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

ОТЧЕТ

по учебной практике

Тема: Минимальное остовное дерево. Алгоритм Борувки

Студент гр. 0303 _____ Середенков А.А.

Студент гр. 0303 _____ Пичугин М.В.

Студент гр. 0303 _____ Сологуб Н.А.

Руководитель _____ Фирсов М.А.

Санкт-Петербург

2022

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Середенков А.А. группы 0303

Студент Пичугин М.В. группы 0303

Студент Сологуб Н.А. группы 0303

Тема практики: Минимальное остовное дерево. Алгоритм Борувки

Задание на практику:

Командная итеративная разработка визуализатора алгоритма Борувки на Kotlin с графическим интерфейсом.

Алгоритм: построение минимального остовного дерева. Алгоритм Борувки

Сроки прохождения практики: 29.06.2022 – 12.07.2022

Дата сдачи отчета: 12.07.2022

Дата защиты отчета: 12.07.2022

Студент(ка)		Середенков А.А.
Студент(ка)		Пичугин М.В.
Студент(ка)		Сологуб Н.А.
Руководитель		Фирсов М.А.

АННОТАЦИЯ

Задача практики – реализовать графическое приложение, отображающее последовательную работу алгоритма Борувки. Перед выполнением были поставлены точные цели и сроки их реализации. После чего выполнялась работа по установленным срокам.

Summary

The task of practice is to implement a graphical application that displays the sequential operation of Boruvka's algorithm. Before implementation, precise goals and deadlines were set. After that, the work was carried out according to the established deadlines.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данная практическая работа состоит в реализации графического отображения работы алгоритма Борувки. Результат работы – готовое приложение с графическим интерфейсом, позволяющее пользователю удобно настраивать параметры работы алгоритма и наблюдать за каждым этапом его работы.

Алгоритм Борувки – это алгоритм поиска минимального остовного дерева в графе.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. Требования к вводу исходных данных

Исходные данные могут вводится 2 способами:

1) С помощью взаимодействие пользователя с интерфейсом по нажатию мыши на кнопки и графическое полотно. Пользователь нажимает на клавишу для создание вершин графа, чтобы указать программе, что на графическом полотне по нажатию мыши создается вершина графа. Затем пользователь нажимает на клавишу для создания рёбер, и нажимает на две произвольные вершины на полотне для отрисовки и создания ребра между ними.
2) С помощью чтения данных из файла. Приложения сначала проверяет исходный файл на правильность заполнения. Если все верно, то рисуется введенный граф, иначе выведется окно с ошибкой(“WRONG FILE!!!!”).

1.1.2. Требования к визуализации

В левой части программы будет большая область на которой будет отображаться граф, в правом верхнем углу будут находится все необходимые кнопки для работы с приложением, в правом нижнем углу будет находится консоль для вывода логов. В логах будет содержаться следующая информация:
текущая рассматриваемая вершина и список инцидентных ей рёбер, минимальное ребро для рассматриваемой вершины, текущее окрашенное дерево и список инцидентных ему рёбер, минимальное ребро для рассматриваемого дерева, сообщение об объединении двух деревьев в одно в ходе добавления ребра к искомому дереву.

Описание кнопок приложения:

Кнопка ***добавить вершину*** - после нажатия, в левой части приложения с помощью мыши необходимо указать местоположение новой вершины в виде круга с индексом внутри. Индекс задается порядком добавления новой вершины. Потом эти вершины можно будет передвигать с помощью мыши.

Кнопка ***удалить вершину*** - при нажатии на кнопку, мышь переходит в режим удаления, нажав на холсте на выбранную вершину она удалится. Вместе с вершиной удалятся и ребра, связанные с ней.

Кнопка ***добавить ребро*** - при нажатии на кнопку мышь переходит в режим создания ребра. Необходимо нажать на две произвольные вершины и ввести вес ребра в всплывающее окно.

Кнопка ***удалить ребро*** - при нажатии выведется окно, в котором будет 2 поля, в которые нужно будет вписать индексы вершин, чье ребро хотим удалить.

