**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Факультет Инфокоммуникационных сетей и систем

Кафедра Защищенных систем связи

Дисциплина Разработка защищенных сетевых приложений

**Отчет по курсовой работе**

*Расчет минимального пути графа метрополитена СПб*

*(тема )*

Студент:

Попов В. С. 

*(Ф.И.О., № группы) (подпись)*

Преподаватель:

Цветков А. Ю.

*(уч. степень, уч. звание, Ф.И.О.) (подпись)*

Санкт-Петербург

2023

**СОДЕРЖАНИЕ**

[Введение 3](file:///C:\Users\user\AppData\Local\Temp\Rar$DIa40564.21973\Отчет.docx#_Toc501391539)

[1 Описание задачи и проектирование архитектуры программного обеспечения 4](file:///C:\Users\user\AppData\Local\Temp\Rar$DIa40564.21973\Отчет.docx#_Toc501391540)

[1.1 Описание задачи и формализация требований к программному обеспечению 4](file:///C:\Users\user\AppData\Local\Temp\Rar$DIa40564.21973\Отчет.docx#_Toc501391541)

[1.2 Проектирование архитектуры программного обеспечения 8](file:///C:\Users\user\AppData\Local\Temp\Rar$DIa40564.21973\Отчет.docx#_Toc501391541)

[2 Реализация программного обеспечения 10](file:///C:\Users\user\AppData\Local\Temp\Rar$DIa40564.21973\Отчет.docx#_Toc501391540)

3 [Тестирование программного обеспечения 17](file:///C:\Users\user\AppData\Local\Temp\Rar$DIa40564.21973\Отчет.docx#_Toc501391547)

[Заключение 21](file:///C:\Users\user\AppData\Local\Temp\Rar$DIa40564.21973\Отчет.docx#_Toc501391547)

[Список используемой литературы 22](file:///C:\Users\user\AppData\Local\Temp\Rar$DIa40564.21973\Отчет.docx#_Toc501391547)

Приложение А Исходный код 23

**ВВЕДЕНИЕ**

Объектом исследования в курсовой работе является метрополитен СПб, а предметом: реализация программы для поиска кратчайшего маршрута от одной станции метро до другой.

Цель курсовой работы: разработка программного продукта, в котором пользователь сможет выбрать начальную станцию и конечную и получить минимальный путь с указанием времени его в минутах.

Задачи, которые требуется решить, чтобы достигнуть поставленной цели:

1) Разработать классы, отвечающие за загрузку исходных данных о графе из файла и их обработку.

2) Разработать оконный интерфейс позволяющий увидеть схему метрополитена СПб и задать начальную станцию и конечную с кнопкой “Поиск”.

3) Добавить возможность многократно использовать программу на различных входных данных.

В первой главе приведена полная постановка задачи и формализация требований к разрабатываемому программному продукту, а также архитектура программы в виде диаграммы классов.

Во второй главе описана поэтапная разработка программного обеспечения (классы, оконный интерфейс).

В третьей главе проводится тестирование разработанных классов.

**1 ОПИСАНИЕ ЗАДАЧИ И ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**1.1 Описание задачи и формализация требований к программному обеспечению**

Исходные данные: реализовать программу по расчету минимального пути графа метрополитена СПб, используя матрицу смежности характеризующая граф. Весом ребра графа является время, затрачиваемое для перемещения от одной станции к другой.

Поскольку предстоит работать с графами, то стоит привести некоторые определения из теории:

**Неориентированным графом** G называется пара G = (V, E), где V — множество вершин, а E ⊂ {{v, u}: v, u ∈V} ⊂ {{v, u}: u, v ∈ V} — множество рёбер. [5].

Пример неориентированного графа приведен на рисунке 1.

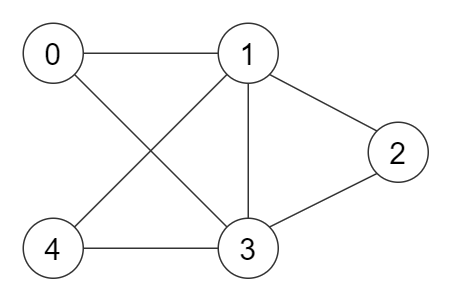


Рисунок 1 – Неориентированный граф

Два ребра являются **смежными**, если они имеют общую концевую вершину.

Два ребра называются **кратными**, если множества их концевых вершин совпадают.

**Степень** deg*V* вершины *V* есть количество рёбер, для которых она является концевой.

**Путь** в графе есть конечная последовательность вершин, в которой каждая вершина (кроме последней) соединена со следующей в последовательности вершин ребром [6].

Теперь схему метрополитена СПб можем упростим до следующего неориентированного графа (см. рисунок 2).

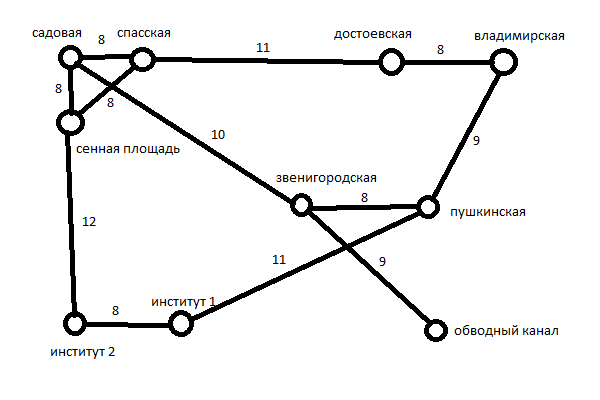


Рисунок 2 – Упрощенная схема метро

Для тестирования программы достаточно лишь небольшой части метрополитена СПб, которую в дальнейшем при желании можно будет расширить / дополнить другими станциями метро.

