriented;

std::vector<bool> visited;

std::vector<int> distances;

size\_t start;

public:

Graph();

Graph(std::vector<std::vector<int>>);

Graph(size\_t, bool, std::vector<std::vector<int>>);

Graph(const Graph&);

Graph(Graph&&);

void set\_size(size\_t);

void set\_oriented(bool);

void set\_graph(std::vector<std::vector<int>>);

void set(size\_t, bool, std::vector<std::vector<int>>);

size\_t get\_size();

bool get\_oriented();

std::vector<std::vector<int>> get\_graph();

bool is\_valid();

void add\_vertex();

void delete\_vertex(size\_t);

void add\_edge(size\_t, size\_t, int);

void delete\_edge(size\_t, size\_t);

void dijkstra();

void prepare\_dijkstra(size\_t);

int get\_min\_distance(size\_t, size\_t);

friend void read\_data(Graph&, std::string);

friend std::ostream& operator << (std::ostream&, const Graph&);

friend std::istream& operator >> (std::istream&, Graph&);

};

std::vector<int> split(const std::string&);

void read\_data(Graph&, std::string);

template <typename T>

std::ostream& operator << (std::ostream&, const std::vector<T>);

std::ostream& operator << (std::ostream&, const Graph&);

std::istream& operator >> (std::istream&, Graph&);

# Файл описания graph.cpp:

#include "graph.hpp"

Graph::Graph()

{

size = 0;

oriented = false;

}

Graph::Graph(std::vector<std::vector<int>> graph)

{

this->graph = graph;

size = graph.size();

oriented = false;

}

Graph::Graph(size\_t size, bool oriented, std::vector<std::vector<int>> graph)

{

this->size = size;

this->oriented = oriented;

this->graph = graph;

}

Graph::Graph(const Graph& oth)

{

size = oth.size;

oriented = oth.oriented;

graph = oth.graph;

}

Graph::Graph(Graph&& oth)

{

size = oth.size;

oth.size = 0;

oriented = oth.oriented;

oth.oriented = false;

graph = oth.graph;

oth.graph.clear();

}

void Graph::set\_size(size\_t size)

{

this->size = size;

}

void Graph::set\_oriented(bool oriented)

{

this->oriented = oriented;

}

void Graph::set\_graph(std::vector<std::vector<int>> graph)

{

this->graph = graph;

}

void Graph::set(size\_t size, bool oriented, std::vector<std::vector<int>> graph)

{

this->size = size;

this->oriented = oriented;

this->graph = graph;

}

size\_t Graph::get\_size()

{

return size;

}

bool Graph::get\_oriented()

{

return oriented;

}

std::vector<std::vector<int>> Graph::get\_graph()

{

return graph;

}

bool Graph::is\_valid()

{

if (graph.size() != size) {

std::cout << graph.size();

throw std::runtime\_error("Invalid size " + std::to\_string(size));

}

if (!std::all\_of(graph.begin(), graph.end(), [this](std::vector<int> line)

{

return size == line.size();

}))

throw std::runtime\_error("Invalid size");

if (!oriented)

{

for (size\_t i = 0; i < size; i++)

for (size\_t j = 0; j < size; j++) {

if (graph[i][j] < 0)

throw std::runtime\_error("Invalid distance");

if (graph[i][j] != graph[j][i])

throw std::runtime\_error("Invalid matrix");

}

}

return true;

}

void Graph::add\_vertex()

{

size++;

graph.push\_back(std::vector<int>(size, 0));

for (size\_t i = 0; i < size - 1; i++)

graph[i].resize(size);

}

void Graph::delete\_vertex(size\_t index)

{

for (size\_t i = 0; i < size; i++)

graph[i].erase(graph[i].begin() + index);

graph.erase(graph.begin() + index);

size--;

}

void Graph::add\_edge(size\_t start, size\_t end, int weight = 1)

{

graph[start][end] = weight;

if (!oriented)

graph[end][start] = weight;

}

void Graph::delete\_edge(size\_t start, size\_t end)

{

graph[start][end] = 0;

if (!oriented)

graph[end][start] = 0;

}

void Graph::dijkstra()

{

size\_t vertex = start;

int min\_distance = 0;

distances[start] = 0;

while (min\_distance < INT32\_MAX)

{

size\_t i = vertex;

min\_distance = INT32\_MAX;

visited[i] = true;

for (size\_t j = 0; j < size; j++)

{

if (graph[i][j] && !visited[j] && distances[j] > distances[i] + graph[i][j])

distances[j] = distances[i] + graph[i][j];

if (!visited[j] && distances[j] < min\_distance)

{

min\_distance = distances[j];

vertex = j;

}

}

}

}

void Graph::prepare\_dijkstra(size\_t start)

{

visited = std::vector<bool>(size, false);

distances = std::vector<int>(size, INT32\_MAX);

this->start = start;

}

int Graph::get\_min\_distance(size\_t start, size\_t end)

{

prepare\_dijkstra(start);

dijkstra();

return distances[end];

}

std::vector<int> split(const std::string& line)

{

std::vector<int> res;

size\_t index = 0;

while (line.find(" ", index) != std::string::npos)

{

size\_t new\_index = line.find(" ", index);

std::string word = line.substr(index, new\_index - index);

if (word.length())

res.push\_back(std::stoi(word));

index = new\_index + 1;

}

if (index < line.length())

res.push\_back(std::stoi(line.substr(index)));

return res;

}

void read\_data(Graph& obj, std::string file\_name)

{

std::ifstream fin(file\_name);

if (!fin)

throw std::runtime\_error("wrong file");

std::string line;

std::getline(fin, line);

size\_t size = std::stoi(line);

std::getline(fin, line);

bool oriented = std::stoi(line);

std::vector<std::vector<int>> graph;

while (!fin.eof())

