

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по учебной практике
Тема: Минимальное остовное дерево

Студент гр. 0382		Кондратов Ю. К.
Студент гр. 0382	_____	Литягин С. М.
Студент гр. 0382	_____	Сергеев Д. А.
Руководитель	_____	Фирсов М. А.

Санкт-Петербург
2022

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Кондратов Ю. К. группы 0382

Студент Литягин С. М. группы 0382

Студент Сергеев Д. А. группы 0382

Тема практики: Минимальное остовное дерево

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Kotlin и Java с графическим интерфейсом.

Алгоритм: алгоритм Краскала.

Сроки прохождения практики: 29.06.2022 – 12.07.2022

Дата сдачи отчета: 11.07.2022

Дата защиты отчета: 11.07.2022

Студент		Кондратов Ю. К.
Студент		Литягин С. М.
Студент		Сергеев Д. А.
Руководитель		Фирсов М. А.

АННОТАЦИЯ

Целью учебной практики является изучение новых языков программирования, а также разработка графического приложения для нахождения наименьшего остовного дерева в графе при помощи алгоритма Краскала.

Приложение разрабатывается на языке Java и Kotlin командой из трёх человек, каждый из которых выполняет определенные задачи.

SUMMARY

The purpose of the training practice is to study new programming languages, as well as to develop a graphical application for finding the smallest spanning tree in a graph using the Kruskal algorithm.

The application is developed in Java and Kotlin by a team of three people, each of whom performs certain tasks.

СОДЕРЖАНИЕ

	Введение	5
1.	Требования к программе	6
1.1.	Исходные требования к программе	6
1.1.1.	Требования к вводу исходных данных	6
1.1.2.	Требования к графическому интерфейсу	6
1.1.3.	Требования к работе алгоритма	8
1.2.	Уточнение требований после сдачи 1-ой версии	8
1.3.	Уточнение требований после сдачи 2-ой версии	9
2.	План разработки и распределение ролей в бригаде	10
2.1.	План разработки	10
2.2.	Распределение ролей в бригаде	11
3.	Особенности реализации	12
3.1.	Особенности реализации алгоритма Краскала	12
3.2.	Особенности реализации графического интерфейса	15
4.	Тестирование	23
4.1	Тестирование внутренней бизнес-логики программы	23
4.2	Тестирование графического интерфейса	26
	Заключение	37
	Список использованных источников	38
	Приложение А. Исходный код программы	39

ВВЕДЕНИЕ

Цель работы.

Изучить новый язык программирования и реализовать визуализатор алгоритма Краскала с графическим интерфейсом при помощи языков Kotlin и Java.

Реализуемый алгоритм применяется для построения минимального остовного дерева взвешенного связного неориентированного графа.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе

1.1.1. Требования к вводу исходных данных

Возможность ввода исходных данных через файл;

Формат входных данных в случае двух вышеперечисленных вариантов должен иметь вид:

N – число ребер графа;

В следующих N строках “ $v_i v_j w_{ij}$ ” – ребро графа $v_i v_j$ с весом w_{ij} .

Возможность ввода исходных данных через графический интерфейс;

1.1.2. Требования к графическому интерфейсу

На рисунке 1 представлен эскиз графического интерфейса.

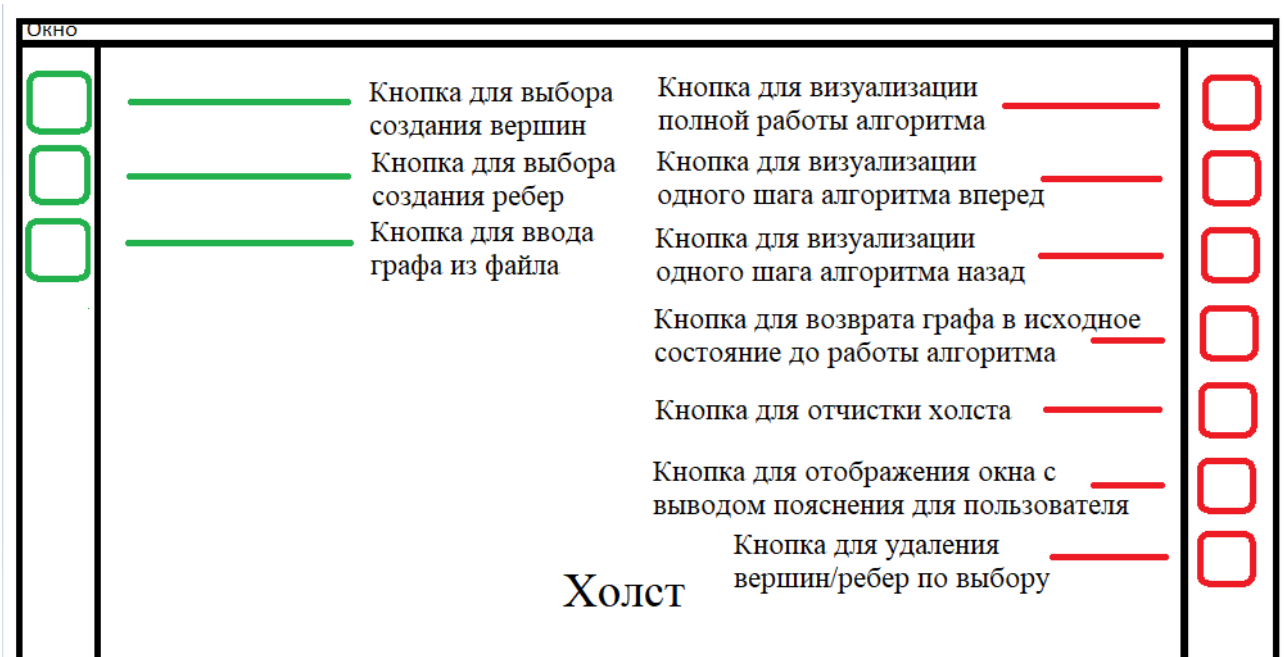


Рисунок 1 – Эскиз графического интерфейса

Возможность создания на холсте по клику мыши вершины с заданием ее наименования (должна быть активна соответствующая кнопка);

Возможность создания ребра между имеющимися вершинами с заданием его веса (должна быть активна соответствующая кнопка). Создание ребра производится левым кликом мыши по вершинам, которые соединяет создаваемое ребро, далее открывается контекстное меню с выбором веса;

Возможность отрисовки графа, ввод которого произведен через файл. При вводе через файл, узлы будут располагаться в вершинах углов правильного многоугольника с центром в середине экрана;

Изначальный граф на холсте имеет черный цвет;

Возможность выполнения полной пошаговой визуализации работы алгоритма по нажатию кнопки (на каждом шаге алгоритма будет выделяться серым цветом; если цикл образуется, то он будет выделен синим цветом, на следующем шаге возвращаются нужные цвета; если ребро не образует цикл, то оно будет перекрашено в красный цвет и вершины, инцидентные ему). При нажатии на кнопку запуска открывается контекстное меню с ползунком выбора скорости работы алгоритма;

Также вместе с полной визуализацией алгоритма будет заполняться файл-лог с пояснениями для пользователя. Выводимое пояснение будет иметь вид:

Описание алгоритма: на каждом шаге выбирается ребро с минимальным весом и не образующее цикл с уже выбранными ребрами.

Начало работы алгоритма;

Выбрано ребро AB с весом w

...

Ребро CD не выбрано, т.к. образует цикл BCD

...

Конец работы алгоритма;

Возможность выполнения одного шага алгоритма вперед по нажатию кнопки (описание выводимых пояснений аналогично полной визуализации алгоритма);

Возможность вывода пояснений для пошаговой версии по нажатию соответствующей кнопки;

Возможность выполнения одного шага алгоритма назад по нажатию кнопки (аналогично выполнению полной визуализации, ребро, выбранное в последнем шаге алгоритма, будет перекрашено в изначальное состояние);

Возможность вернуться в исходное состояние по нажатию кнопки;

Возможность очистки холста;

Возможность удаления вершин и ребер с холста (должна быть активна соответствующая кнопка). Для удаления нужно будет кликнуть по вершине или ребру, которое нужно удалить;

Возможность перемещения вершин по холсту (должна быть активна кнопка создания вершины). Для этого нужно зажать левую кнопку мыши на вершине и перетащить ее по холсту;

1.1.3. Требования к работе алгоритма

Возможность полного выполнения алгоритма (по завершении работы полученное дерево будет выведено в отдельный файл в формате: " $v_i v_j w_{ij}$ " – ребро графа $v_i v_j$ с весом w_{ij});

Возможность выполнения одного шага алгоритма вперед;

Возможность выполнения одного шага алгоритма назад;

Возможность вывода промежуточных данных;

1.2. Уточнение требований после сдачи 1-ой версии

Реализация автоматического добавления нумерации к имени вершины в случае, если такое имя уже существует или в соответствующем поле ничего не введено;

Реализация изменения веса ребра (для этого должна быть активна кнопка создания ребра; при нажатии на уже существующее ребро его вес будет изменен на указанный в соответствующем поле);

Реализация запроса на полный путь к файлу с входными данными (при нажатии на кнопку ввода графа через файл);

Реализация обработки случая, что во время выполнения визуализации алгоритма (пошаговой/полной) может быть изменен граф (возможность перемещения вершин должна быть сохранена);

Реализация изменения размеров окна и холста (можно реализовать в 3-ей версии);

Реализация добавления ребер после перемещения вершин по холсту

1.3. Уточнение требований после сдачи 2-ой версии

Сделать более внятные пояснения к выбору скорости визуализации;

Выделять выбранную кнопку;

Исправить некорректную расцветку после цикла;

Пояснения при наведении мыши;

Перемещение вершин не должно останавливать алгоритм;

Исправить баг, что после перемещения вершины удаление не приводит к остановке визуализации;

Шкала скорости от 1 до 20, где 19 – почти мгновенно (50 мс), 20 – сразу результат шага алгоритма или его полной визуализации.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

План разработки представлен в таблице 1.

Таблица 1 – План разработки

№ п/п	Цель	Срок выполнения
1	Распределение ролей	01.07.2022
2	Реализация приложения, демонстрирующего интерфейс (создание кнопок, реализация создания/удаления вершин и ребер на холсте, реализация очистки холста). Сдача прототипа	04.07.2022
3	Реализация структуры данных дерева, реализация ввода исходных данных через файл и через GUI	05.07.2022
4	Реализация алгоритма Краскала (полностью все шаги алгоритма и выполнение отдельного шага алгоритма), реализация полной пошаговой визуализации выполнения алгоритма. Сдача 1-ой версии	06.07.2022
5	Исправление замечаний и внесение правок. Реализация визуализации шага алгоритма назад, реализация выбора скорости визуализации полного алгоритма, реализация вывода пояснений для полной и пошаговой визуализаций, реализация выделения циклов другим цветом на шаге. Сдача 2-ой версии	08.07.2022
6	Исправление замечаний и внесение правок. Сдача финальной версии	11.07.2022

2.2. Распределение ролей в бригаде

В данной работе Кондратов Ю. А. реализовывает следующие задачи на языке Java:

Структура данных дерева, алгоритм Краскала (все шаги, один шаг вперед, один шаг назад), вывод пояснений для пошаговой версии алгоритма, ввод исходных данных через файл, вывод остовного дерева в файл.

В данной работе Литягин С. М. реализовывает следующие задачи на языке Kotlin:

Реализация интерфейса приложения, ввод исходных данных через GUI, удаление вершин/ребер с холста, отчистка холста, возврат графа в исходное состояние, перемещение вершин по холсту.

В данной работе Сергеев Д. А. реализовывает следующие задачи на языке Kotlin:

Реализация визуализации полной/пошаговой версии алгоритма, выбор скорости работы визуализации алгоритма.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Особенности реализации алгоритма Краскала

Реализация бизнес-логики приложения, а именно воспроизведения алгоритма Краскала на заданном графе, происходила в два этапа.

На этапе проектирования был разработан интерфейс взаимодействия с графической частью приложения. Интерфейс состоит из следующих методов:

1. *public void run()* - метод, запускающий алгоритм.
2. *public List<Edge> stepForward()* - метод, возвращающий ребро добавленное на очередном шаге, либо список ребер, образующих цикл.
3. *public List<Edge> stepBack()* - метод, возвращающий ребра, цвет которых нужно изменить для визуализации шага назад
4. *public String getLog()* - метод, возвращающий строку с информацией о процессе работы алгоритма
5. *public boolean isConnected()* - метод, предназначенный для проверки поданного на вход алгоритму графа на связность.

На этапе реализации сначала было принято решение о способе хранения графа. Граф хранится в виде списка ребер, каждое ребро представляет из себя объект класса *Edge*. В данном классе имеется три приватных поля: *String start* и *String end* для хранения концов ребра; *int weight* для хранения веса ребра.

В процессе работы алгоритма происходит сортировка списка ребер, поэтому необходимо средствами языка программирования указать, что сравнение двух ребер происходит по величине веса. Для этого класс *Edge* реализует интерфейс *Comparable<Edge>*, переопределяя метод *compareTo*.

Далее ,было необходимо реализовать способ проверки на образование цикла. Для решения этой задачи был выбран способ, связанный с использованием системы непересекающихся множеств. Эта структура данных реализована в классе *DisjointSets*.

Непосредственно реализация алгоритма и интерфейса взаимодействия с графической частью находится в классе *Kruskal*. В этом классе содержатся следующие **поля**:

1. *List<Edge> edges* - поле для хранения обрабатываемого графа;
2. *List<Edge> mst* - поле для хранения остовного дерева;
3. *DisjointSets<String> vertices* - поле, хранящее систему непересекающихся множеств на вершинах графа - необходимо для проверки на образование цикла;
4. *List<List<Edge>> steps* - список информации о всех шагах алгоритма - необходимо для реализации обратного хода алгоритма и вывода информации о его работе;
5. *int i* - номер текущего шага работы алгоритма;
6. *StringBuilder log* - поле, хранящее текущую информацию о работе алгоритма.