В самой программе неориентированный граф может быть представлен в виде:

1. Матрицы смежности;
2. Списков смежности;
3. Матрицы инциденций.

По заданию требуется представить граф в виде матрицы смежности поэтому остановимся на нем.

Вес дуги неориентированного графа будет ассоциироваться со временем, которое потребуется, чтобы добраться от станции A до станции B.

Найти кратчайший путь между двумя выбранными станциями метро возможно при помощи широко известного алгоритма Дейкстры [2].

Схема алгоритма Дейкстры приведена на рисунке 3.

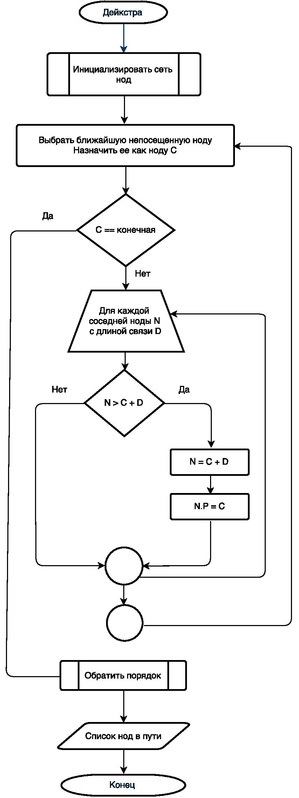


Рисунок 3 – Алгоритм Дейкстры

С учетом объектно-ориентированного подхода требования к разрабатываемой программе определяются в виде диаграммы вариантов использования (use case) [1]. Совокупность всех возможных вариантов использования описывает функциональные возможности системы в целом (рисунок 4).

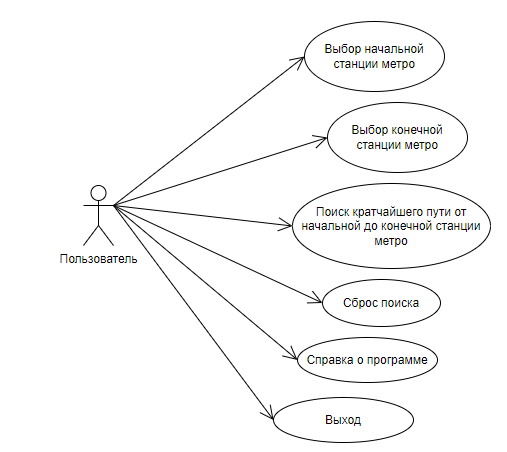


Рисунок 4 – Диаграмма вариантов использования

**1.2 Проектирование архитектуры программного обеспечения**

Проведем объектно-ориентированную декомпозицию задачи и выделим следующие классы (рисунок 5).