{

std::getline(fin, line);

std::vector<int> arr = split(line);

graph.push\_back(arr);

}

fin.close();

obj.set(size, oriented, graph);

obj.is\_valid();

}

std::ostream& operator<<(std::ostream& out, const Graph& graph)

{

for (size\_t i = 0; i < graph.size; i++)

{

for (size\_t j = 0; j < graph.size; j++)

out << graph.graph[i][j] << " ";

out << std::endl;

}

return out;

}

std::istream& operator>>(std::istream& in, Graph& graph)

{

size\_t n = graph.size;

graph.graph = std::vector<std::vector<int>>(n, std::vector<int>(n));

for (size\_t i = 0; i < n; i++)

for (size\_t j = 0; j < n; j++)

in >> graph.graph[i][j];

return in;

}

template<typename T>

std::ostream& operator<<(std::ostream& out, const std::vector<T> arr)

{

for (T item : arr)

out << item << " ";

return out;

}

**Исполняемый файл main.cpp:**

#include <iostream>

#include "graph.hpp"

int main() {

std::string file\_name = "test3.txt";

Graph graph;

try {

read\_data(graph, file\_name);

std::cout << "Minimum distance: " << graph.get\_min\_distance(0, 7) << std::endl;

}

catch (const std::runtime\_error& ex) {

std::cout << ex.what() << std::endl;

return 1;

}

std::cout << "Graph:\n" << graph << std::endl;

return 0;

}

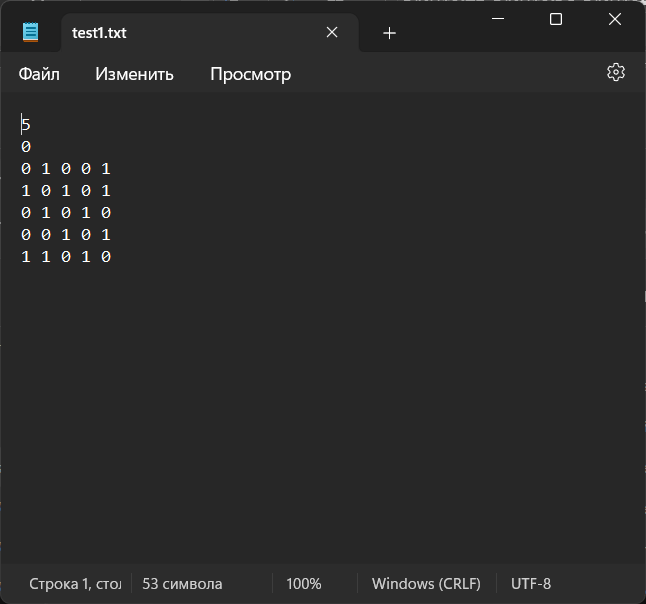
**Снимки выполнения работы программы**

Рисунок 1 – тестовые данные test1

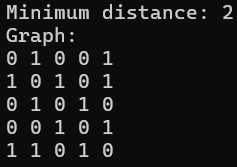


Рисунок 2 – Результат работы программы для тестовых данных test1 (поиск кратчайшего пути из вершины 1 до вершины 3)

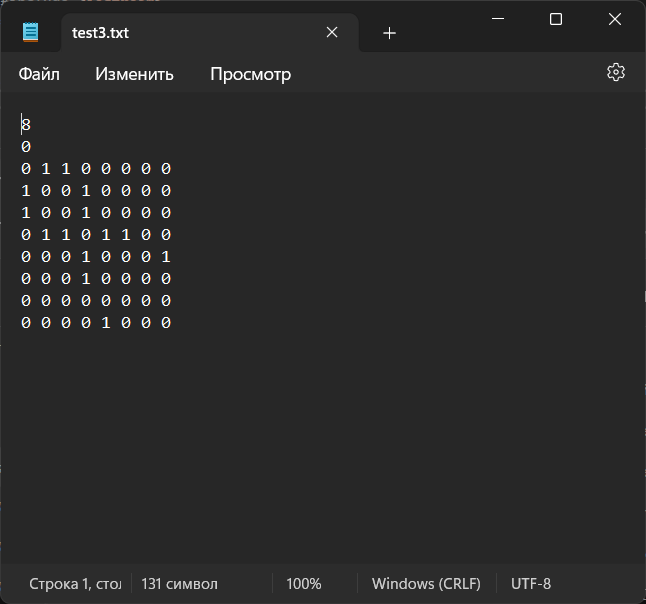


Рисунок 3 – тестовые данные test2

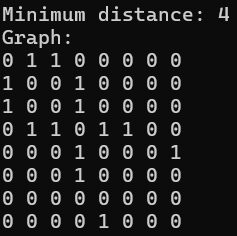


Рисунок 4 – Результат работы программы для тестовых данных test3 (поиск кратчайшего пути из вершины 1 до вершины 8)

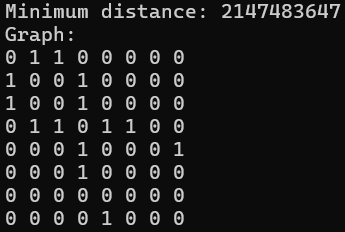


Рисунок 5 – Результат работы программы для тестовых данных test3 (поиск кратчайшего пути из вершины 1 до вершины 7) – бесконечное расстояние, т.е. отсутствие связи между вершинами

Заключение

Задачи лабораторной работы были решены, результаты проверены. Изучены на практике алгоритмы на графах и принципы объектно ориентированного программирования в языке C/C++.