Рассмотрим теперь **основные методы**, реализованные в классе *Kruskal*:

1. *public Kruskal(Collection<Edge> edges)* - конструктор класса алгоритма. В нем происходит инициализация всех необходимых полей.
2. *public boolean isConnected()* - проверка графа на связность. Проверка реализована следующим образом: создается система непересекающихся множеств на вершинах графа, после чего производится проход по всем ребрам графа и объединение множеств соединенных этими ребрами вершин. Если в итоге в системе остается единственное множество, то граф связан.
3. *public void run()* - метод, запускающий алгоритм. В данном методе производится полный прогон алгоритма, в результате чего список в поле *steps* заполняется информацией, необходимой для отображения шага в графическом интерфейсе.
4. *public List<Edge> stepForward()* - метод, возвращающий результат *i* шага - список ребер (элемент списка *steps*). Если очередное ребро добавлено успешно, то список состоит только из этого ребра, а если очередное ребро

образовало цикл, то список состоит из ребер, входящих в этот цикл. Также в методе производится запись информации о шаге в поле *log* и инкрементирование значения поля *i*. Если работа алгоритма закончена, метод вернет *null*.

5. *public List<Edge> stepBack()* - метод, возвращающий результат предыдущего шага для отображения шага назад. Возвращаемые значения аналогичны методу *stepForward*. В методе производится запись в *log* и декрементирование значения поля *i*.

6. *public String getLog()* - метод, информацию о работе алгоритма. Представляет из себя геттер для поля *log*.

Также в классе *Kruskal* реализованы некоторые приватные

вспомогательные методы:

1. *private String cycleToString(List<Edge> res)* - метод, предназначенный для преобразования цикла в виде списка ребер в его строковое представление.

2. *private List<Edge> DFS(Map<String, Map<String, Integer>> graph, String start, String end, Set<String> visited, List<Edge> path)* - метод, предназначенный для поиска пути от вершины *start* до вершины *end* пути в графе *graph*.

3. *private Map<String, Map<String, Integer>> fromEdgesToGraph(List<Edge> edges)* - метод, предназначенный для преобразование графа, представленного в виде списка ребер в его представление в виде списка инцидентности, реализованного при помощи *Map*.

4. *private List<Edge> getCycle(Edge bridge)* - метод, предназначенный для поиска цикла в графе. В этом методе сначала происходит преобразование графа в виде списка ребер в его представление в виде списка инцидентности при помощи метода *fromEdgesToGraph*, затем на полученном графе ищется путь от вершины начала до вершины конца ребра *bridge*, при добавлении которого в остов образуется цикл. Такой путь будет единственным, так как граф по построению представляет из

себя дерево. При добавлении к этому пути ребра `bridge` получим искомый цикл.

5. *private List<Edge> getCycledTree(Edge bridge)* - метод, предназначенный для получения списка ребер, представляющих часть остова, в которой при добавлении ребра `bridge` образуется цикл.

3.2. Особенности реализации графического интерфейса

Графический интерфейс реализован на языке Kotlin с помощью библиотеки TornadoFX.

Чтобы создать приложение Tornado FX, был создан класс *MyApp*, наследуемый от класса *App*. Данный класс является точкой входа в приложение и определяет начальное представление, согласно аргументам конструктора. Начальным представлением является класс *Interface*, наследуемый от абстрактного класса *View* (далее такие классы будут именоваться классам). Для его реализации переопределяется переменная *root*, приравняв объекту класса *BorderPane*. Это класс категории “контейнер”, имеющий 5 элементов: *center*, *left*, *right*, *bottom*, *top*. Также класс *Interface* имеет два поля: *canvasController*: *CanvasController* и *mController*: *MenuController*. Это объекты классов-контроллеров, которые будут реализовывать основную бизнес-логику GUI.

Вернемся к *root*. Как было сказано ранее, в данной переменной хранится контейнер класса *BorderPane*. В данном приложении используется лишь 3 его элемента: *center*, *right*, *left*.

В элементе *left* располагается объект класса-представления *LeftMenu*, в элементе *right* - объект класса-представления *RightMenu*. В элементе *center* располагается объект класса *ScrollPane*, в котором располагается объект класса *Pane* (также является классом категории “контейнер”). Также здесь происходит привязка полей *canvasWidth* и *canvasHeight* объекта *canvasController* к значениям полей ширины и высоты элемента (т.е. в случае изменения ширины или высоты значения соответствующих полей *canvasController*’а также

изменяться). *Pane* - это наш основной “холст”, на котором будет располагаться граф. Данному контейнеру устанавливаются размеры пользовательского дисплея за вычетом ширины *RightMenu* приложения. Также полям *pane* объектов *mController* и *canvasController* передается ссылка на данный контейнер, а еще происходит привязка полей *canvasMaxWidth* и *canvasMaxHeight* объекта *canvasController* к значениям максимальной широты и максимальной высоты “холста”. *ScrollPane* позволит нам просматривать разные области “холста”, путем перемещения вертикального и горизонтального ползунков. Размеры этой области приравниваются размерам, доступным элементу *center* в окне приложения. Помимо этого, у объекта *Pane* реализуется обработка события *onMouseClicked*. В этом случае, если активной кнопкой будет являться кнопка *Node*, будет вызван метод объекта *mController* *btNode(x, y)*, где *x, y* - координата клика мыши по *Pane*. Данный метод описан далее.

Класс-представления *LeftMenu*. Для его реализации переопределяется переменная *root*, приравняв объекту класса *Vbox*. Также класс *LeftMenu* имеет поле *mController: MenuController*. В данном контейнере все объекты располагаются вертикально. Здесь были расположены объекты в следующем порядке (далее упоминается enum-класс *NameButton*; подробнее о нем в описании класса-контроллера *MenuController*):

1. *Label* с текстом “Create:”, к которому также применен стиль *forLabels* из класса *Styles*;
2. *Button* с текстом “Node”, к которому также применен стиль *forActButton* из класса *Styles* и присвоен *id* = “NODE”. При нажатии на эту кнопку вызывается метод *btClicked(NameButton.NODE)* объекта *mController*;
3. *Textfield*, которому установлен *prompt*-текст “Input node's name”; также при инициализации в поле *nodeTextField* объекта *mController* передается ссылка на это поле;
4. *Button* с текстом “Edge”, к которому также применен стиль *forDisButton* из класса *Styles* и присвоен *id* = “EDGE”. При нажатии на эту кнопку вызывается метод *btClicked(NameButton.EDGE)* объекта *mController*;

5. *Textfield*, которому установлен *prompt*-текст “Input edge's weight” и применен фильтр ввода, разрешающий вводить лишь целые числа; также при инициализации в поле *weightTextField* объекта *mController* передается ссылка на это поле;
6. *Label* с текстом “Import from file:”, к которому также применен стиль *forLabels* из класса *Styles*;
7. *Button* с текстом “Graph”, к которому также применен стиль *forDisButton* из класса *Styles* и присвоен *id* = “GRAPH”. При нажатии на эту кнопку вызывается метод *btGraph()* объекта *mController*;
8. *Label* с текстом “Clear canvas:”, к которому также применен стиль *forLabels* из класса *Styles*;
9. *Button* с текстом “Clear”, к которому также применен стиль *forDisButton* из класса *Styles* и присвоен *id* = “CLEAR”. При нажатии на эту кнопку вызывается метод *btClear()* объекта *mController*;
10. *Label* с текстом “Delete edge/node:”, к которому также применен стиль *forLabels* из класса *Styles*;
11. *Button* с текстом “Delete”, к которому также применен стиль *forDisButton* из класса *Styles* и присвоен *id* = “DELETE”. При нажатии на эту кнопку вызывается метод *btClicked(NameButton.DELETE)* объекта *mController*;

Класс-представления *RightMenu*. Для его реализации переопределяется переменная *root*, приравняв объекту класса *Vbox*. Также класс *RightMenu* имеет поле *mController*: *MenuController*. В данном контейнере все объекты располагаются вертикально. Здесь были расположены объекты в следующем порядке:

1. *Label* с текстом “Visualisation:”, к которому также применен стиль *forLabels* из класса *Styles*;
2. *Button* с текстом “Full”, к которому также применен стиль *forDisButton* из класса *Styles* и присвоен *id* = “FULL”. При нажатии на эту кнопку вызывается метод *btFull()* объекта *mController*;

3. *Button* с текстом “One step in”, к которому также применен стиль *forDisButton* из класса *Styles* и присвоен *id* = “ONE_STEP_IN”. При нажатии на эту кнопку вызывается метод *btOneIn()* объекта *mController*;
4. *Button* с текстом “One step out”, к которому также применен стиль *forDisButton* из класса *Styles* и присвоен *id* = “ONE_STEP_OUT”. При нажатии на эту кнопку вызывается метод *btOneOut()* объекта *mController*;
5. *Textfield*, которому установлен *prompt*-текст “Input speed (1..20)” и применен фильтр ввода, разрешающий вводить лишь целые числа; также при инициализации в поле *speedTextField* объекта *mController* передается ссылка на это поле;
6. *Label* с текстом “Initial state:”, к которому также применен стиль *forLabels* из класса *Styles*;
7. *Button* с текстом “Graph”, к которому также применен стиль *forDisButton* из класса *Styles* и присвоен *id* = “GRAPH_START”. При нажатии на эту кнопку вызывается метод *btGraphStart()* объекта *mController*;
8. *Label* с текстом “Explanation:”, к которому также применен стиль *forLabels* из класса *Styles*;
9. *Button* с текстом “Show”, к которому также применен стиль *forDisButton* из класса *Styles* и присвоен *id* = “SHOW”. При нажатии на эту кнопку вызывается метод *btShow()* объекта *mController*;
10. *Label* с текстом “Save explanations in file:”, к которому также применен стиль *forLabels* из класса *Styles*;
11. *Button* с текстом “Save”, к которому также применен стиль *forDisButton* из класса *Styles* и присвоен *id* = “SAVE”. При нажатии на эту кнопку вызывается метод *btSave()* объекта *mController*;

Вернемся к ранее упомянутым классам-контроллерам.

Класс *CanvasController*. Отвечает за создание вершин и ребер на холсте.

Содержит методы:

1. *setThisPane(pane: Pane)* – используется для привязки холста к классу-контроллеру
2. *checkEdge(id1: String, id2: String): Boolean* – проверяет, существует ли ребро между вершинами *id1* и *id2* и возвращает *false*, если ребро не существует, и *true* в обратном случае
3. *addCircle(x: Double, y: Double, nodeRadius: Double, currText: String): Boolean* – отвечает за добавление вершины графа на холст: рисует шар с радиусом *nodeRadius*, в центре которого написано имя вершины – *currText*, также устанавливает слушатели различных действия с вершиной:
 - a. Нажатие на левую клавишу мыши(далее ЛКМ) – перестают отображаться ребра, инцидентные данной вершине;
 - b. Нажатие на ЛКМ и движение мыши, при этом не отпуская ЛКМ – вершина передвигается по холсту, при этом нельзя наложить одну вершину на другую, а также вынести вершину за пределы холста;
 - c. Отпускание ЛКМ – Если активна кнопка Node, то рёбра, скрытые при перемещении вершины, снова отображаются на холсте, если активна кнопка Edge, то если это первое нажатие по вершине с активной кнопкой Edge – данная вершина запоминается, если второе – строится ребро, между вершиной, которая запоминается при первом нажатии и текущей вершиной, вес ребра равен числу, вписанному в соответствующее поле(также устанавливается слушатель нажатий на ребро, позволяющий поменять вес ребра или удалить его), если активна кнопка Delete, то удаляется вершина и рёбра, инцидентные ей, возвращает *true*, после выполнения всех действий
4. *clearAllObject()* – очищает холст
5. *addGraphEdge(id1: String, id2: String, weight: Int)* – метод добавления ребра, при считывание графа с файла

6. *aloneNode()*: *Boolean* – возвращает *true*, если существует вершина, которой не инцидентно ни одно ребро, и *false* в обратном случае.

Класс *MenuController*. Отвечает за взаимодействие между элементами интерфейса и нижестоящими контролерами. Содержит методы:

1. *setNodeText(textField: TextField)* – устанавливает поле ввода названия вершины;
2. *setWeightText(textField: TextField)* – устанавливает поле ввода веса ребра;
3. *setSpeedText(textField: TextField)* – устанавливает поле ввода скорости выполнения алгоритма;
4. *btClicked(nameButton: NameButton)* – устанавливает нажатую кнопку активной (класс перечисления *NameButton* состоит из *id* каждой кнопки);
5. *btNode(x: Double, y: Double)* – обрабатывает нажатие на холст с активной кнопкой “Node”, передаёт информацию из полей *canvasController*’у для дальнейшей обработки;
6. *underLine()* – перерисовывает вершины
7. *btGraph()* - обрабатывает нажатие на кнопку “Graph” в *LeftMenu*, в случае, если в стандартной директории имеется файл *graph.txt*, то считывает его, в обратном случае – позволяет пользователю выбрать файл для считывания и запоминает путь к нему, затем передаёт *canvasController*’у дальнейшую обработку;
8. *visualisationSpeed()* - возвращает значение скорости визуализации, вычисляемое согласно значению в поле *speedTextField*;
9. *btClear()* – обрабатывает нажатие на кнопку “Clear”, передаёт дальнейшую обработку *canvasController*’у;
10. *btShow()* – обрабатывает нажатие на кнопку “Show”, открывает окно с выводом пояснений;
11. *btSave()* – обрабатывает нажатие на кнопку “Save”, сохраняет пояснения в файл по стандартному пути, если таковой существует, в обратном случае

- позволяет пользователю выбрать файл для сохранения и запоминает путь к нему;
12. *initKruskal()* – инициализирует граф для выполнения алгоритма Краскала, если граф не удовлетворяет требованиям, то пользователю выводится окно с ошибкой;
 13. *visDisable(value: Boolean)* – делает часть интерфейса доступным/недоступным для взаимодействия в зависимости от *value*;
 14. *btFull()* – обрабатывает нажатие на кнопку “Full”, выполняются все шаги в алгоритме Краскала;
 15. *btOneIn()* – обрабатывает нажатие на кнопку “One Step In” выполняется один шаг вперёд в алгоритме Краскала;
 16. *btOneOut()* – обрабатывает нажатие на кнопку “One Step Out”, выполняется один шаг назад в алгоритме Краскала;
 17. *btGraphStart()* – обрабатывает нажатие на кнопку “Graph” в *RightMenu*, возвращает граф в состояние до выполнения алгоритма Краскала;
 18. *colorInBlack()* – перекрашивает все элементы холста в черный цвет;
 19. *notConnectedGraph()* – выводит окно с ошибкой о том, что граф не удовлетворяет требованиям.