Кнопка ***открыть файл*** - при нажатии откроется окно, в котором можно будет выбрать файл формата *.txt из которого хотим получить исходные данные.

Кнопка ***сохранить граф*** - при нажатии откроется окно, в котором можно будет выбрать или создать файл формата *.txt в котором будут записаны параметры графа представленного в левой части приложения.

Кнопка ***очистить холст*** - при нажатии очистит левую часть приложения от нарисованного на нем графа.

Кнопка ***результат*** - при нажатии выведет конечный результат работы алгоритма в левую часть приложения, а также в правый нижний угол программы, где будет расположен терминал, выведет процесс работы алгоритма в виде логов.

Кнопка ***шаг вперед*** - при нажатии запустит алгоритм, который будет выполняться пошагово, то есть, чтобы завершить работу алгоритма нужно будет продолжать нажимать на эту кнопку, до тех пор пока полностью не

закончится работа алгоритма и не выведется конечный результат. Шагом в данной реализации алгоритма будет считаться добавление нового ребра к искомому дереву. Также для работы данной функции приложения вместо нажатия на эту кнопку, можно использовать клавишу “Z”.

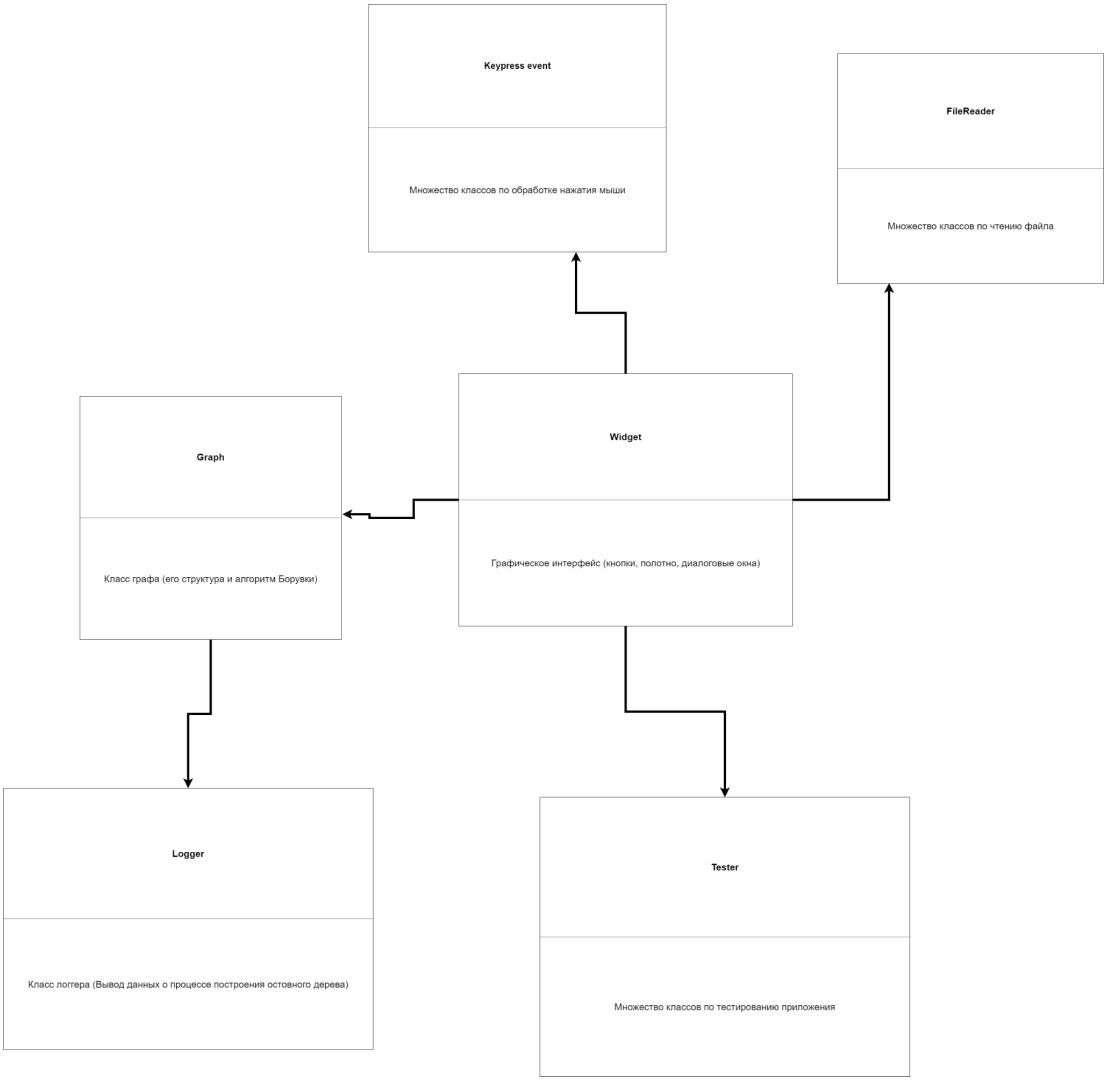
В исходном окне приложения пользователю будет предоставляться выбор для ввода данных: через нажатие кнопки мыши или через файл. Если пользователь выберет ввод данных через файл, то ему необходимо будет нажать соответствующую кнопку после чего откроется окно, в котором можно будет выбрать файл формата *.txt из которого хотим получить исходные данные, на основе которых вершины графа будут отрисовываться на равном расстоянии друг от друга, образуя матричный вид, после чего будет возможность с помощью мыши передвигать эти вершины в удобное для пользователя расположение. В самом файле информация о графе будет написана следующим образом: на первой строчке будет написано количество (*n*) ребер в графе, после чего последует *n* строк содержащих информацию о каждом ребре графа(первые две цифры будут указывать на индексы соединенных вершин, а последняя цифра указывает на вес самого ребра). Также после этого у пользователя будет возможность добавить новые вершины и ребра, с помощью других кнопок. Если же пользователь выберет ввод данных через кнопку мыши, то с помощью других кнопок пользователю будет необходимо добавить все необходимые вершины и ребра с весами.