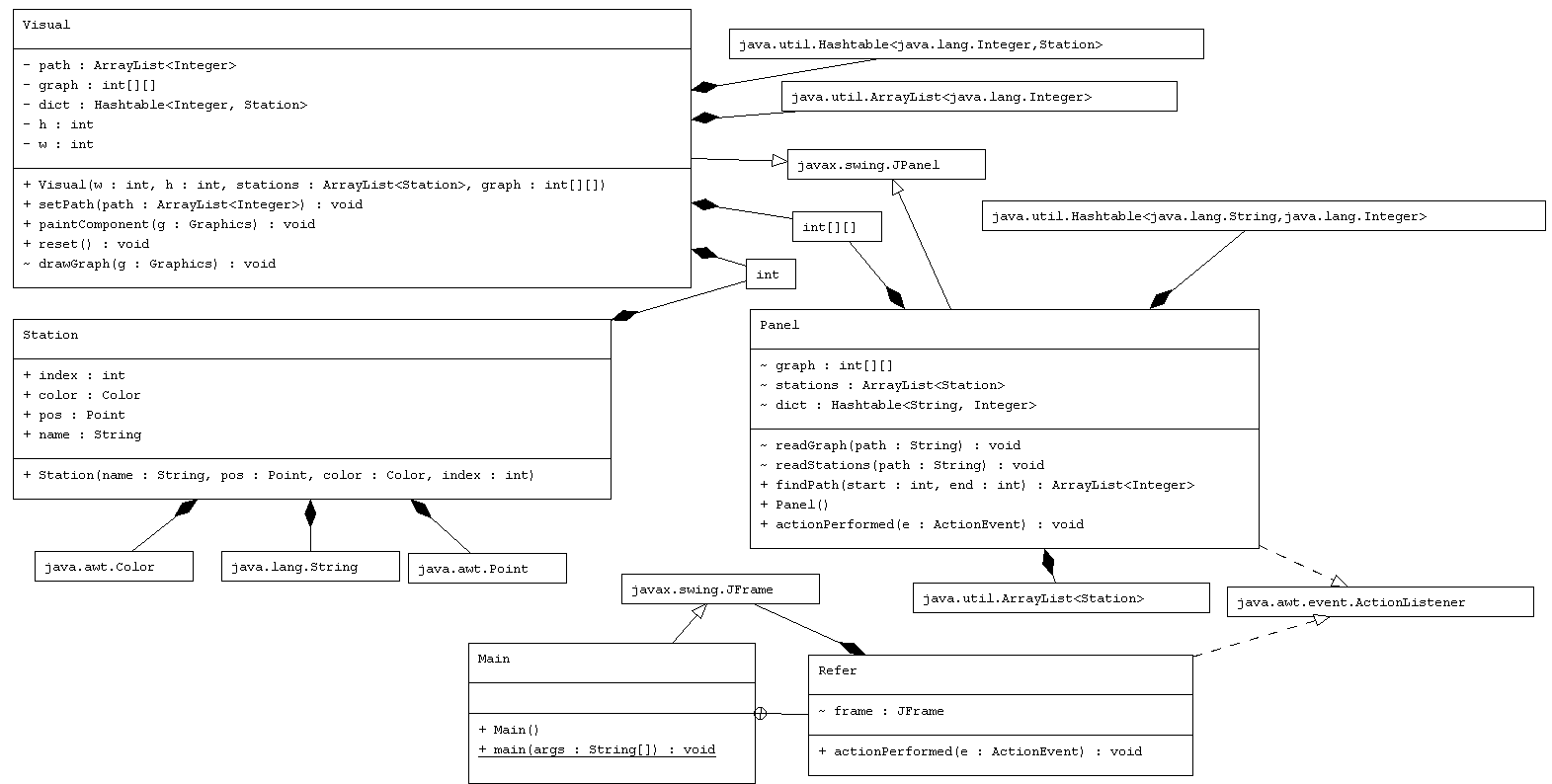


Рисунок 5 – UML диаграмма классов

Класс Main – главная форма приложения, в котором описывается дополнительный класс “Справка” и создается экземпляр класса Panel;

Класс Panel – включает панель для отрисовки графа и все остальные компоненты, чтобы пользователь смог выбрать станции и начать поиск кратчайшего пути. Кроме того, содержит 3 основных метода:

1. readGraph – читает граф из файла и переводит его в матрицу смежности попутно заполняя словарь для быстрого перевода названия станции в номер / индекс массива;
2. readStations – читает станции метро и координаты по которым их нужно отобразить на панели;
3. findPath – ищет кратчайший путь из вершины start до вершины end.

Класс Visual – производит отрисовку графа по матрице смежности и кратчайшего пути;

Класс Station – содержит название станции метро, позицию на панели (координаты X, Y), цвет и номер для идентификации.

**2 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Начнем реализацию с методов класса Main (главная форма приложения).

Класс Main.java:

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

*//основная программа*

**public** **class** Main **extends** JFrame {

*//справка*

**public** **class** Refer **implements** ActionListener {

JFrame frame;

@Override

**public** **void** actionPerformed(ActionEvent e) {

**if** (frame != **null**) {

frame.dispose();

}

frame = **new** JFrame();

frame.setVisible(**true**);

frame.setSize(**new** Dimension(500, 280));

frame.setTitle("Справка");

frame.setLayout(**null**);

JLabel about = **new** JLabel(

"<html>" +

"<p>'Начальная станция' - выбирается станция от которой начинается формирование кратчайшего пути;</p><br>" +

"<p>'Конечная станция' - выбирается станция до которой формируется кратчайший путь;</p><br>" +

"<p>'Общее время в пути' - длина кратчайшего пути в минутах;</p><br>" +

"<p>Кнопка 'Поиск' - отображает кратчайший путь красной линией на рисунке и добавляет его время в минутах на метку 'Общее время в пути';</p><br>" +

"<p>Кнопка 'Сброс' - сбрасывает начальную и конечную станции, а также убирает ранее найденный кратчайший путь;</p><br>" +

"</html>");

about.setBounds(20, 20, 450, 220);

frame.add(about);

frame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

frame.setResizable(**false**);

}

}

*//конструктор*

**public** Main() {

*//настройка формы*

setResizable(**false**);

setTitle("Метро СПб");

**int** w = 656;

**int** h = 600;

setPreferredSize(**new** java.awt.Dimension(w, h));

GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

pack();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

*//о программе*

JMenuBar menuBar = **new** JMenuBar();

JMenu menu = **new** JMenu("О программе");

*//справка*

JMenuItem refer = **new** JMenuItem("Справка");

refer.addActionListener(**new** Refer());

menu.add(refer);

*//размещаем меню*

menuBar.add(menu);

setJMenuBar(menuBar);

*//добавляем панель*

JPanel panel = **new** Panel();

panel.setBounds(0, 0, w, h);

panel.setLayout(**null**);

add(panel);

}

*//главная функция*

**public** **static** **void** main(String[] args) {

**new** Main().setVisible(**true**);

}

}

Класс Panel включает методы для чтения исходных данных из файлов graph.txt и stations.txt, структура которых приведена далее.

Структура файла graph.txt:

Количество станций метро N

Название\_1 Название\_2-Вес Название\_3-Вес…Название\_N-Вес

…

Названиe\_N

Структура файла stations.txt:

Количество станций метро N

Название\_1 X Y C

…

Название\_N X Y C

, где X, Y – координаты вершины графа на панели, а C – цвет вершины.