Класс-перечисление *NameButton*. Содержит в себе константы - уникальные коды кнопок, используется для определения кнопки, которая была нажата.

Класс *Styles*. Отвечает за стили элементов интерфейса. Определяет атрибуты для каждого элемента интерфейса:

1. *forCanv* - для “холста”
 - a. Задний фон - (r: 213, g: 213, b: 213, o: 1.0)

2. *forLables* - для объектов *lable*:

- a. Отступы - 4 пикселя
- b. Размер шрифта - 12 пикселей
- c. Выделение - полужирный
- d. Подчеркивание - присутствует

3. *forActButton* - активная кнопка:

- a. Выделение - жирный
- b. Обводка - сверху - темно красный, справа - темно зеленый, слева - темно оранжевый, снизу - фиолетовый
- c. Задний фон - линейный градиент
- d. Цвет текста - (r: 254, g: 229, b:174, o: 1.0)

4. *forDisButton* - неактивная кнопка:

- a. Выделение - жирный
- b. Обводка - сверху - красный, справа - темно зеленый, слева - оранжевый, снизу - фиолетовый

Разработанный программный код см. в приложении А.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование внутренней бизнес-логики программы

1) Тестирование метода run.

В результате работы этой функции в поле mst объекта класса Kruskal должно находиться верно построенное минимальное остовное дерево.

Результаты тестирования представлены в таблице 2.

Таблица 2 - Результат тестирования метода run

№	Тест	Результат (содержимое поля mst)	Вердикт
1	1 a b 1	a b 1	passed
2	3 a b 1 b c 2 a c 3	a b 1 b c 2	passed
3	4 a b 1 b c 1 b d 2 c d 3	a b 1 b c 1 b d 2	passed
4	5 a b 1 b c 1 a c 1 c d 2 b d 2	a b 1 b c 1 c d 2	passed

5	14	7 6 1	passed
	0 1 4	6 5 2	
	1 2 8	2 8 2	
	2 3 7	0 1 4	
	3 4 9	2 5 4	
	0 7 8	2 3 7	
	7 6 1	1 2 8	
	6 5 2	3 4 9	
	5 4 10		
	1 7 11		
	2 8 2		
	2 5 4		
	3 5 14		
	7 8 7		
	6 8 6		

2) Тестирование метода stepForward.

Данный метод возвращает ребро, добавленное на очередном шаге либо цикл, а также записывает информацию в поле log. Тестирование производилось на графе, представленном на рисунке 2.

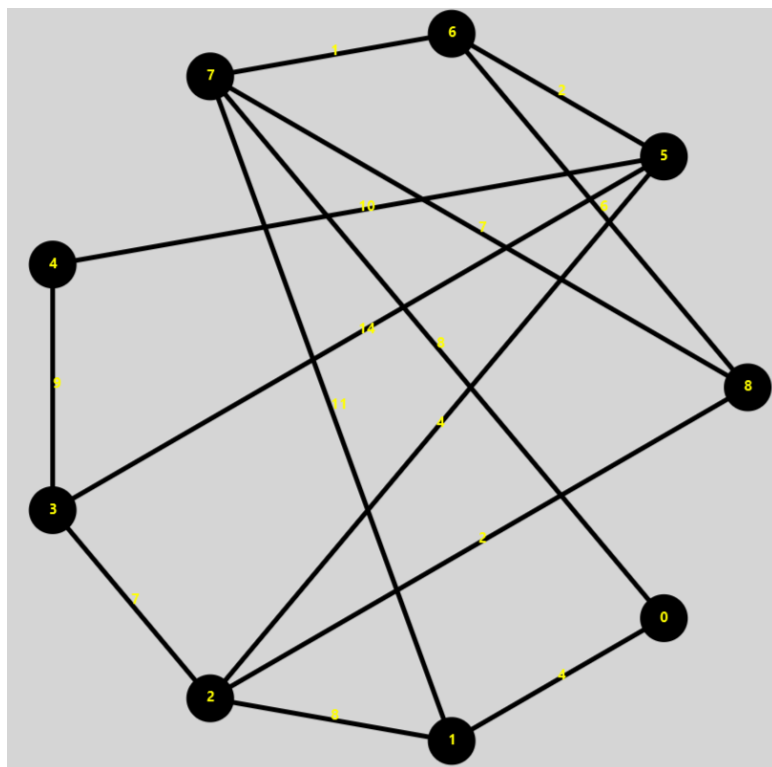


Рисунок 2 - Граф для тестирования stepForward

Результаты тестирования представлены в таблице 3.

Таблица 3 - Результаты тестирования stepForward

№	Номер шага алгоритма	Log	Результат stepForward	Вердикт
1	0	Начало работы алгоритма; Выбрано ребро 7-6 с весом 1	[7 6 1]	passed
2	1	Выбрано ребро 6-5 с весом 2	[6 5 2]	passed
3	5	Ребро 6-8 не выбрано, т.к. образует цикл 6-5-2-8-6	[6 5 2, 5 2 4, 2 8 2, 6 8 6]	passed
4	10	Конец работы алгоритма;	null	passed

3) Тестирование функции stepBack.

Тестирование производилось аналогично stepForward.

Результаты тестирования представлены в таблице 4.

Таблица 4 - Результаты тестирования stepBack

№	Номер шага алгоритма	Log	Результат stepForward	Вердикт
1	0	Шаг назад невозможен;	null	passed
2	1	Ребро 7-6 с весом 1 удалено из остова	[7 6 1]	passed
3	6	Ребро, 6-8 образующее цикл 6-5-2-8-6 удалено из рассмотренных	[6 5 2, 5 2 4, 2 8 2, 6 8 6]	passed
4	7	Ребро 2-3 с весом 7 удалено из остова	[2 3 7]	passed

4) Тестирование функции isConnected

Результаты тестирования функции isConnected представлены в таблице 5.

Таблица 5 - Результаты тестирования функции isConnected

№	Тест	Результат isConnected	Вердикт
1	1 a b 1	true	passed
2	3 a b 1 b c 2 a c 3	true	passed

3	4 a b 1 b c 1 e d 2 d f 3	false	passed
4	6	false	passed

4.2. Тестирование графического интерфейса

1) Тестирование добавления вершин.

Сначала была добавлена вершина с именем “V”. Далее без изменения имени было добавлено ещё 3 вершины. После было добавлено 4 вершины без имени. Результат представлен на рисунке 3. Результат верный. Тест пройден.

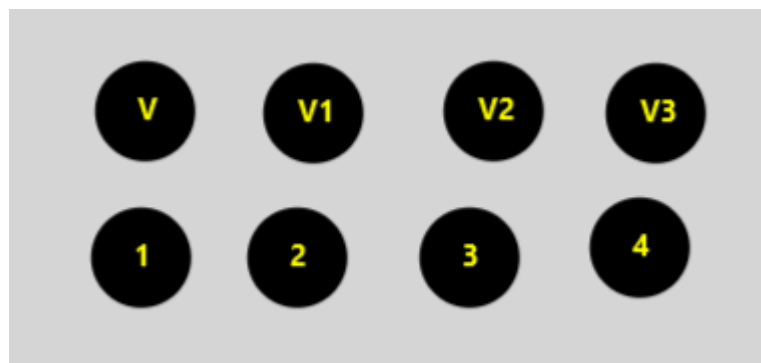


Рисунок 3 - Результат тестирования добавления вершин

2) Тестирование добавления ребер.

Сначала было добавлено ребро с указанием веса 123 между вершинами “V” и “1”. Далее строка ввода веса была очищена и добавлено ещё одно ребро между вершинами “V1” и “2”.

Результат тестирования добавления ребер представлен на рисунке 4. Результат верный. Тест пройден.

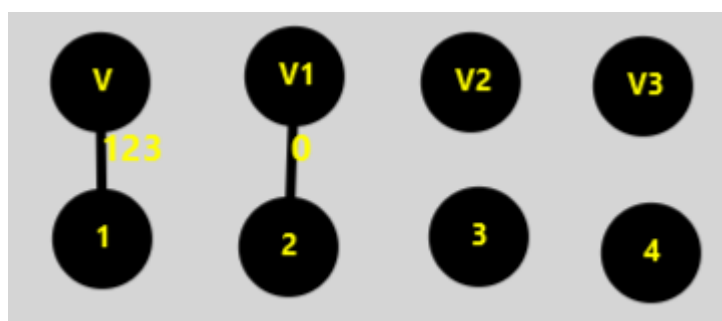


Рисунок 4 - Результат тестирования добавления ребер

Далее была протестирована функция смена веса ребра. Для этого в строке ввода веса был введен вес 213, после чего произведен клик по ребру “V1”-”1”. В результате вес ребра сменился (см. рисунок 5). Тест пройден.

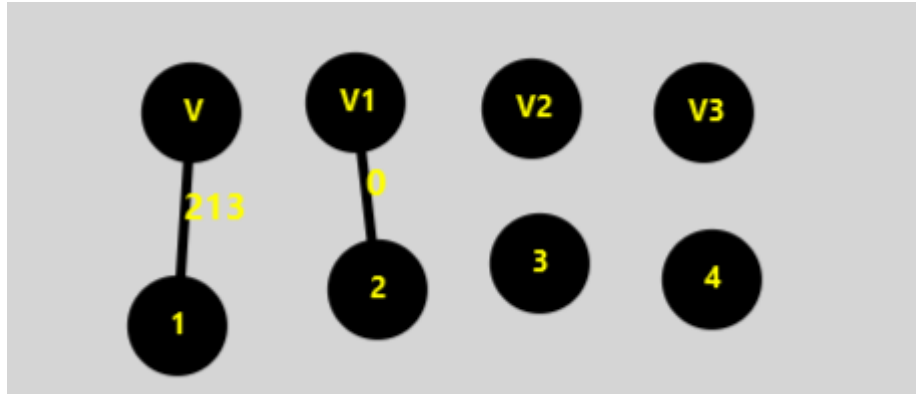


Рисунок 5 - Тестирование функции смены веса ребра

3) Тестирование передвижения элементов.

При активированной кнопке Node вершины были перемещены. Также было отмечено, что при перемещении вершины не накладываются друг на друга. Результат представлен на рисунке 6. Тест пройден.

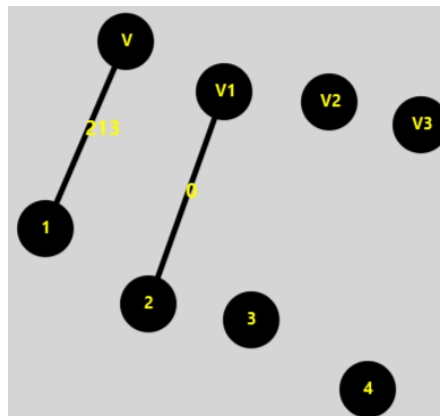


Рисунок 6 - Результат передвижения вершин

4) Тестирование удаления элементов.

При активированной кнопке удаления были произведены нажатия на вершины “V2” и “V1”. При удалении вершины также было удалено ребро инцидентное ей. Результат представлен на рисунке 7. Тест пройден.

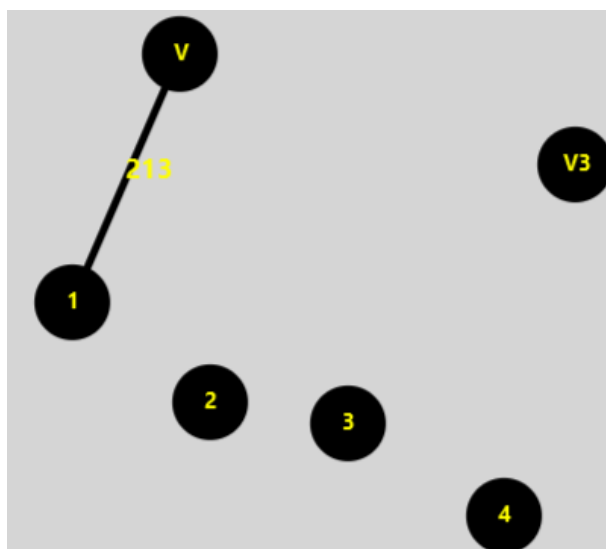


Рисунок 7 - Результат тестирования удаления элементов

5) Тестирование ввода графа из файла.