Элементы графа такие как вершины и ребра будут выглядеть следующим образом:

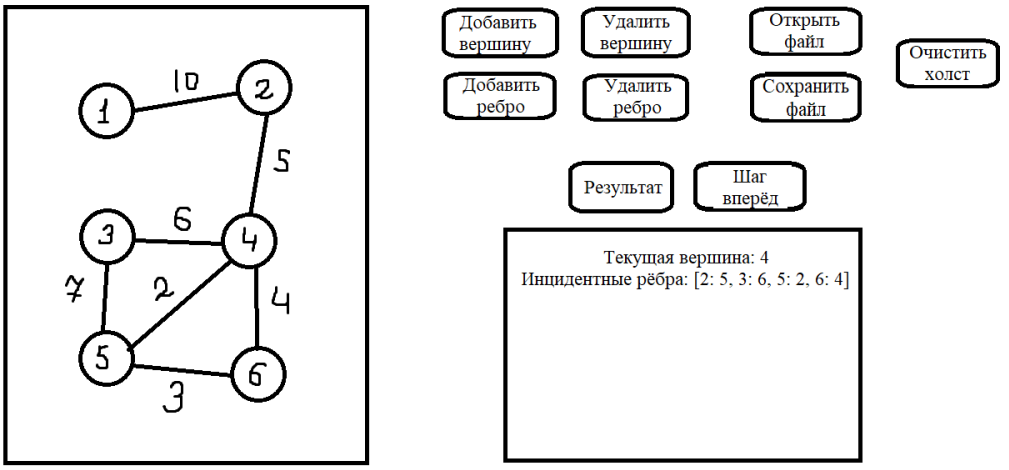
- Вершины выглядят как окружности, внутри которых будут написаны их номера.
- Рёбра представляют собой линии соединяющие вершины, над которыми будет написан их вес.