Зная теперь как устроены файлы не составит труда написать под них соответствующие методы:

Метод readGraph (читает файл graph.txt):

*//получаем данные о станциях из файла (матрица смежности)*

**void** readGraph(String path) {

**try** {

BufferedReader reader = **new** BufferedReader(**new** FileReader(path));

String line = reader.readLine(); *//размер графа*

**int** size = Integer.parseInt(line);

graph = **new** **int**[size][size];

dict.clear();

**int** k = 0;

*//по каждой вершине графа*

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

*//формат строки:*

*// название тек. ст. -> смежные ст. метро - время прибытия от тек. ст.*

line = reader.readLine();

String[] lines = line.split(" ");

String name1 = lines[0];

**if** (!dict.containsKey(name1)) {

dict.put(name1, k++); *//присваиваем уник. номер*

}

*//цикл по каждой смежной вершине*

**for** (**int** j = 1; j < lines.length; j++) {

String[] vert = lines[j].split("-");

**if** (vert.length == 2) {

*//получаем смежную вершину и её вес*

String name2 = vert[0];

**int** min = Integer.parseInt(vert[1]);

**if** (!dict.containsKey(name2)) {

dict.put(name2, k++); *//присваиваем уник. номер*

}

*//граф неориентированный*

graph[dict.get(name1)][dict.get(name2)] = min;

graph[dict.get(name2)][dict.get(name1)] = min;

} **else** {

**throw** **new** Exception("Вершина графа должна иметь вес!");

}

}

}

reader.close();

} **catch** (Exception e) {

showMessageDialog(**null**, e.getMessage(),

"Ошибка!", JOptionPane.INFORMATION\_MESSAGE);

}

}

Метод readStations (читает файл stations.txt):

*//получаем список станций и их местоположение для отрисовки*

**void** readStations(String path) {

**try** {

BufferedReader reader = **new** BufferedReader(**new** FileReader(path));

String line = reader.readLine(); *//кол-во вершин*

**int** size = Integer.parseInt(line);

stations.clear();

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

line = reader.readLine();

*//формат строки: название позиция по X позиция по Y цвет вершины*

String[] lines = line.split(" ");

**if** (lines.length == 4) {

Point p = **new** Point(Integer.parseInt(lines[1]), Integer.parseInt(lines[2]));

String c = lines[3];

Color color = Color.green;

**if** (c.equals("m")) {

color = Color.magenta;

} **else** **if** (c.equals("r")) {

color = Color.red;

} **else** **if** (c.equals("b")) {

color = Color.blue;

} **else** **if** (c.equals("o")) {

color = Color.orange;

}

Station station = **new** Station(lines[0], p, color, dict.get(lines[0]));

stations.add(station);

} **else** {

**throw** **new** Exception("Неверно указан формат станции метро!");

}

}

reader.close();

} **catch** (Exception e) {

showMessageDialog(**null**, e.getMessage(),

"Ошибка!", JOptionPane.INFORMATION\_MESSAGE);

}

}

Сами станции выбирает пользователь, после чего производится поиск кратчайшего пути от станции start до end с помощью алгоритма Дейкстры:

*//поиск пути из start в end*

*//алгоритм Дейкстры*

**public** ArrayList<Integer> findPath(**int** start, **int** end) {

**int** inf = 10000;

**int** n = stations.size();

**int**[] d = **new** **int**[n];

**boolean**[] used = **new** **boolean**[n];

**int**[] p = **new** **int**[n];

**for** (**int** i = 0; i < n; i++) {

d[i] = inf;

used[i] = **false**;

}

d[start] = 0;

p[start] = -1;

**for** (**int** i = 0; i < n; i++) {

*//находим следующую непосещенную вершину*

**int** v = -1;

**for** (**int** j = 0; j < n; j++) {

**if** (!used[j] && (v == -1 || d[j] < d[v])) {

v = j;

}

}

*//вершина недостижима*

**if** (d[v] == inf) {

**break**;

}

used[v] = **true**;

*//проводим релаксации*

**for** (**int** j = 0; j < n; j++) {

**if** (graph[v][j] != 0) {

**if** (d[v] + graph[v][j] < d[j]) {

d[j] = d[v] + graph[v][j];

p[j] = v;

}

}

}

}

*//восстанавливаем путь*

ArrayList<Integer> path = **new** ArrayList<>();

**for** (**int** v = end; v != -1; v = p[v]) {

path.add(v);

}

Collections.reverse(path);

**return** path;

}

Полный листинг класса Panel можно найти в приложении А.

Класс Visual.java:

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.util.ArrayList;

**import** java.util.Hashtable;

*//панель для отрисовки*

**public** **class** Visual **extends** JPanel {

**private** **int** w, h; *//размеры панели*

**private** Hashtable<Integer, Station> dict; *//словарь для быстрого доступа к данным*

**private** **int**[][] graph; *//матрица смежности*

**private** ArrayList<Integer> path; *//кратчайший маршрут*

*//конструктор*

**public** Visual(**int** w, **int** h, ArrayList<Station> stations, **int**[][] graph) {

**this**.w = w;

**this**.h = h;

**this**.graph = graph;

dict = **new** Hashtable<>();

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

dict.put(stations.get(i).index, stations.get(i));

}

setPath(**new** ArrayList<>());

}

*//установка кр. маршрута*

**public** **void** setPath(ArrayList<Integer> path) {

**this**.path = path;

repaint();