В файл graph.txt, находящийся в директории с jar-файлом проекта, были добавлены данные о графе. Если файла нет в директории, то открывается окно выбора файла. Содержимое файла представлено на рисунке 8. В результате нажатия кнопки Graph на полотне появилось графическое представление введенного графа. Результат представлен на рисунке 9.

graph.txt – Блокнот

Файл Правка Формат Вид Справка

```

14
0 1 4
1 2 8
2 3 7
3 4 9
0 7 8
7 6 1
6 5 2
5 4 10
1 7 11
2 8 2
2 5 4
3 5 14
7 8 7
6 8 6

```

Рисунок 9 - Содержимое файла graph.txt

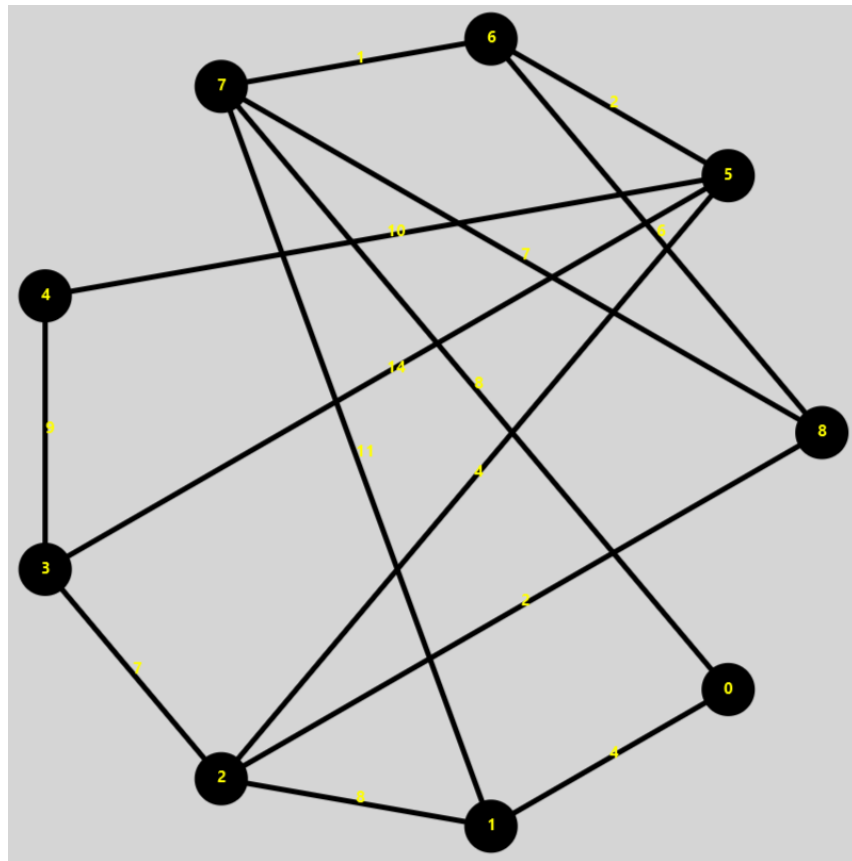


Рисунок 10 - Графическое представление графа из файла graph.txt

6) Тестирование кнопки Clear.

При нажатии кнопки Clear полотно было очищено. Все элементы были удалены. Тест пройден.

7) Тестирование кнопки One step in.

На графе, импортированном из файла graph.txt была опробована кнопка One step in. В результате было выделено цветом нужное ребро (см. рисунок 11). Тест пройден. При образовании цикла ребра, входящие в него, тоже были выделены верно (см. рисунок 12). Если установлена скорость 20, то результат шага отображается мгновенно. Тест пройден.

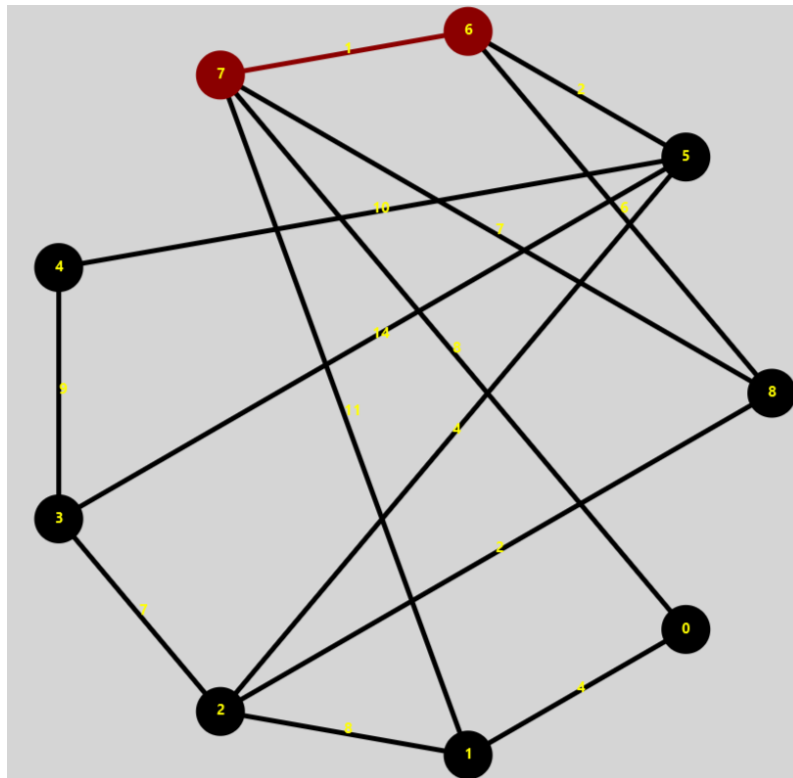


Рисунок 11 - Результат тестирования кнопки One step in

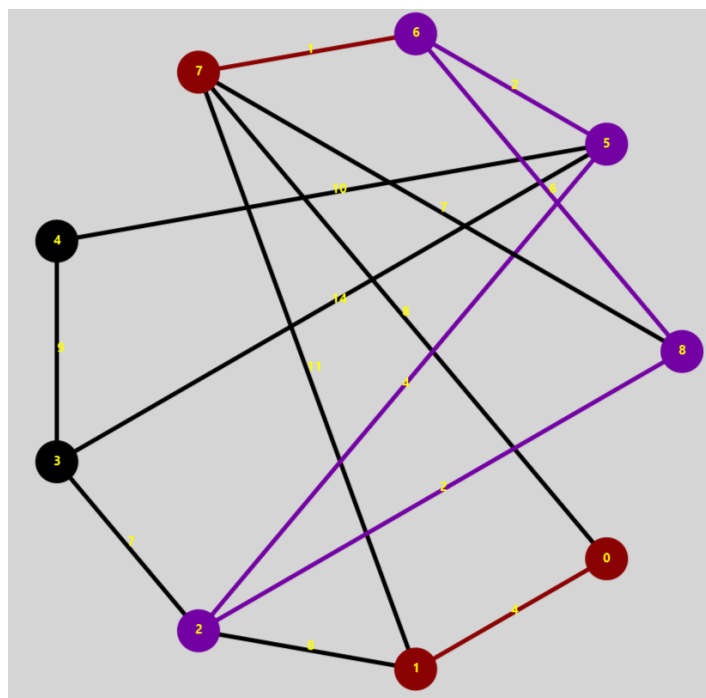


Рисунок 12 - Выделение цикла

8) Тестирование кнопки One step out.

На графе из предыдущего шага был выполнен шаг назад (нажата кнопка One step out). В результате ребро, вошедшее в остов было перекрашено обратно в чёрный цвет (см. рисунок 13). При шаге назад также выделяется цикл (см.

рисунок 14). Если установлена скорость 20, то результат шага отображается мгновенно. Тест пройден.

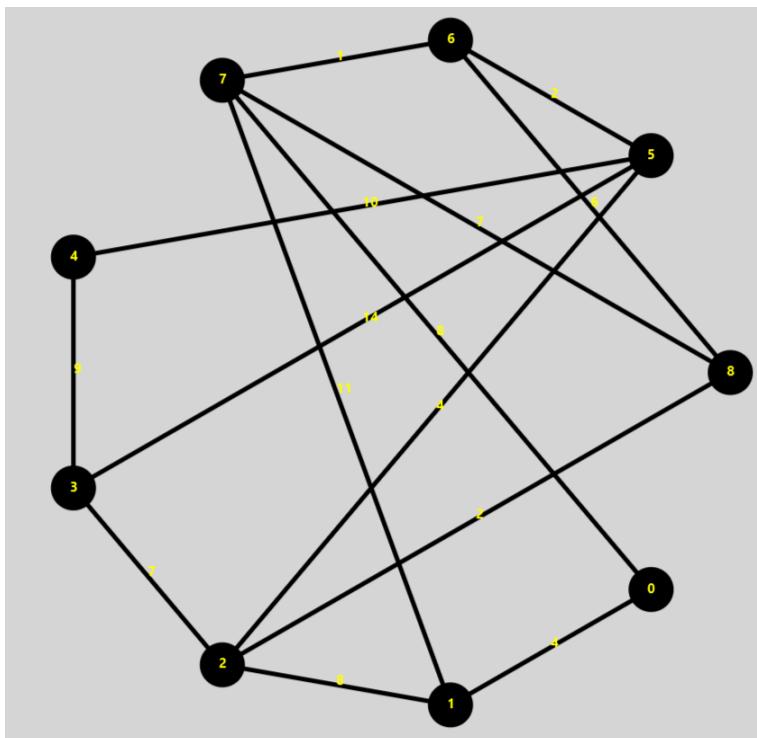


Рисунок 13 - Результат выполнения шага назад

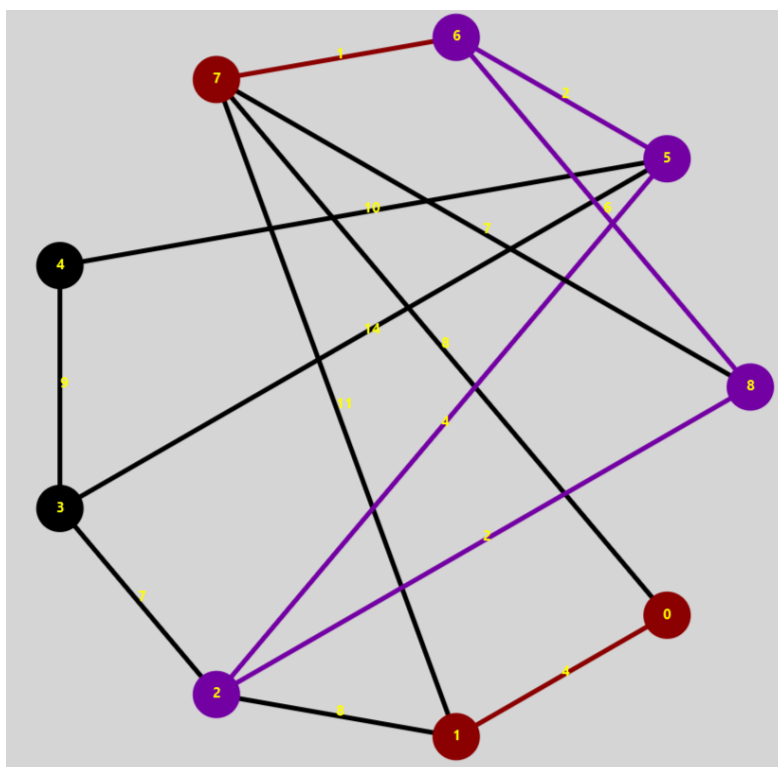


Рисунок 14 - Выделение цикла при шаге назад

9) Тестирование кнопки Full.

При нажатии кнопки Full происходит полное выполнение алгоритма. В результате красным цветом выделено минимальное остовное дерево (см. рисунок 15). При скорости 20 сразу отображается результат. Тест пройден.

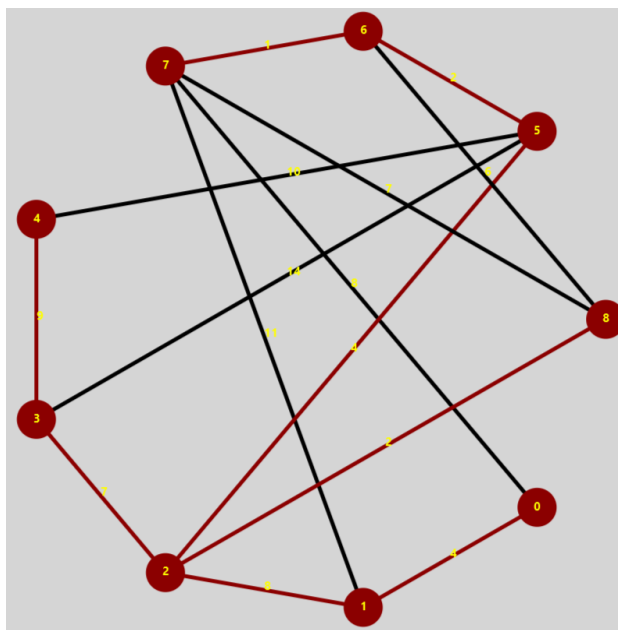


Рисунок 15 - Результат полного выполнения алгоритма

10) Тестирование возвращения к началу работы.

При нажатии кнопки Graph под строкой Initial state все ребра были перекрашены обратно в черный цвет (см. рисунок 16). Тест пройден.

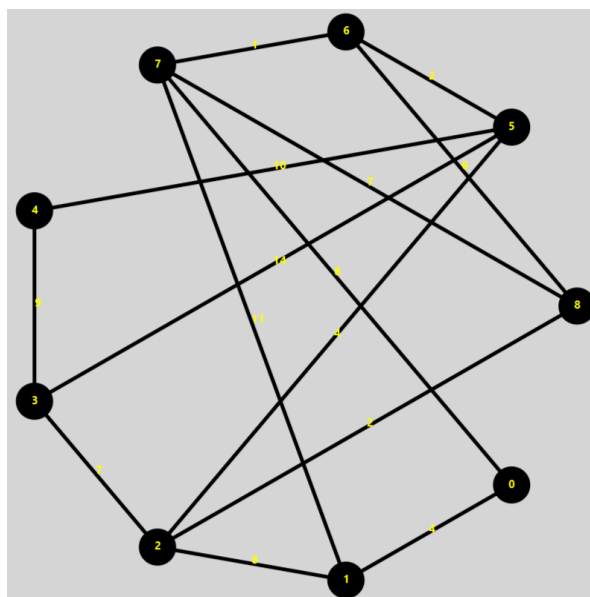


Рисунок 16 - Результат возврата к начальному состоянию

11) Тестирование редактирования графа во время работы алгоритма.

После нескольких шагов вперёд была перемещена вершина “4”. В результате работа алгоритма не остановилась (см. рисунок 17). При удалении вершины “4” работы была остановлена (см. рисунок 18). При удалении ребра “6” - “5” работа также была остановлена (см. рисунок 19). Тест пройден.