Процесс алгоритма Борувки будет отображаться через постепенное перекрашивание графа в разные цвета рёбер и вершин, а затем последующее перекрашивание деревьев, полученных на промежуточных результатах. Сначала для каждой вершины будет находится минимальное по весу инцидентное ребро и перекрашивать ребро и вершину в какой-то цвет. То же самое будет происходить для всех остальных вершин. Затем алгоритм будет приращивать и красить минимальное по весу ребро к дереву, полученному в результате предыдущих шагов. Если приращённое ребро соединяет два множества деревьев, то они перекрашиваются в один цвет и становятся единым деревом. Алгоритм работает до тех пор пока не останется одно единственное дерево с одним цветом.

1.1.3. UML диаграмма



1.1.4. Эскиз приложения



1.1.5. Псевдокод алгоритма

// G — исходный граф

// w — весовая функция

function boruvkaMST():

while T.size<n−1

for k∈ Components // Components — множество компонент связности в T. Для

w(minEdge[k])=∞ //каждой компоненты связности вес минимального ребра = ∞

findComp(T) // Разбиваем граф T на компоненты связности обычным dfs-ом.

for (u,v)∈E

if u.comp≠v.comp

if w(minEdge[u.comp])>w(u,v)

minEdge[u.comp]=(u,v)

if w(minEdge[v.comp])>w(u,v)

minEdge[v.comp]=(u,v)

for k∈ Components

T.addEdge(minEdge[k]) // Добавляем ребро, если его не было в T

return T

1.2. Уточнение требований после сдачи прототипа

1)Добавить возможность копирования данных из консоли в буфер обмена.

2) При изменении размера окна приложения её элементы должны масштабироваться в соответствии с этими изменениями.

1.3. Уточнение требований после сдачи 1-й версии

1)Расширение холста в ширину при расширении окна в ширину.

2)Уменьшить размер вершин.

3)При включении режима проведения рёбер должен автоматически отключаться режим добавления вершин, и наоборот.

4) Исправить недостаток: при проведении рёбер щелчок вершины не всегда срабатывает (если щёлкнуть точно в центр).

5)Автоматическое получение фокуса полем для ввода веса ребра.

6)Улучшить рисование рёбер (они должны соединять центры вершин).

7)Имена вершин выводить другим цветом, чтобы имена были хорошо видны над рёбрами.

Исправления в пояснения (только главные):

1) "The first component of connectivity contains the following edges kotlin.Unit" выводится ПОСЛЕ вывода рёбер. Также "kotlin.Unit" явно лишнее.

2) Исправить неверные списки рёбер при объединении компонент.

3) "Current minimal edge:" и "The current cell to which the edge is directed:" в начале процесса поиска ребра некорректны (не получают значения, хотя первое ребро уже рассмотрено).

4) Добавить пустую строку после "==========".

1.4. Уточнение требований после сдачи 2-й версии

1)Исправить недостаток: при проведении рёбер щелчок вершины не всегда срабатывает (если щёлкнуть точно в центр)

2)Автоматическое получение фокуса полем для ввода веса ребра.

3)Изменить размещение вершин и рёбер при чтении из файла в соответствии со спецификацией.

4)Изменить: при очистке холста должна очищаться и консоль.

6)Исправить сохранение файла

7)Рёбра, содержащиеся в минимальном остовном дереве выделить жирной линией, остальные наоборот сделать тоньше.

8)Поменять название окна добавления ребра

9)Добавить (запрет редактирования графа/сброс алгоритма) при изменении графа

10)Изменить вывод при задании несвязного графа

11)Улучшить рисование рёбер (они должны соединять центры вершин)

12)Изменить: как только алгоритм доходит до конца кнопка “результат” должна смениться на “заново” и вернуть граф к изначальному виду, чтобы иметь возможность вновь запустить алгоритм.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

1) Реализация графического интерфейса без функционала
2) Реализация графа
3) Реализация алгоритма Борувки
4) Добавление графического отображение графа и его удаление (функционал кнопок добавления и удаления)
5) Добавления графического отображения алгоритма Борувки (функционал кнопок результата и шагов)
6) Добавления чтения из файла и возможности сохранить граф (функционал кнопок открытие файла и сохранения)

2.2. Распределение ролей в бригаде

Сологуб Н.А.:

- Проектирование и реализация графического интерфейса пользователя (за исключением вывода графа на экран).
- Реализация вывода пояснений хода алгоритма в отдельную консоль.
- Ручное тестирование реализованных частей интерфейса.