}

*//отрисовка компонентов*

@Override

**public** **void** paintComponent(Graphics g) {

**super**.paintComponent(g);

drawGraph(g);

}

*//сброс*

**public** **void** reset() {

setPath(**new** ArrayList<>());

repaint();

}

*//отрисовка графа*

**void** drawGraph(Graphics g) {

**int** paddX = -25; *//смещение по x*

**int** d = 15; *//диаметр круга*

*//пути для каждой вершины по матрице смежности*

g.setColor(Color.BLACK);

**for** (**int** i = 0; i < graph.length; i++) {

**for** (**int** j = 0; j < graph[i].length; j++) {

**if** (graph[i][j] != 0) {

Station start = dict.get(i);

Station end = dict.get(j);

*//ребро*

g.drawLine(start.pos.x + d / 2 + paddX, start.pos.y + d / 2,

end.pos.x + d / 2 + paddX, end.pos.y + d / 2);

*//вес ребра*

**int** mx = (start.pos.x + end.pos.x) / 2 + d / 2 + paddX;

**int** my = (start.pos.y + end.pos.y) / 2 + d / 2;

g.drawString(graph[i][j] + "", mx, my - 2);

}

}

}

*//кратчайший путь*

**if** (path.size() > 1) {

g.setColor(Color.red);

**for** (**int** i = 0; i < path.size() - 1; i++) {

Station start = dict.get(path.get(i));

Station end = dict.get(path.get(i + 1));

g.drawLine(start.pos.x + d / 2 + paddX, start.pos.y + d / 2,

end.pos.x + d / 2 + paddX, end.pos.y + d / 2);

}

}

*//названия и подписи*

**for** (**int** index : dict.keySet()) {

Station s = dict.get(index);

g.setColor(s.color);

g.fillOval(s.pos.x + paddX, s.pos.y, d, d);

g.drawString(s.name, s.pos.x + paddX, s.pos.y - 6);

}

}

}

Данный класс отвечает за отрисовку: вершин графа, названий станций метро, весов (минут), дуг и кратчайшего пути по возможности.

Класс Station также можно найти в приложении A.

**3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Для проверки работоспособности программы возьмем следующие наборы данных:

Файл graph.txt:

10

садовая спасская-4 сенная\_площадь-5 звенигородская-10

спасская достоевская-11 сенная\_площадь-3

достоевская владимирская-4

звенигородская пушкинская-5 обводный\_канал-9

владимирская пушкинская-9

пушкинская институт\_1-11

институт\_1 институт\_2-6

институт\_2 сенная\_площадь-12

сенная\_площадь

обводный\_канал

Файл stations.txt:

10

садовая 70 50 m

спасская 150 50 o

сенная\_площадь 70 150 b

достоевская 400 50 o

владимирская 510 50 r

пушкинская 430 200 r

звенигородская 300 200 m

обводный\_канал 450 330 m

институт\_1 180 330 r

институт\_2 70 330 b

Запустим программу и увидим, что граф был построен корректно и все вершины соответствуют тем, что были представлены в файлах (см. рисунок 6).

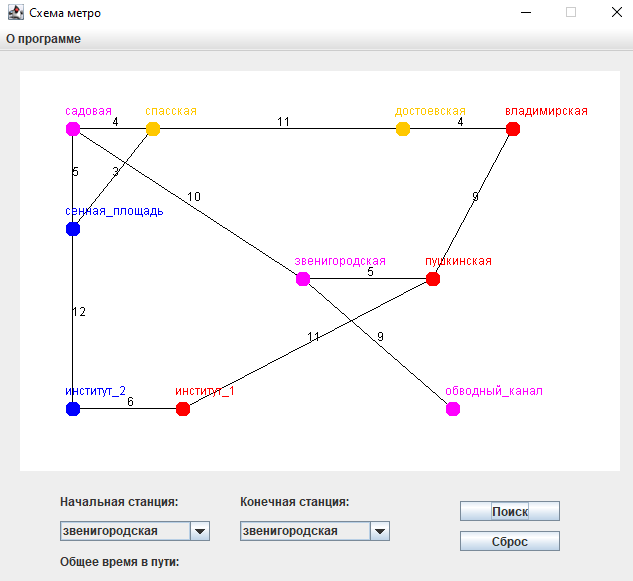


Рисунок 6 – Первый запуск приложения

Попробуем найти кратчайший путь от станции “Садовая” до “Институт 1” (см. рисунок 7).

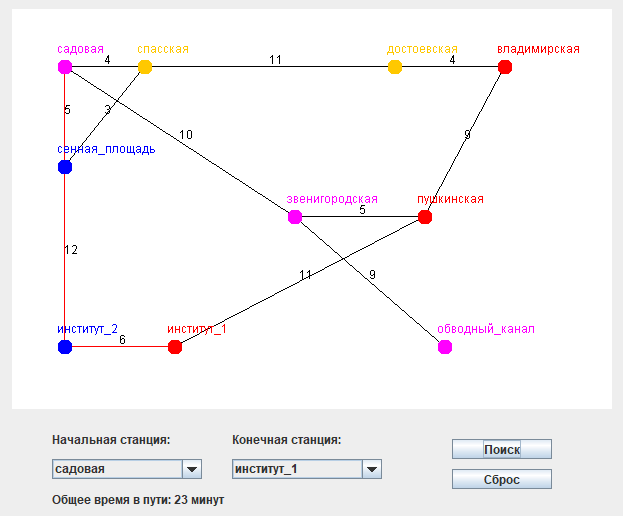


Рисунок 7 – Кратчайший путь от “Садовая” до “Институт 1”

Теперь найдем путь от станции “Достоевская” до “Институт 2” (см. рисунок 8).