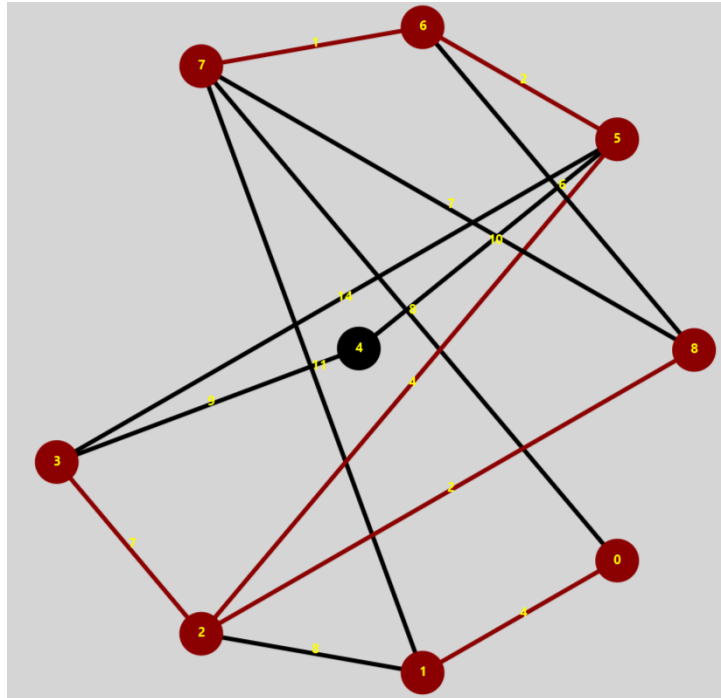


Рисунок 17 - Результат перемещения вершины во время работы алгоритма

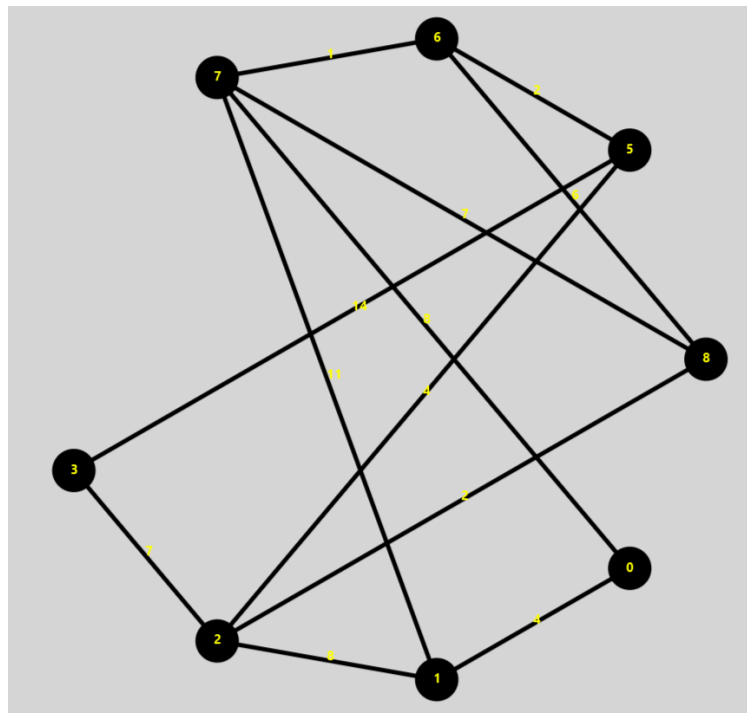


Рисунок 18 - Результат удаления вершины во время работы алгоритма.

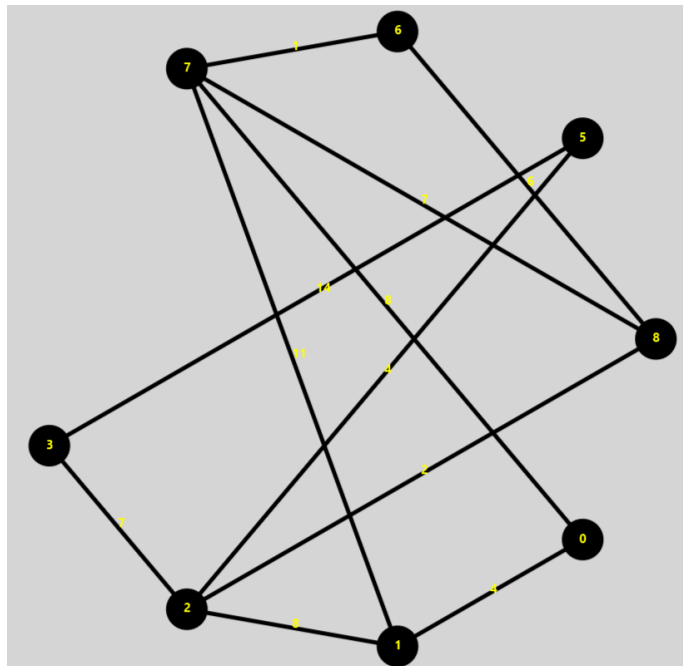


Рисунок 19 - Результат удаления ребра во время работы алгоритма

12) Тестирование показа объяснений.

Во время работы алгоритма после шести шагов вперед, двух шагов назад и нажатия кнопки Full была нажата кнопка Show. В результате в отдельном окне были выведены пояснения к алгоритму (см. рисунок 20). Тест пройден.

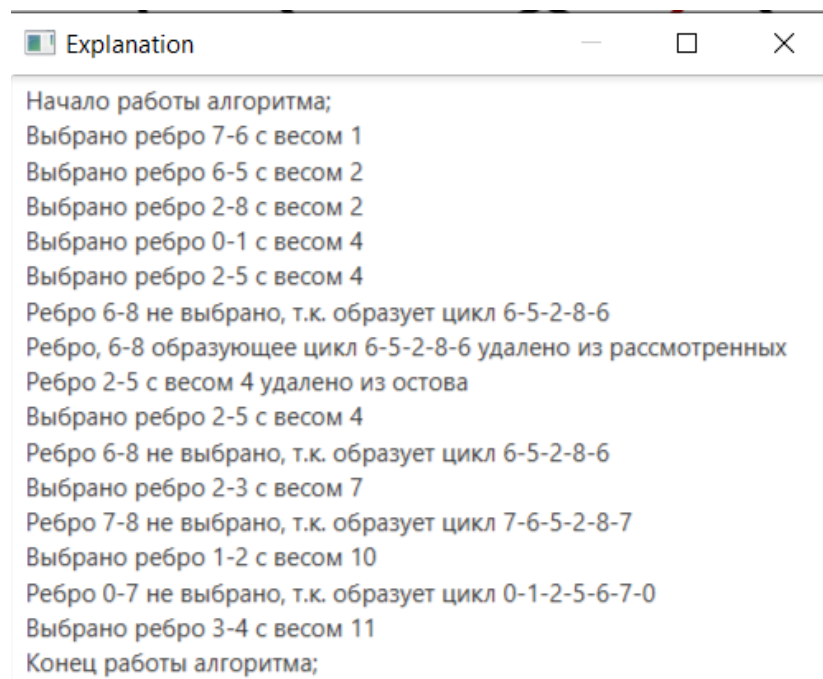
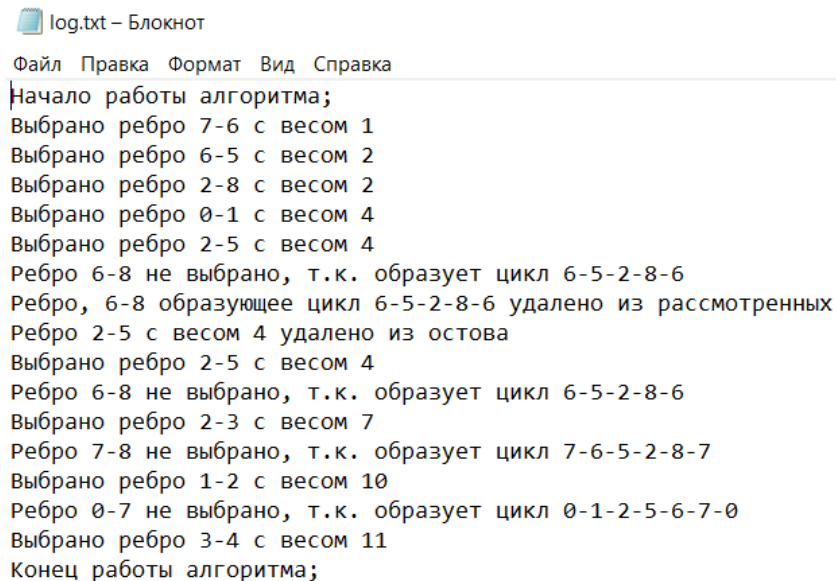


Рисунок 20 - Пояснения к работе алгоритма

13) Тестирование записи объяснений в файл.

После предыдущего шага была нажата кнопка Save. В файл log.txt были записаны объяснения к работе алгоритма (см. рисунок 21). Тест пройден.



```
log.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
Начало работы алгоритма;
Выбрано ребро 7-6 с весом 1
Выбрано ребро 6-5 с весом 2
Выбрано ребро 2-8 с весом 2
Выбрано ребро 0-1 с весом 4
Выбрано ребро 2-5 с весом 4
Ребро 6-8 не выбрано, т.к. образует цикл 6-5-2-8-6
Ребро, 6-8 образующее цикл 6-5-2-8-6 удалено из рассмотренных
Ребро 2-5 с весом 4 удалено из остова
Выбрано ребро 2-5 с весом 4
Ребро 6-8 не выбрано, т.к. образует цикл 6-5-2-8-6
Выбрано ребро 2-3 с весом 7
Ребро 7-8 не выбрано, т.к. образует цикл 7-6-5-2-8-7
Выбрано ребро 1-2 с весом 10
Ребро 0-7 не выбрано, т.к. образует цикл 0-1-2-5-6-7-0
Выбрано ребро 3-4 с весом 11
Конец работы алгоритма;
```

Рисунок 21 - Пояснения, записанные в файл log.txt

14) Тестирование выделения кнопок.

Изначально активна кнопка “Node”. При нажатии на любую другую кнопку, кнопка “Node” перестает выделяться. Выделяется новая активная кнопка. На рисунке 22 показан GUI с активной кнопкой “Node”. На рисунке 23 показан GUI после нажатия на кнопку “Full”.