Середенков А.А.

- Разбиение алгоритма Борувки
- Разбиение реализованного алгоритма Борувки на логические части для возможности его последующего пошагового отображения в графическом интерфейсе.
- Реализация возможности сохранения и загрузки созданного графа.

· Тестирование работы алгоритма

Пичугин М.В.:

· Создание базовых классов графа.

· Создание классов для графического отображения графа

· Реализация графической части отображения алгоритма Борувки.

· Тестирование графического отображения алгоритма.

2.3. Определения этапов разработки приложения

На этапе прототипа должен быть реализован графический интерфейс, без реализованного алгоритма.

В первой версии должна быть реализована работа алгоритма с некоторыми ограничениями в виде: не будет возможности загружать исходные данные через файл, не будет возможности пошагового выполнения алгоритма.

Во второй версии приложение должно иметь полную работоспособность, без ограничений как в предыдущей версии.

В третьей версии должны быть исправлены все недочеты предыдущих версий.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

class Component - класс компонент-связности, имеет следующие поля:

Cells: MutableList<Cell> - список всех вершин составляющих данную компоненту

VXCells: MutableList<VXCell> - список всех вершин, для отрисовки, составляющих данную компоненту

allEdges: MutableList<Edge> - список всех ребер составляющих данную компоненту

allVXEdges: MutableList<VXEdge> - список всех ребер, для отрисовки, составляющих данную компоненту

edges: MutableMap<Cell, Int> - словарь ребер соединяющих компоненту с другими

class Edge - класс ребра, имеет следующие поля:

source: Cell - начальная вершина ребра

target: Cell - конечная вершина ребра

weight: Int - вес ребра

class Cell - класс вершины, имеет следующие поля:

cellId: String - имя вершины

children: MutableMap<Cell, Int> - словарь смежных вершин

class Model - класс для работы с элементами графа, имеет следующие поля:

end_flag: Boolean - переменная, сообщающая о завершении алгоритма

index: Int - переменная, для получения визуальной составляющей ребра

dif_colors: MutableList<Color> - первоначальные цвета компоненты, для выполнения различности цвета для каждой компоненты

rand: Random - объект для генерации случайного цвета

graphParent: Cell - абстрактный корневой узел, с которого начинает строится граф

allCells: MutableList<Cell> - список всех вершин графа

allVXCells: MutableList<VXCell> - список всех графических вершин графа

addedCells: MutableList<Cell> - список всех вершин графасписок всех добавленных вершин графа

addedVXCells: MutableList<VXCell> **-** список всех добавленных графических вершин графа

removedCells: MutableList<Cell> - список всех удаленных вершин

removedVXCells: MutableList<VXCell> **-** список всех удаленных графических вершин

allEdges: MutableList<Edge> - список всех ребер

allVXEdges: MutableList<VXEdge> - список всех графических рёбер графа

addedEdges: MutableList<Edge> - список всех добавленных рёбер графа

addedVXEdges: MutableList<VXEdge> - список всех добавленных графических рёбер графа

removedEdges: MutableList<Edge> - список удаленных ребер

removedVXEdges: MutableList<VXEdge> - список всех удалённых графических рёбер графа

cellMap: HashMap<String,Cell> **-** словарь вершин, для доступа к объекту вершины по её имени.