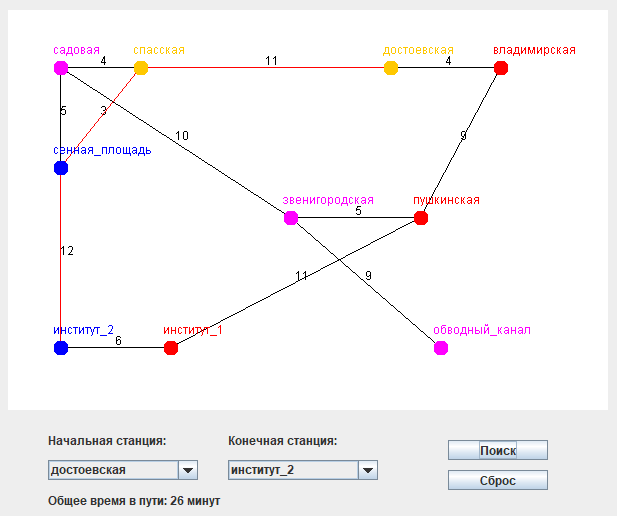


Рисунок 8 – Кратчайший путь от “Достоевская” до “Институт 2”

Судя по проведенному тестированию, программа работает корректно.

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы было реализовано программное обеспечение при помощи объектно-ориентированного языка программирования Java позволяющее найти путь минимальной длины от начальной станции метрополитена СПб до конечной с указанием времени его в минутах. Поставленная на курсовую работу цель была достигнута.

**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. Самуйлов С.В. Объектно-ориентированное моделирование на основе UML [Электронный ресурс]: учебное пособие / С.В. Самуйлов. — Электрон. текстовые данные. — Саратов: Вузовское образование, 2016. — 37 c. — 2227- 8397. — URL: http://www.iprbookshop.ru/47277.html (дата обращения: 15.04.2023).

2. Kormen T. Introduction to Algorithms / T. Kormen, C. Leizerson, R. Rivest, K. Stein. 2005. — 1296 с.

3. Катаев М. Ю. Объектно-ориентированное программирование: учеб. пособие / М. Ю. Катаев, А. Я. Суханов. – Томск: ТМЦДО, 2007. – 160 с.

4. Романенко В. В. Объектно-ориентированное программирование : учеб. пособие / В. В. Романенко. – Томск: ФДО, ТУСУР, 2014. – 475 с.

5. Харари Фрэнк Теория графов = Graph theory/Пер. с англ. и предисл. В. П. Козырева. Под ред. Г.П.Гаврилова. Изд. 2-е. — М.: Едиториал УРСС, 2003. — 296 с. — ISBN 5-354-00301-6.

6. Асанов М. О., Баранский В. А., Расин В. В. Дискретная математика: графы, матроиды, алгоритмы — НИЦ РХД, 2001. — 288 с. — ISBN 5-93972-076-5.

7. Ссылка на «GitHub» - https://github.com/Nezz1s/IBS-23\_KR\_Popov

**ПРИЛОЖЕНИЕ A**

**Исходный код**

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

*//основная программа*

**public** **class** Main **extends** JFrame {

*//справка*

**public** **class** Refer **implements** ActionListener {

JFrame frame;

@Override

**public** **void** actionPerformed(ActionEvent e) {

**if** (frame != **null**) {

frame.dispose();

}

frame = **new** JFrame();

frame.setVisible(**true**);

frame.setSize(**new** Dimension(500, 280));

frame.setTitle("Справка");

frame.setLayout(**null**);

JLabel about = **new** JLabel(

"<html>" +

"<p>'Начальная станция' - выбирается станция от которой начинается формирование кратчайшего пути;</p><br>" +

"<p>'Конечная станция' - выбирается станция до которой формируется кратчайший путь;</p><br>" +

"<p>'Общее время в пути' - длина кратчайшего пути в минутах;</p><br>" +

"<p>Кнопка 'Поиск' - отображает кратчайший путь красной линией на рисунке и добавляет его время в минутах на метку 'Общее время в пути';</p><br>" +

"<p>Кнопка 'Сброс' - сбрасывает начальную и конечную станции, а также убирает ранее найденный кратчайший путь;</p><br>" +

"</html>");

about.setBounds(20, 20, 450, 220);

frame.add(about);

frame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

frame.setResizable(**false**);

}

}

*//конструктор*

**public** Main() {

*//настройка формы*

setResizable(**false**);

setTitle("Схема метро");

**int** w = 656;

**int** h = 600;

setPreferredSize(**new** java.awt.Dimension(w, h));

GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

pack();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

*//о программе*

JMenuBar menuBar = **new** JMenuBar();

JMenu menu = **new** JMenu("О программе");

*//справка*

JMenuItem refer = **new** JMenuItem("Справка");

refer.addActionListener(**new** Refer());

menu.add(refer);

*//размещаем меню*

menuBar.add(menu);

setJMenuBar(menuBar);