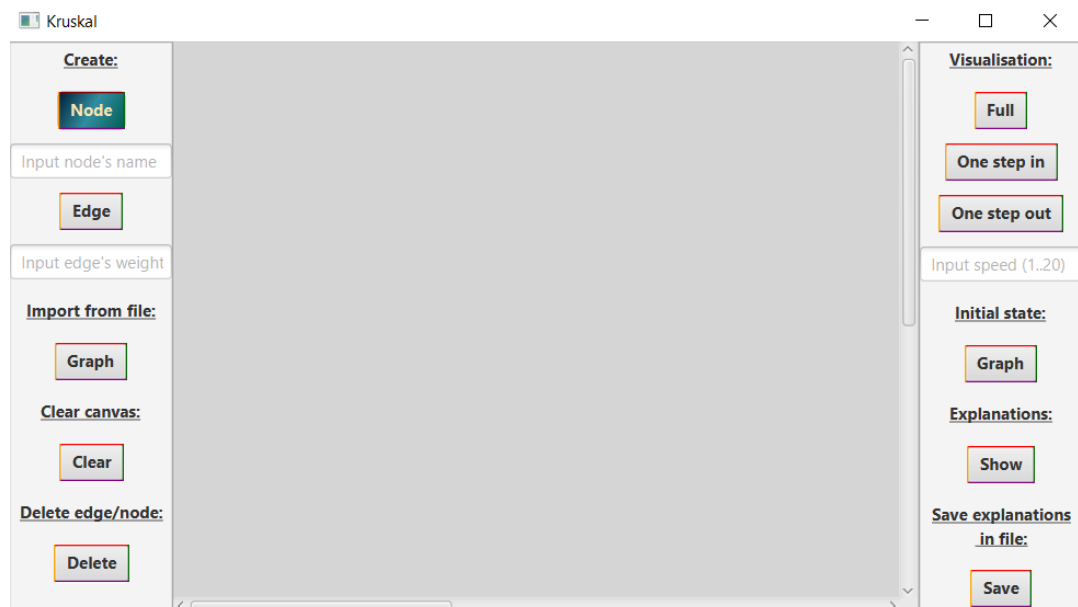


Рисунок 22 - Активная кнопка “Node” в GUI

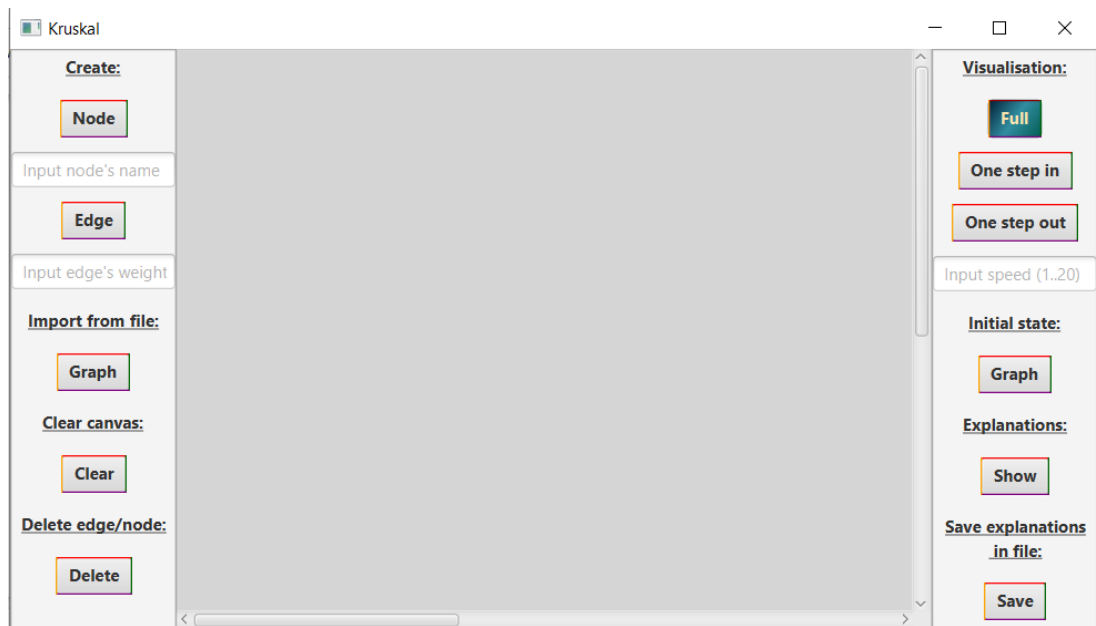


Рисунок 23 - Активная кнопка “Full” в GUI

ЗАКЛЮЧЕНИЕ

В ходе работы были изучены новые языки программирования, а также было разработано графическое приложение для нахождения наименьшего остовного дерева в графа алгоритмом Краскала, причем реализовано оно с помощью языков Java и Kotlin. Все задачи выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация к библиотеке TornadoFX // TornadoFX Guide. URL: <https://edvin.gitbooks.io/tornadofx-guide/content/> (дата обращения: 01.07.2022).
2. Статья по TornadoFX для начинающих // Introduction to TornadoFX. URL: <https://www.baeldung.com/kotlin/tornadofx-intro> (дата обращения: 01.07.2022).
3. Онлайн-курс по языку Kotlin // Введение в Kotlin JVM. URL: <https://stepik.org/course/5448/promo#toc> (дата обращения: 30.06.2022).
4. Онлайн-курс по языку Java // Java. Базовый курс. URL: <https://stepik.org/course/187/promo#toc> (дата обращения: 30.06.2022).
5. Сайт для задавания или поиска вопросов, а также ответов на них // Stack Overflow. URL: <https://stackoverflow.com/> (дата обращения: 2.07.2022).
6. Описание библиотеки JavaFX // Введение в JavaFX. URL: <https://metanit.com/java/javafx/1.1.php> (дата обращения: 5.07.2022).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.kt

```
package com.example

import tornadofx.launch

fun main() {
    launch<MyApp>()
}
```

Название файла: MyApp.kt

```
package com.example

import com.example.view.Interface
import tornadofx.App

class MyApp: App(Interface::class, Styles::class)
```

Название файла: Styles.kt

```
package com.example

import javafx.scene.paint.*
import javafx.scene.text.FontWeight
import tornadofx.*

class Styles : Stylesheet() {
    companion object {
        val forLabels by cssclass()
        val forCanv by cssclass()
        val forActButton by cssclass()
        val forDisButton by cssclass()
        val forStage by cssclass()
    }

    init {
        forLabels {
            padding = box(4.px)
            fontSize = 12.px
            fontWeight = FontWeight.BOLD
            underline = true
        }
        forStage{
        }
        forCanv{
            backgroundColor += c(213, 213, 213, 1.0)
        }
        forActButton{
            fontWeight = FontWeight.EXTRA_BOLD
            borderColor += box(
```

```

        top = javafx.scene.paint.Color.DARKRED,
        right = javafx.scene.paint.Color.DARKGREEN,
        left = javafx.scene.paint.Color.DARKORANGE,
        bottom = javafx.scene.paint.Color.PURPLE
    )
    backgroundColor += LinearGradient(0.0, 0.0, 1.0, 1.0, true,
CycleMethod.REPEAT, Stop(0.0, c(0,26,51,1.0)), Stop(0.5,
c(50,142,161,1.0)), Stop(1.0, c(1,93,82,1.0)))
    textFill = c(254,229,174,1.0)
}
forDisButton{
    fontWeight = FontWeight.EXTRA_BOLD
    borderColor += box(
        top = javafx.scene.paint.Color.RED,
        right = javafx.scene.paint.Color.DARKGREEN,
        left = javafx.scene.paint.Color.ORANGE,
        bottom = javafx.scene.paint.Color.PURPLE
    )
}
}
}

```

Название файла: Interface.kt

```

package com.example.view

import com.example.Styles
import com.example.controller.*
import com.sun.glass.ui.Screen
import tornadofx.*

class Interface : View("Kruskal") {
    val canvasController: CanvasController by inject()
    val mController: MenuController by inject()

    override val root = borderpane {
        addClass(Styles.forStage)
        left<LeftMenu>()
        right<RightMenu>()
        mController.leftMenu = this@borderpane.left
        mController.rightMenu = this@borderpane.right

        center{

this@center.setPrefSize(Screen.getMainScreen().width.toDouble()/2,
Screen.getMainScreen().height.toDouble()/2)

canvasController.canvasHeight.bind(this@center.heightProperty())

canvasController.canvasWidth.bind(this@center.widthProperty())
            scrollpane {
                pane{

setPrefSize(Screen.getMainScreen().width.toDouble()-150,
Screen.getMainScreen().height.toDouble()-80)
                    mController.pane = this

```



```

        left = javafx.scene.paint.Color.DARKGRAY,
        bottom = javafx.scene.paint.Color.DARKGRAY
    )
}
alignment = Pos.TOP_CENTER
spacing = 10.0
maxWidth = 120.0
Platform.runLater {
    label {
        text = "Visualisation:"
        addClass(Styles.forLabels)
    }
    button {
        id = "FULL"
        text = "Full"
        addClass(Styles.forDisButton)
        action {
            mController.btFull()
        }
        tooltip("Visualisation all Kruskal steps")
    }
    button {
        id = "ONE_STEP_IN"
        text = "One step in"
        addClass(Styles.forDisButton)
        action {
            mController.btOneIn()
        }
        tooltip("Visualisation one Kruskal steps forward")
    }
    button {
        id = "ONE_STEP_OUT"
        text = "One step out"
        addClass(Styles.forDisButton)
        action {
            mController.btOneOut()
        }
        tooltip("Visualisation one Kruskal steps back")
    }
    textfield{
        mController.setSpeedText(this)
        id = "weight"
        filterInput { it.controlNewText.isInt() }
        promptText = "Input speed (1..20)"
        tooltip("Input visualisation speed, where 1 - extremely
slow, 20 - instantly")
        action {
            if(text!="") {
                var value = text.toInt()
                if(value < 1) value = 1
                else if(value > 20) value = 10
                text = value.toString()
            }
            else{
                text = "1"
            }
            positionCaret(text.length)
        }
    }
}

```

```

        }
    }
    label {
        text = "Initial state:"
        addClass(Styles.forLabels)
    }
    button {
        id = "GRAPH_START"
        text = "Graph"
        addClass(Styles.forDisButton)
        action {
            Platform.runLater {
                mController.btGraphStart()
            }
        }
        tooltip("Resetting the graph to the beginning")
    }
    label {
        text = "Explanations:"
        addClass(Styles.forLabels)
    }
    button {
        id = "SHOW"
        text = "Show"
        addClass(Styles.forDisButton)
        action {
            mController.btShow()
        }
        tooltip("Click to open window with explanations")
    }
    label {
        text = "Save explanations\n in file:"
        addClass(Styles.forLabels)
        style{
            textAlignment = TextAlignment.CENTER
        }
    }
    button {
        id = "SAVE"
        text = "Save"
        addClass(Styles.forDisButton)
        action {
            mController.btSave()
        }
        tooltip("Click to save explanations in file")
    }
}
}
}

```

Название файла: LeftMenu.kt

```

package com.example.view

import com.example.Styles
import com.example.controller.MenuController
import com.example.controller.NameButton

```

```

import javafx.application.Platform
import javafx.geometry.Pos
import tornadofx.*

class LeftMenu : View("My View") {
    val mController: MenuController by inject()

    override val root = vbox {
        style {
            borderColor += box(
                top = javafx.scene.paint.Color.DARKGRAY,
                right = javafx.scene.paint.Color.DARKGRAY,
                left = javafx.scene.paint.Color.DARKGRAY,
                bottom = javafx.scene.paint.Color.DARKGRAY
            )
        }
        maxWidth = 120.0
        alignment = Pos.TOP_CENTER
        spacing = 10.0
        Platform.runLater {
            label {
                text = "Create:"
                addClass(Styles.forLabels)
            }
            button() {
                id = "NODE"
                text = "Node"
                tooltip("You should click on canvas to create node\n" +
                    "Note: if you click on exist node - you can
move it on canvas")
                addClass(Styles.forActButton)
                action {
                    mController.btClicked(NameButton.NODE)
                }
            }
            textfield{
                mController.setNodeText(this)
                id = "node"
                promptText = "Input node's name"
                tooltip("Input the unique node's name")
            }

            button() {
                id = "EDGE"
                tooltip("You should click on node A and node B to
create edge")
                text = "Edge"
                addClass(Styles.forDisButton)
                action {
                    mController.btClicked(NameButton.EDGE)
                }
            }
            textfield{
                mController.setWeightText(this)
                filterInput { it.controlNewText.isInt() }
                promptText = "Input edge's weight"
                tooltip("Input the edge's weight")
            }
        }
    }
}

```

```

        action {
            if(text!="") text = text.toInt().toString()
            positionCaret(text.length)
        }
    }

    label {
        text = "Import from file:"
        addClass(Styles.forLabels)
    }
    button {
        id = "GRAPH"
        text = "Graph"
        tooltip("Import graph from file \"graph.txt\"")
        addClass(Styles.forDisButton)
        action {
            mController.btGraph()
        }
    }

    label {
        text = "Clear canvas:"
        addClass(Styles.forLabels)
    }
    button {
        id = "CLEAR"
        text = "Clear"
        addClass(Styles.forDisButton)
        action {
            mController.btClear()
        }
    }
    label {
        text = "Delete edge/node:"
        addClass(Styles.forLabels)
    }
    button {
        id = "DELETE"
        text = "Delete"
        tooltip("Click on edge or circle to delete it")
        addClass(Styles.forDisButton)
        action {
            mController.btClicked(NameButton.DELETE)
        }
    }
}
}
}

```

Название файла: CanvasController.kt

```
package com.example.controller
```

```

import javafx.beans.property.SimpleDoubleProperty
import javafx.collections.FXCollections
import javafx.scene.Group
import javafx.scene.Node

```

```

import javafx.scene.layout.Pane
import javafx.scene.paint.Color
import javafx.scene.paint.Paint
import javafx.scene.robot.Robot
import javafx.scene.shape.Circle
import javafx.scene.shape.Line
import javafx.scene.text.FontWeight
import javafx.scene.text.Text
import tornadofx.*
import java.lang.Math.abs
import java.lang.Math.pow

class CanvasController: Controller() {
    val mController: MenuController by inject()
    val names = mutableListOf<String>()
    val nodes = mutableListOf<Group>()
    val edges = mutableListOf<Line>()
    val tempEdges = mutableListOf<Line>()
    val tempTexts = mutableListOf<Text>()
    var tempNode: Group? = null
    lateinit var pane: Pane

    val black = c(0,0,0,1.0)
    val gray = c(50,143,156,1.0)
    val red = c(139,0,0,1.0)
    val violent = c(114,0,163,1.0)
    var canvasWidth = SimpleDoubleProperty()
    var canvasHeight = SimpleDoubleProperty()
    var canvasMaxWidth = SimpleDoubleProperty()
    var canvasMaxHeight = SimpleDoubleProperty()
    var dragFlag = false

    fun setThisPane(pane: Pane){
        this.pane = pane
    }
    fun checkEdge(id1: String, id2: String): Boolean{
        val name1 = id1+"_"+id2
        val name2 = id2+"_"+id1
        edges.forEach{
            if(it.id == name1 || it.id == name2) return false
        }
        return true
    }

    fun addCircle(x: Double, y: Double, nodeRadius: Double, currText:
String): Boolean{
        if((x-nodeRadius) <=0 || (x+nodeRadius) >= canvasMaxWidth.value
|| (y - nodeRadius) <= 0 || (y+nodeRadius) >= canvasMaxHeight.value)
return false
        nodes.forEach {
            if(Math.sqrt(pow(it.layoutX-x,2.0) +
pow(it.layoutY-y,2.0))<=2*nodeRadius) return false
        }
        names.add(currText)
        mController.pane.group {
            id = currText
            layoutX = x

```

```

        layoutY = y
        nodes.add(this@group)
        circle {
            id = currText
            centerX = 0.0
            centerY = 0.0
            radius = nodeRadius
            fill = black
        }
        text{
            this.x = 0.0 - currText.length*3
            this.y = 0.0 + 3
            text = currText
            fill = Color.YELLOW
            style{
                fontWeight = FontWeight.BOLD
            }
        }
        mController.btGraphStart(mController.actBt)

        setOnMousePressed {
            if(mController.actBt == NameButton.NODE){
                for(edge in edges){
                    val ids = edge.id.split("_")
                    if(this@group.id in ids){
                        tempEdges.add(edge)
                        tempTexts.add(pane.children.find{ it is
Text && it.id == edge.id } as Text)
                    }
                }
                tempEdges.forEach{ it.isVisible = false }
                tempTexts.forEach{ it.isVisible = false }
            }
        }
        setOnMouseDragged {
            if(mController.actBt == NameButton.NODE){
                if(this@group.layoutX+it.x+nodeRadius <
canvasMaxWidth.value && this@group.layoutX+it.x-nodeRadius > 0 &&
                    this@group.layoutY+it.y+nodeRadius <
canvasMaxHeight.value && this@group.layoutY+it.y-nodeRadius > 0){
                    var flagCollision = true
                    for(n in nodes){
                        if(n != this@group){
                            if(pow(this.layoutX+it.x-n.layoutX,2.0)+pow(this.layoutY+it.y-n.layoutY
,2.0) <= pow(2*nodeRadius,2.0)+1){
                                flagCollision = false
                            }
                        }
                    }
                    if(flagCollision){
                        this@group.layoutX += it.x - 0.7
                        this@group.layoutY += it.y - 0.7
                    }
                }
                dragFlag = true
            }
        }
    }
}