allComponents: MutableList<Component> - список компонент связности

n: Int - переменная, отвечающая за индекс рассматриваемой компоненты, в основном алгоритме

log: Logger - поле, фиксирующее изменения в графе

class RandomLayout - класс, задающий случайное размещение вершин при чтении файла графа

rnd: Random - объект класса Random, задающий случайные координаты для вершин

class SquareLayout - класс, задающий матричное расположение вершин при чтении файла графа

step_width: Int - шаг для смещение координат вершины по оси x

step_height: Int - шаг для смещение координат вершины по оси y

separ: Int - количество вершин + 1, находящихся в одной строке

width_val: Int - ширина холста

height_val: Int - высота холста

x: Int - координата x

y: Int - координата y

class Logger - класс для вывода промежуточных данных в консоль, расположенной в приложении, имеет следующие поля:

console: TextArea - консоль расположенная в окне приложения

class VXCell - класс для графического отображения вершины графа, имеет следующие поля:

cellId: String - имя вершины

x: Double - координаты, на которых расположена вершина

y: Double - координаты, на которых расположена вершина

name: Text - имя вершины, для отображения на холсте

edgesmap: MutableMap<VXCell, VXEdge> - словарь, хранящий ребра ключ которого смежная вершина

weightmap: MutableMap<VXCell, Int> - словарь, хранящий смежную вершину и вес ребра, через которую они соединены

neighbors: MutableList<VXCell> - список смежных графических вершин

cell: Cell - вершина, которую нужно отобразить

isDragging: Boolean - флаг, отвечающий за начало и конец передвижения вершины

class VXEdge - класс для отображения введенного ребра на холст, имеет следующие поля:

weigh: Int - поле, хранящее вес ребра.

name: Text - имя ребра, равное его весу, необходимое для корректного отображения ребра

source: VXCell - поле, хранящее графическую вершину из которого направлено ребро

target: VXCell - поле, хранящее графическую вершину в которую направлено ребро

class NewFolderController - класс вызывающий окно для установки или изменения веса ребра, для , имеет следующие поля:

dialog_edge: AnchorPane - окно для записи веса ребра

weight_value: TextField - текстовое поле, куда записывается вес ребра

main: HelloController - ссылка на основной контроллер

class HelloController - класс отвечающий за работу кнопок в окне приложения, имеет следующие поля:

cell_number: Int - имя следующей вершины

weight: Int - вес полученный при добавлении или изменении веса ребра

create_components: Boolean - флаг, отвечающий за создания компонент-связности

buff_vxcells: MutableList<VXCell> - список всех графических вершин для открытия файла

buff_vxedges: MutableList<VXEdge> - список всех графических ребер для открытия файла

cellmap: MutableMap<Int,VXCell> - словарь вершин, где ключ это индекс вершины, а значение это сама вершина

layout: SquareLayout - объект для размещения вершин графа при открытии файла

edge: MutableList<VXCell> - список для создание и удаления ребра

model: Model - хранит все данные и методы для работы с графом

vxcells: MutableList<VXCell> - список всех графических вершин

vxedges: MutableList<VXEdge> - список всех графических ребер

node: AnchorPane - объект на котором размещаются все элементы приложения

result_button: Button - кнопка результата

step_forward_button: Button - кнопка шага вперед

del_edge_button: Button - кнопка удаления ребра

add_vert_button: Button - кнопка добавления вершины

add_edge_button: Button - кнопка добавления ребра

del_vert_button: Button - кнопка удаления вершины

scroll_pane: ScrollPane - область с прокруткой

holst: Pane - холст

my_console: TextArea - поле-консоль, куда выводятся промежуточные данные

3.2. Основные методы

Методы класса ***Component****:*

fun getCells() - метод, возвращающий список *Cells*

fun getAllEdges() - метод, возвращающий список *allEdges*

fun getEdges() - метод, возвращающий словарь *edges*

fun checkComponentCell(name: String): Int - метод, возвращает индекс по которой находится вершина с именем *name* в списке *Cells,* если вершины нет в списке возвращает размер списка

fun printneigbors() - метод, возвращающий строку, содержащую ребра, через которые данная компонента соединена с другими

fun removeEdge(temp: Component) - метод, удаляет из словаря ребра соединяющую текущую компоненту и компоненту *temp*

fun addCells(temp: Cell) - метод, добавляет в список *Cells* вершину *temp*

fun addVXCells(temp: VXCell) - метод, добавляет в список *VXCells* вершину *temp*

fun addAllEdges(temp: Edge, cur: VXEdge) - метод, добавляет в список *allEdges* ребро *temp,* а в список *allVXEdges* ребро *cur*