*//добавляем панель*

JPanel panel = **new** Panel();

panel.setBounds(0, 0, w, h);

panel.setLayout(**null**);

add(panel);

}

*//главная функция*

**public** **static** **void** main(String[] args) {

**new** Main().setVisible(**true**);

}

}

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.io.BufferedReader;

**import** java.io.FileReader;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Hashtable;

**import** static javax.swing.JOptionPane.showMessageDialog;

*//интерфейс*

**public** **class** Panel **extends** JPanel **implements** ActionListener {

Hashtable<String, Integer> dict; *//словарь станций*

ArrayList<Station> stations; *//массив для отрисовки*

**int**[][] graph; *//матрица смежности*

*//получаем данные о станциях из файла (матрица смежности)*

**void** readGraph(String path) {

**try** {

BufferedReader reader = **new** BufferedReader(**new** FileReader(path));

String line = reader.readLine(); *//размер графа*

**int** size = Integer.parseInt(line);

graph = **new** **int**[size][size];

dict.clear();

**int** k = 0;

*//по каждой вершине графа*

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

*//формат строки:*

*// название тек. ст. -> смежные ст. метро - время прибытия от тек. ст.*

line = reader.readLine();

String[] lines = line.split(" ");

String name1 = lines[0];

**if** (!dict.containsKey(name1)) {

dict.put(name1, k++); *//присваиваем уник. номер*

}

*//цикл по каждой смежной вершине*

**for** (**int** j = 1; j < lines.length; j++) {

String[] vert = lines[j].split("-");

**if** (vert.length == 2) {

*//получаем смежную вершину и её вес*

String name2 = vert[0];

**int** min = Integer.parseInt(vert[1]);

**if** (!dict.containsKey(name2)) {

dict.put(name2, k++); *//присваиваем уник. номер*

}

*//граф неориентированный*

graph[dict.get(name1)][dict.get(name2)] = min;

graph[dict.get(name2)][dict.get(name1)] = min;

} **else** {

**throw** **new** Exception("Вершина графа должна иметь вес!");

}

}

}

reader.close();

} **catch** (Exception e) {

showMessageDialog(**null**, e.getMessage(),

"Ошибка!", JOptionPane.INFORMATION\_MESSAGE);

}

}

*//получаем список станций и их местоположение для отрисовки*

**void** readStations(String path) {

**try** {

BufferedReader reader = **new** BufferedReader(**new** FileReader(path));

String line = reader.readLine(); *//кол-во вершин*

**int** size = Integer.parseInt(line);

stations.clear();

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

line = reader.readLine();

*//формат строки: название позиция по X позиция по Y цвет вершины*

String[] lines = line.split(" ");

**if** (lines.length == 4) {

Point p = **new** Point(Integer.parseInt(lines[1]), Integer.parseInt(lines[2]));

String c = lines[3];

Color color = Color.green;

**if** (c.equals("m")) {

color = Color.magenta;

} **else** **if** (c.equals("r")) {

color = Color.red;

} **else** **if** (c.equals("b")) {

color = Color.blue;

} **else** **if** (c.equals("o")) {

color = Color.orange;

}

Station station = **new** Station(lines[0], p, color, dict.get(lines[0]));

stations.add(station);

} **else** {

**throw** **new** Exception("Неверно указан формат станции метро!");

}

}

reader.close();

} **catch** (Exception e) {

showMessageDialog(**null**, e.getMessage(),

"Ошибка!", JOptionPane.INFORMATION\_MESSAGE);

}

}

*//поиск пути из start в end*

*//алгоритм Дейкстры*

**public** ArrayList<Integer> findPath(**int** start, **int** end) {

**int** inf = 10000;

**int** n = stations.size();

**int**[] d = **new** **int**[n];

**boolean**[] used = **new** **boolean**[n];

**int**[] p = **new** **int**[n];

**for** (**int** i = 0; i < n; i++) {

d[i] = inf;

used[i] = **false**;

}

d[start] = 0;

p[start] = -1;

**for** (**int** i = 0; i < n; i++) {

*//находим следующую непосещенную вершину*

**int** v = -1;

**for** (**int** j = 0; j < n; j++) {

**if** (!used[j] && (v == -1 || d[j] < d[v])) {

v = j;

}

}

*//вершина недостижима*

**if** (d[v] == inf) {

**break**;

}

used[v] = **true**;

*//проводим релаксации*

**for** (**int** j = 0; j < n; j++) {

**if** (graph[v][j] != 0) {

**if** (d[v] + graph[v][j] < d[j]) {

d[j] = d[v] + graph[v][j];

p[j] = v;

}

}

}

}

*//восстанавливаем путь*

ArrayList<Integer> path = **new** ArrayList<>();

**for** (**int** v = end; v != -1; v = p[v]) {

path.add(v);

}

Collections.reverse(path);

**return** path;

}

*//конструктор*

**public** Panel() {

*//загрузка данных из файлов*

dict = **new** Hashtable<>();

stations = **new** ArrayList<>();

readGraph("graph.txt");

readStations("stations.txt");

**if** (dict.size() != stations.size()) {

showMessageDialog(**null**, "Количество вершин для " +

"отрисовки и для расчета " +

"не совпадают!",

"Внимание!", JOptionPane.INFORMATION\_MESSAGE);