```

```

    }
    setOnMouseReleased {
        if(mController.actBt == NameButton.DELETE){
            val tempEdges = mutableListOf<Line>()
            for(edge in edges){
                if(this@group.id in edge.id.split("_")){
                    tempEdges.add(edge)
                }
            }
            tempEdges.forEach{ l ->
                pane.children.remove(pane.children.find { it is
Text && it.id == l.id })
            }
            edges.removeAll(tempEdges)
            pane.children.removeAll(tempEdges)
            pane.children.remove(this@group)
            names.remove(this@group.id)
            nodes.remove(this@group)
            tempEdges.clear()
            mController.btGraphStart(mController.actBt)
        }
        if(mController.actBt == NameButton.EDGE){
            if(tempNode == null) tempNode = this@group
            else if(tempNode != this@group){
                val gr1 = tempNode!!
                val gr2 = this@group
                if(gr1.id != gr2.id && checkEdge(gr1.id,
gr2.id)) {
                    pane.line {
                        id = gr1.id + "_" + gr2.id
                        startX = gr1.layoutX
                        startY = gr1.layoutY
                        endX = gr2.layoutX
                        endY = gr2.layoutY
                        strokeWidth = mController.lineThick
                        stroke = black
                        edges.add(this@line)
                        setOnMouseReleased {
                            if(mController.actBt ==
NameButton.EDGE){
                                val weight =
mController.weightTextField.text
                                (parent.getChildList()?.find {
it is Text && it.id == this.id } as Text).text = if (weight == "") "0"
                                else weight
                                mController.btGraphStart(mController.actBt)
                            }
                            if(mController.actBt ==
NameButton.DELETE){
                                parent.getChildList()?.remove(
                                    parent.getChildList()?.find
{ it is Text && it.id == this.id })
                                edges.remove(this@line)
                                parent.getChildList()?.remove(this@line)

```



```

mController.btGraphStart(mController.actBt)
    }

mController.btGraphStart(mController.actBt)
    }
}

val weight =
mController.weightTextField.text
    pane.text {
        id = gr1.id + "_" + gr2.id
        layoutX = (gr1.layoutX + gr2.layoutX) /
2
        layoutY = (gr1.layoutY + gr2.layoutY) /
2
        if(mController.actBt ==
NameButton.EDGE) {
            text = if (weight == "") "0" else
weight
mController.btGraphStart(mController.actBt)
            }
            fill = Color.YELLOW

            style {
                fontSize = 14.px
                fontWeight = FontWeight.EXTRA_BOLD
            }
        }
        tempNode = null
        mController underline()
    }
}
if(mController.actBt == NameButton.NODE){
    tempEdges.forEach{
        val ids = it.id.split("_")
        if(this@group.id == ids[0]){
            it.startX = layoutX
            it.startY = layoutY
        }
        else{
            it.endX = layoutX
            it.endY = layoutY
        }
        val textEdge = tempTexts.find{t -> t.id ==
it.id }!!

        textEdge.layoutX = (it.startX + it.endX)/2
        textEdge.layoutY = (it.startY + it.endY)/2
        textEdge.isVisible = true
        it.isVisible = true
    }
    tempEdges.clear()
    tempTexts.clear()
}
}
}

```

```

        return true
    }

    fun clearAllObject(){
        nodes.clear()
        edges.clear()
        names.clear()
    }

    fun addGraphEdge(id1: String, id2: String, weight: Int){
        val gr1 = nodes.find{it.id == id1}
        val gr2 = nodes.find{it.id == id2}
        if(gr1 == null || gr2 == null) return
        pane.line {
            id = gr1.id + "_" + gr2.id
            startX = gr1.layoutX
            startY = gr1.layoutY
            endX = gr2.layoutX
            endY = gr2.layoutY
            strokeWidth = mController.lineThick
            stroke = black
            edges.add(this@line)
            setOnMouseReleased {
                if(mController.actBt == NameButton.EDGE){
                    val weight2 = mController.weightTextField.text
                    (parent.getChildList()?.find { it is Text && it.id
== this.id } as Text).text = if (weight2 == "") "0" else weight2
                }
                if(mController.actBt == NameButton.DELETE){
                    parent.getChildList()?.remove(
                        parent.getChildList()?.find { it is Text &&
it.id == this.id })
                    edges.remove(this@line)
                    parent.getChildList()?.remove(this@line)
                }
                mController.btGraphStart(mController.actBt)
            }
        }
        pane.text {
            id = gr1.id + "_" + gr2.id
            layoutX = (gr1.layoutX + gr2.layoutX) / 2
            layoutY = (gr1.layoutY + gr2.layoutY) / 2
            if(mController.actBt == NameButton.NODE){
                text = tempTexts.find { it.id.contains(gr1.id) &&
it.id.contains(gr2.id)}!!.text
            }
            else if(mController.actBt == NameButton.EDGE ||
mController.actBt == NameButton.GRAPH) {
                text = weight.toString()
            }
            mController.btGraphStart(mController.actBt)
            fill = Color.YELLOW
            style {
                fontWeight = FontWeight.BOLD
            }
        }
        mController underline()
    }

```

```

    }

    fun aloneNode(): Boolean{
        for(n in nodes){
            if(edges.find { n.id in it.id.split("_") }==null) return
true
        }
        return false
    }
}

```

Название файла: MenuController.kt

```

package com.example.controller

import com.example.Styles
import com.example.view.ExplainView
import javafx.animation.PauseTransition
import javafx.application.Platform
import javafx.beans.property.SimpleBooleanProperty
import javafx.beans.property.SimpleStringProperty
import javafx.scene.Group
import javafx.scene.Node
import javafx.scene.control.Alert
import javafx.scene.control.Button
import javafx.scene.control.TextField
import javafx.scene.layout.Pane
import javafx.scene.paint.Paint
import javafx.scene.shape.Circle
import javafx.scene.shape.Line
import javafx.scene.text.Text
import javafx.stage.FileChooser
import javafx.util.Duration
import kraskal.Edge
import kraskal.Kruskal
import tornadofx.*
import java.io.File
import java.lang.Double.min
import java.nio.file.Paths

enum class NameButton{
    NODE,
    EDGE,
    GRAPH,
    GRAPH_START,
    CLEAR,
    DELETE,
    FULL,
    ONE_STEP_IN,
    ONE_STEP_OUT,
    SHOW,
    SAVE
}

class MenuController: Controller() {
    val canvasController: CanvasController by inject()

```

```

val fileChooser = FileChooser()
lateinit var pane: Pane
lateinit var rightMenu: Node
lateinit var leftMenu: Node
lateinit var nodeTextField: TextField
lateinit var weightTextField: TextField
lateinit var speedTextField: TextField
var algKruskal: Kruskal? = null
var actBt: NameButton = NameButton.NODE
                                val btState =
Array<SimpleBooleanProperty>(11,{SimpleBooleanProperty()})
    var explainText = SimpleStringProperty("")
    var flagExpClose = true
    var flagOstov = false
    var lineThick = 4.0
    var nodeRadius = 20.0

    fun setNodeText(textField: TextField){
        nodeTextField = textField
    }
    fun setWeightText(textField: TextField){
        weightTextField = textField
    }
    fun setSpeedText(textField: TextField){
        speedTextField = textField
    }
    fun btClicked(nameButton: NameButton){
        leftMenu.getChildList()!!.find { it is Button && it.id ==
actBt.toString() }?.removeClass(Styles.forActButton)
        rightMenu.getChildList()!!.find { it is Button && it.id ==
actBt.toString() }?.removeClass(Styles.forActButton)
        leftMenu.getChildList()!!.find { it is Button && it.id ==
actBt.toString() }?.addClass(Styles.forDisButton)
        rightMenu.getChildList()!!.find { it is Button && it.id ==
actBt.toString() }?.addClass(Styles.forDisButton)

        btState.forEach { state -> state.value = false }
        btState[nameButton.ordinal].value = true

        (leftMenu.getChildList()!!.find { it is Button && it.id ==
nameButton.toString() }?.removeClass(Styles.forDisButton))
        (rightMenu.getChildList()!!.find { it is Button && it.id ==
nameButton.toString() }?.removeClass(Styles.forDisButton))
        (leftMenu.getChildList()!!.find { it is Button && it.id ==
nameButton.toString() }?.addClass(Styles.forActButton))
        (rightMenu.getChildList()!!.find { it is Button && it.id ==
nameButton.toString() }?.addClass(Styles.forActButton))

        actBt = nameButton
        canvasController.tempNode = null
        printState()
    }

    fun btNode(x: Double, y: Double){
        btClicked(NameButton.NODE)
        var currText = nodeTextField.text

```

```

        if(currText in canvasController.names || currText==""){
            var ind = 1
            while(currText+ind.toString() in canvasController.names){
                ind++
            }
            currText += ind.toString()
        }
        canvasController.addCircle(x, y, nodeRadius, currText)
    }

    fun underLine(){
        val tempOther = mutableListOf<Node>()
        for(i in pane.getChildList()){
            if(!(i is Line)) tempOther.add(i)
        }
        pane.getChildList()?.removeAll(tempOther)
        pane.getChildList()?.addAll(tempOther)
        tempOther.clear()
    }

    var defaultGraphPath =
Paths.get("").toAbsolutePath().toString()+"/graph.txt"

    fun btGraph(){
        btClicked(NameButton.GRAPH)
        val nodeNames = mutableListOf<String>()
        val listString = mutableListOf<String>()

        if (File(defaultGraphPath).exists())
            File(defaultGraphPath).readLines().forEach{
                btClear()
                btClicked(NameButton.GRAPH)
                listString.add(it)
                if (it.contains(" "))
                {
                    val tmplist = it.split(" ")
                    for (i in 0..1){
                        if (!(tmplist[i] in nodeNames))
                            nodeNames.add(tmplist[i])
                    }
                }
            }
        else
        {
            try {
                defaultGraphPath =
fileChooser.showOpenDialog(primaryStage).path
                File(defaultGraphPath).readLines().forEach{
                    btClear()
                    btClicked(NameButton.GRAPH)
                    listString.add(it)
                    if (it.contains(" "))
                    {
                        val tmplist = it.split(" ")
                        for (i in 0..1){
                            if (!(tmplist[i] in nodeNames))
                                nodeNames.add(tmplist[i])
                        }
                    }
                }
            }
        }
    }

```

```

        }
    }
}
} catch(e: Exception){
    println(e)
}
}

        val radius = min(canvasController.canvasWidth.value/2,
canvasController.canvasHeight.value/2) - 100
        if(radius <= 40) return
        val centerX = (canvasController.canvasWidth-260).value/2
        val centerY = canvasController.canvasHeight.value/2
        for (i in 1..nodeNames.size){
            val angle = 2*Math.PI*i/nodeNames.size

canvasController.addCircle(centerX+radius*Math.cos(angle),centerY+radius*Math.sin(angle),20.0, nodeNames[i-1])
        }
        for (str in listString){
            if (str.contains(" "))
            {
                val tmplist = str.split(" ")
                try{

canvasController.addGraphEdge(tmplist[0],tmplist[1],tmplist[2].toInt())
                    }catch(e: Exception){
                        println(e)
                    }
                }
            }
        }

fun btClear(){
    btClicked(NameButton.CLEAR)
    canvasController.clearAllObject()
    algKruskal = null
    flagOstov = false
    pane.getChildList()?.removeAll(pane.children)
}

                                var                                defaultLogPath                                =
Paths.get("").toAbsolutePath().toString()+"/log.txt"

fun btShow(){
    btClicked(NameButton.SHOW)
    if(flagExpClose){
        flagExpClose = false
        ExplainView().openWindow()
    }
}

fun btSave() {
    btClicked(NameButton.SAVE)
    if(File(defaultLogPath).exists()){
        File(defaultLogPath).writeText(explainText.value)
    }
}

```

```

        else{
            try {
                val temp = defaultLogPath
                defaultLogPath =
fileChooser.showOpenDialog(primaryStage).path
                File(defaultLogPath).writeText(explainText.value)
                File(defaultLogPath).writeText(temp)
            } catch(e: Exception){
                println(e)
            }
        }
    }

fun initKruskal(){
    val canvEdges = canvasController.edges
    val edges = mutableListOf<Edge>()
    if(canvEdges.size == 0){
        println(1)
        notConnectedGraph()
        return
    }
    canvEdges.forEach{ edge ->
        val gr = edge.id.split("_")
        val weight = (pane.children.find { it is Text && it.id ==
edge.id } as Text).text
        edges.add(Edge(gr[0], gr[1], weight.toInt()))
    }
    algKruskal = Kruskal(edges)
    if(!algKruskal!!.isConnected || canvasController.aloneNode()){
        println((!algKruskal!!.isConnected).toString() +
canvasController.aloneNode().toString())
        notConnectedGraph()
        algKruskal = null
        return
    }
    algKruskal!!.run()
}

fun visDisable(value: Boolean){
    pane.isDisable = value
    (rightMenu.getChildList()!!!.find { it is Button && it.text ==
"Full" } as Button).isDisable = value
    (rightMenu.getChildList()!!!.find { it is Button && it.text ==
"Full" } as Button).isDisable = value
    (rightMenu.getChildList()!!!.find { it is Button && it.text ==
"One step in" } as Button).isDisable = value
    (rightMenu.getChildList()!!!.find { it is Button && it.text ==
"One step out" } as Button).isDisable = value
    (rightMenu.getChildList()!!!.find { it is Button && it.text ==
"Graph" } as Button).isDisable = value
    leftMenu.isDisable = value
}

fun visualisationSpeed(): Double{
    var speed = if(speedTextField.text == "") 1.0 else
speedTextField.text.toDouble()
    speed = when(speed) {