}

*//панель для отрисовки станций метро*

**int** w = 600;

**int** h = 400;

**int** paddX = 40; *//отступ слева*

Visual panel = **new** Visual(w, h, stations, graph);

panel.setBounds(20, 20, w, h);

panel.setBackground(Color.WHITE);

add(panel);

*//установка начальной станции*

JLabel lb1 = **new** JLabel("Начальная станция:");

JComboBox combo1 = **new** JComboBox(dict.keySet().toArray());

lb1.setBounds(paddX + 20, h + 40, 200, 20);

combo1.setBounds(paddX + 20, h + 70, 150, 20);

add(lb1);

add(combo1);

*//установка конечной станции*

JLabel lb2 = **new** JLabel("Конечная станция:");

JComboBox combo2 = **new** JComboBox(dict.keySet().toArray());

lb2.setBounds(paddX + 200, h + 40, 200, 20);

combo2.setBounds(paddX + 200, h + 70, 150, 20);

add(lb2);

add(combo2);

*//примерное время*

JLabel lb3 = **new** JLabel("Общее время в пути:");

lb3.setBounds(paddX + 20, h + 100, 200, 20);

add(lb3);

*//найти кратчайший путь*

JButton find = **new** JButton("Поиск");

find.addActionListener((e) -> {

String start = combo1.getSelectedItem().toString();

String end = combo2.getSelectedItem().toString();

ArrayList<Integer> path = findPath(dict.get(start), dict.get(end));

**int** d = 0;

**for**(**int** i = 0; i < path.size() - 1; i++){

d += graph[path.get(i)][path.get(i + 1)];

}

lb3.setText("Общее время в пути: " + d + " минут");

panel.setPath(path);

});

find.setBounds(paddX + 420, h + 50, 100, 20);

add(find);

*//сброс поиска*

JButton reset = **new** JButton("Сброс");

reset.addActionListener((e) -> {

combo1.setSelectedIndex(0);

combo2.setSelectedIndex(0);

lb3.setText("Общее время в пути:");

panel.reset();

});

reset.setBounds(paddX + 420, h + 80, 100, 20);

add(reset);

}

@Override

**public** **void** actionPerformed(ActionEvent e) {

repaint();

}

}

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.util.ArrayList;

**import** java.util.Hashtable;

*//панель для отрисовки*

**public** **class** Visual **extends** JPanel {

**private** **int** w, h; *//размеры панели*

**private** Hashtable<Integer, Station> dict; *//словарь для быстрого доступа к данным*

**private** **int**[][] graph; *//матрица смежности*

**private** ArrayList<Integer> path; *//кратчайший маршрут*

*//конструктор*

**public** Visual(**int** w, **int** h, ArrayList<Station> stations, **int**[][] graph) {

**this**.w = w;

**this**.h = h;

**this**.graph = graph;

dict = **new** Hashtable<>();

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

dict.put(stations.get(i).index, stations.get(i));

}

setPath(**new** ArrayList<>());

}

*//установка кр. маршрута*

**public** **void** setPath(ArrayList<Integer> path) {

**this**.path = path;

repaint();

}

*//отрисовка компонентов*

@Override

**public** **void** paintComponent(Graphics g) {

**super**.paintComponent(g);

drawGraph(g);

}

*//сброс*

**public** **void** reset() {

setPath(**new** ArrayList<>());

repaint();

}

*//отрисовка графа*

**void** drawGraph(Graphics g) {

**int** paddX = -25; *//смещение по x*

**int** d = 15; *//диаметр круга*

*//пути для каждой вершины по матрице смежности*

g.setColor(Color.BLACK);

**for** (**int** i = 0; i < graph.length; i++) {

**for** (**int** j = 0; j < graph[i].length; j++) {

**if** (graph[i][j] != 0) {

Station start = dict.get(i);

Station end = dict.get(j);

*//ребро*

g.drawLine(start.pos.x + d / 2 + paddX, start.pos.y + d / 2,

end.pos.x + d / 2 + paddX, end.pos.y + d / 2);

*//вес ребра*

**int** mx = (start.pos.x + end.pos.x) / 2 + d / 2 + paddX;

**int** my = (start.pos.y + end.pos.y) / 2 + d / 2;

g.drawString(graph[i][j] + "", mx, my - 2);

}

}

}

*//кратчайший путь*

**if** (path.size() > 1) {

g.setColor(Color.red);

**for** (**int** i = 0; i < path.size() - 1; i++) {

Station start = dict.get(path.get(i));

Station end = dict.get(path.get(i + 1));

g.drawLine(start.pos.x + d / 2 + paddX, start.pos.y + d / 2,

end.pos.x + d / 2 + paddX, end.pos.y + d / 2);

}

}

*//названия и подписи*

**for** (**int** index : dict.keySet()) {

Station s = dict.get(index);

g.setColor(s.color);

g.fillOval(s.pos.x + paddX, s.pos.y, d, d);

g.drawString(s.name, s.pos.x + paddX, s.pos.y - 6);

}

}

}

**import** java.awt.\*;

*//одна станция метро*

**public** **class** Station {

**public** String name; *//название*

**public** Point pos; *//позиция при отрисовке*

**public** Color color; *//цвет*

**public** **int** index; *//для идентификации*

*//конструктор*

**public** Station(String name, Point pos, Color color, **int** index) {

**this**.name = name;

**this**.pos = pos;

**this**.color = color;

**this**.index = index;

}

}