```

```

        20.0 -> 0.0
        19.0 -> 50.0
        18.0 -> 100.0
        17.0 -> 500.0
        16.0 -> 1000.0
        in 1.0..15.00 -> (17.0-speed)*1000
        else -> 1000.0
    }
    return speed
}

fun btFull(){
    btClicked(NameButton.FULL)
    Thread{
        while (!flagOstov){
            visDisable(true)
            if(actBt != NameButton.ONE_STEP_IN && actBt !=
NameButton.FULL) break
            if(!btOneIn()) break
            Thread.sleep(visualisationSpeed().toLong())
        }
        visDisable(false)
        btClicked(NameButton.FULL)
    }.start()
}

fun btOneIn(): Boolean{
    if(actBt != NameButton.FULL) btClicked(NameButton.ONE_STEP_IN)
    if(actBt != NameButton.ONE_STEP_IN && actBt != NameButton.FULL)
return false
    if(algKruskal == null && !flagOstov) {
        initKruskal()
    }
    if(flagOstov || algKruskal == null) return false
    else{
        var x: List<Edge?>? = ArrayList()
        x = algKruskal!!.stepForward()
        if(x == null){
            flagOstov = true
            explainText.value=algKruskal!!.log
            return false
        }
        visDisable(true)
        if(x.size >= 1) {
            val edgeToRecolor = mutableListOf<Line>()
            val circleToRecolor = mutableListOf<Circle>()
            val circleColor = mutableListOf<Paint>()
            val edgeColor = mutableListOf<Paint>()
            x.forEach { edge ->
                val line = (pane.getChildList()?.find { it is Line
&& it.id == edge.start+"_"+edge.end }
                ?: (pane.getChildList()?.find { it is Line &&
it.id == edge.end+"_"+edge.start })) as Line
                edgeToRecolor.add(line)
                edgeColor.add(line.stroke)
                val gr1 = (pane.getChildList()?.find { it is Group
&& it.id == line.id.split("_")[0] } as Group

```



```

        val gr2 = (pane.getChildList()?.find { it is Group
&& it.id == line.id.split("_")[1]} as Group)
        if((gr1.children[0] as Circle) !in circleToRecolor)
        {
            circleToRecolor.add((gr1.children[0] as
Circle))
            circleColor.add((gr1.children[0] as
Circle).fill)
        }
        if((gr2.children[0] as Circle) !in circleToRecolor)
        {
            circleToRecolor.add((gr2.children[0] as
Circle))
            circleColor.add((gr2.children[0] as
Circle).fill)
        }
    }
    when(x.size) {
        1 -> {
            canvasController.gray}
            edgeToRecolor.forEach{it.stroke =
            canvasController.gray}
            circleToRecolor.forEach{it.fill =
            canvasController.gray}
        }
        else ->{
            canvasController.violent}
            edgeToRecolor.forEach{it.stroke =
            canvasController.violent}
            circleToRecolor.forEach{it.fill =
            canvasController.violent}
        }
    }

    val pt =
    PauseTransition(Duration.millis(visualisationSpeed()))
    pt.setOnFinished {
        if(x.size > 1){
            for(i in 0..edgeToRecolor.size-1){
                edgeToRecolor[i].stroke = edgeColor[i] }
            for(i in 0..circleToRecolor.size-1){
                circleToRecolor[i].fill = circleColor[i] }
        } else{
            edgeToRecolor.forEach{it.stroke =
            canvasController.red}
            circleToRecolor.forEach{it.fill =
            canvasController.red}
        }
        circleColor.clear()
        edgeColor.clear()
        circleToRecolor.clear()
        edgeToRecolor.clear()
    }
    if(actBt == NameButton.ONE_STEP_IN)
    visDisable(false)
    pt.play()
}
}
explainText.value=algKruskal!!.log
return true

```

```

    }

    fun btOneOut() {
        btClicked(NameButton.ONE_STEP_OUT)
        if (algKruskal == null)
            return
        if (flagOstov)
            flagOstov = false
        val x = algKruskal!!.stepBack()
        if(x == null){
            explainText.value=algKruskal!!.log
            algKruskal = null
            return
        }
        visDisable(true)
        if(x.size >= 1) {
            val edgeToRecolor = mutableListOf<Line>()
            val circleToRecolor = mutableListOf<Circle>()
            val circleColor = mutableListOf<Paint>()
            val edgeColor = mutableListOf<Paint>()
            x.forEach { edge ->
                val line = (pane.getChildList()?.find { it is Line &&
it.id == edge.start+"_"+edge.end }
                ?: (pane.getChildList()?.find { it is Line && it.id
== edge.end+"_"+edge.start })) as Line
                edgeToRecolor.add(line)
                edgeColor.add(line.stroke)
                val gr1 = (pane.getChildList()?.find { it is Group &&
it.id == line.id.split("_")[0]} as Group)
                val gr2 = (pane.getChildList()?.find { it is Group &&
it.id == line.id.split("_")[1]} as Group)
                if((gr1.children[0] as Circle) !in circleToRecolor) {
                    circleToRecolor.add((gr1.children[0] as Circle))
                    circleColor.add((gr1.children[0] as Circle).fill)
                }
                if((gr2.children[0] as Circle) !in circleToRecolor) {
                    circleToRecolor.add((gr2.children[0] as Circle))
                    circleColor.add((gr2.children[0] as Circle).fill)
                }
            }
            when(x.size){
                1 -> {
                    edgeToRecolor.forEach{it.stroke =
canvasController.gray}
                    circleToRecolor.forEach{it.fill =
canvasController.gray}
                }
                else ->{
                    edgeToRecolor.forEach{it.stroke =
canvasController.violent}
                    circleToRecolor.forEach{it.fill =
canvasController.violent}
                }
            }
        }
        val pt =
PauseTransition(Duration.millis(visualisationSpeed()))
        pt.setOnFinished {

```

```

        if(x.size > 1){
            for(i in 0..edgeToRecolor.size-1){
                edgeToRecolor[i].stroke = edgeColor[i]
            }
            for(i in 0..circleToRecolor.size-1){
                circleToRecolor[i].fill = circleColor[i]
            }
        }
    }
    else{
        edgeToRecolor.forEach{it.stroke =
canvasController.black}
        circleToRecolor.forEach{ circle ->
            var redFlag = false
            for(node in pane.children){
                if(node is Line && circle.id in
node.id.split("_")){
                    if(node.stroke ==
canvasController.red){
                        redFlag = true
                        break
                    }
                }
            }
            if(redFlag) circle.fill = canvasController.red
            else circle.fill = canvasController.black
        }
    }
    circleColor.clear()
    edgeColor.clear()
    circleToRecolor.clear()
    edgeToRecolor.clear()
    visDisable(false)
}
pt.play()
}
explainText.value=algKruskal!!.log
}

fun btGraphStart(bt: NameButton = NameButton.GRAPH_START){
    btClicked(bt)
    explainText.value = ""
    flagOstov = false
    algKruskal = null
    colorInBlack()
}

fun colorInBlack(){
    for(i in pane.children){
        if(i is Line) i.stroke = canvasController.black
        if(i is Group){
            for(j in i.children){
                if(j is Circle) j.fill = canvasController.black
            }
        }
    }
}
}

```

```

fun notConnectedGraph() {
    Platform.runLater {
        alert(type = Alert.AlertType.INFORMATION,
            title = "Kruskal",
            content = "Graph should be connected",
            header = "")
    }
}

fun printState() {
    println(btState.joinToString())
    println(actBt)
}
}

```

Название файла: Edge.java

```

package kraskal;

import org.jetbrains.annotations.NotNull;

public class Edge implements Comparable<Edge>{
    private final String start;
    private final String end;
    private final int weight;

    public Edge(String start, String end, int weight) {
        this.start = start;
        this.end = end;
        this.weight = weight;
    }

    @Override
    public int compareTo(@NotNull Edge o) {
        if (this.weight == o.weight) return 0;
        return (this.weight > o.weight) ? 1: -1;
    }

    public String getStart() {
        return start;
    }

    public String getEnd() {
        return end;
    }

    public int getWeight() {
        return weight;
    }

    @Override
    public String toString() {
        return start + ' ' + end + ' ' + weight;
    }
}

```

Название файла: Kruskal.java

```
package kraskal;

import org.jetbrains.annotations.NotNull;

import javax.sound.sampled.AudioFormat;
import java.util.*;
import java.util.stream.Collectors;

public class Kruskal {
    private final List<Edge> edges;
    private List<Edge> mst;
    private final DisjointSets<String> vertices;
    private List<List<Edge>> steps;
    private int i = 0;
    private final StringBuilder log = new StringBuilder();

    public Kruskal(Collection<Edge> edges) {
        this.edges = new LinkedList<>(edges);
        Collections.sort(this.edges);

        Set<String> v = new HashSet<>();
        for (Edge e : edges) {
            v.add(e.getEnd());
            v.add(e.getStart());
        }
        this.vertices = new DisjointSets<>(v);
    }

    public boolean isConnected() {
        DisjointSets<String> ds = new
        DisjointSets<>(vertices.getParents().keySet());
        for (Edge e : edges) ds.unite(e.getStart(), e.getEnd());
        String parent = ds.find(edges.get(0).getStart());
        for (String p : ds.getParents().keySet()) if
        (!ds.find(p).equals(parent)) return false;
        return true;
    }

    public void run() {
        this.mst = new LinkedList<>();
        steps = new ArrayList<>();
        while (!(mst.size() == vertices.size() - 1)) {
            Edge edge = edges.remove(0);
            String start = edge.getStart(), end = edge.getEnd();
            if (vertices.find(start).equals(vertices.find(end))) {
                steps.add(getCycle(edge));
                continue;
            }
            mst.add(edge);
            vertices.unite(start, end);
            List<Edge> res = new ArrayList<>();
            res.add(edge);
            steps.add(res);
        }
    }
}
```

```

public List<Edge> stepForward() {
    if (i == 0) log.append("Начало работы алгоритма;\n");
    if (i < steps.size()) {
        List<Edge> res = steps.get(i++);
        if (res.size() == 1) {
            Edge e = res.get(0);
            log.append(String.format("Выбрано ребро %s-%s с весом %o\n", e.getStart(), e.getEnd(), e.getWeight()));
        } else {
            String cycle = cycleToString(res);
            Edge bridge = res.get(res.size() - 1);
            log.append(String.format("Ребро %s-%s не выбрано, т.к. образует цикл %s\n",
                                     bridge.getStart(), bridge.getEnd(), cycle));
        }
        return res;
    }
    log.append("Конец работы алгоритма;\n");
    return null;
}

public List<Edge> stepBack() {
    if (i > 0) {
        List<Edge> res = steps.get(--i);
        if (res.size() == 1) {
            Edge e = res.get(0);
            log.append(String.format("Ребро %s-%s с весом %o удалено из остова\n", e.getStart(), e.getEnd(), e.getWeight()));
        } else {
            String cycle = cycleToString(res);
            Edge bridge = res.get(res.size() - 1);
            log.append(String.format("Ребро, %s-%s образующее цикл %s удалено из рассмотренных\n",
                                     bridge.getStart(), bridge.getEnd(), cycle));
        }
        return res;
    }
    log.append("Шаг назад невозможен;\n");
    return null;
}

private String cycleToString(List<Edge> res) {
    StringBuilder cycle = new StringBuilder();
    cycle.append(res.get(0).getStart());
    cycle.append("-");
    for (int i = 0; i < res.size() - 1; i++) {
        cycle.append(res.get(i).getEnd());
        cycle.append("-");
    }
    cycle.append(res.get(0).getStart());
    return cycle.toString();
}

public List<List<Edge>> getSteps() {
    return steps;
}

```

```

    public String getLog() {
        return log.toString();
    }

    private List<Edge> DFS(Map<String, Map<String, Integer>> graph,
String start, String end, Set<String> visited, List<Edge> path) {
        if (start.equals(end)) {
            return path;
        }
        visited.add(start);
        for (Map.Entry<String, Integer> next :
graph.get(start).entrySet()) {
            String v = next.getKey();
            if (!visited.contains(v)) {
                List<Edge> newPath = new ArrayList<>(path);
                newPath.add(new Edge(start, v, next.getValue()));
                List<Edge> res = DFS(graph, v, end, visited, newPath);
                if (res != null) return res;
            }
        }
        return null;
    }

    private List<Edge> getCycle(Edge bridge) {
        Map<String, Map<String, Integer>> graph =
fromEdgesToGraph(getCycledTree(bridge));
        List<Edge> res = DFS(graph, bridge.getStart(), bridge.getEnd(),
new HashSet<>(), new ArrayList<>());
        res.add(bridge);
        return res;
    }

    private List<Edge> getCycledTree(Edge bridge) {
        String parent = vertices.find(bridge.getStart());
        return mst.stream().filter(k ->
vertices.find(k.getStart()).equals(parent)).collect(Collectors.toList()
);
    }

    private Map<String, Map<String, Integer>>
fromEdgesToGraph(List<Edge> edges) {
        Map<String, Map<String, Integer>> graph = new HashMap<>();
        for (Edge e : edges) {
            Map<String, Integer> start =
graph.computeIfAbsent(e.getStart(), k -> new HashMap<>());
            Map<String, Integer> end =
graph.computeIfAbsent(e.getEnd(), k -> new HashMap<>());
            start.put(e.getEnd(), e.getWeight());
            end.put(e.getStart(), e.getWeight());
        }
        return graph;
    }
}

```

Название файла: DisjointSets.java

```

package kraskal;

import org.jetbrains.annotations.NotNull;

import java.util.*;
import java.util.stream.Collectors;

public class DisjointSets<T> {
    private final Map<T, T> parents = new HashMap<>();

    public DisjointSets(@NotNull Collection<T> arr) {
        for (T x: arr) parents.put(x, x);
    }

    public T find(T x) {
        T parent = parents.get(x);
        if (parent.equals(x)) return x;
        return find(parent);
    }

    public Map<T, T> getParents() {
        return parents;
    }

    public void unite(T a, T b) {
        parents.put(find(a), find(b));
    }

    public List<T> getSet(T parent) {
        return parents.keySet().stream().filter(k ->
find(k).equals(parent)).collect(Collectors.toList());
    }

    public int size() {
        return parents.size();
    }
}

```