fun addEdges(temp: Cell, num: Int) - метод, добавляет в словарь *edges* ребро содержащее вершину *temp* с весом *num*

fun setCells(temp: MutableList<Cell>) - метод, заменяет список *Cells* списком *temp*

fun setEdges(temp: MutableList<Edge>) - метод, заменяет список *allEdges* списком *temp*

fun setAllEdges(temp: MutableMap<Cell, Int>) - метод, заменяет словарь *edges* словарем *temp*

fun printallCells() - метод, возвращающий строку, содержащую все вершины данной компоненты

fun mergeComp(temp: Component) - метод, объединяющий текущую компоненту с компонентой *temp*

Методы класса ***Edge****:*

fun getsource() - метод, возвращающий вершину *source*

fun gettarget() - метод, возвращающий вершину *target*

fun getweight() - метод, возвращающий вес ребра *weight*

Методы класса ***Cell****:*

fun addCellChild(cell: Cell, weight : Int) - метод, добавляющий новую смежную вершину *cell* в словарь *children*

fun getCellChildren() - метод, возвращающий словарь *children*

fun removeCellChild(cell: Cell) - метод, удаляющий из словаря *children* смежную вершину *cell*

fun getcellId() - метод, возвращающий имя вершины

Методы класса ***Model****:*

fun setAllComponents(temp:MutableList<Component>) - метод, заменяющий список *allComponents* списком *temp*

fun getAllComponents() - метод, возвращающий список всех компонент связности

fun clearAddedLists() - метод, очищающий списки *addedCells* и *addedEdges*

fun getaddedCells() - метод, возвращающий список *addedCells*

fun get_color() - метод получения цвета для компоненты

fun getremovedCells() - метод, возвращающий список *removedCells*

fun getallCells() - метод, возвращающий список *allCells*

fun getAddedEdges() - метод, возвращающий список *addedEdges*

fun getRemovedEdges() - метод, возвращающий список *removedEdges*

fun getAllEdges() - метод, возвращающий список *allEdges*

fun addCell(cell: VXCell) - метод, добавляющий графическую вершину графа

fun del_edge(edge: Edge,cur: VXEdge) - метод, удаляющий графическую вершину

fun addEdge(sourceId: String, targetId: String, weight : Int) - метод, создающий ребро и добавляющий его в список *addedEdges*

fun findEdgeinList(temp1: Cell, temp2: Cell, weight: Int) - метод, находящий в списке *allEdges* ребро по двум вершинам между которыми оно лежит и его веса

fun checkEdge(temp1: Cell, temp2: Cell, weight: Int) - метод, проверяющий находится ли ребро в списке *allEdges* по двум вершинам и весу

fun findEdge(comp: Component, temp: Cell, weight: Int) - метод, возвращающий ребро, соединяющее две входные вершины

fun findComp() - метод возвращающий компоненту, если она одна в списке *allComponents*, или если у неё нет рёбер соединяющих её с другими компонентами связности

fun merge() - произвести слияние добавленных вершин (рёбер) со всеми вершинами (рёбрами) и удалить из всех вершин удалённые вершины (рёбра)

fun remove_cell(cell:VXCell) - метод, удаляющий графическую вершину графа

fun clear_all() - метод, очищающий все списки объекта класса

fun getallVXCells() - метод, возвращающий список *allVXCells*

fun createAllComponents() - метод, заполняющий список *allComponents* и выводящий *true* или *false*(смогли ли создаться компоненты)

fun removecomponents() - метод, очищающий список компонент связности и обнуляющий индекс *n*

fun stepAlgoritm() - метод, прогоняющий один шаг алгоритма

fun printresult() - метод, выводящий получившееся минимальное остовное дерево

fun BoruvkaMST(index:Int) - метод, запускающий основной алгоритм(алгоритм Борувки)

fun checkEdgesOut() - метод, проверяющий содержит ли компонента выходящие из неё рёбра

Методы класса ***RandomLayout***:

fun execute() - метод, генерирующий случайные координаты для размещения объектов на холсте

Методы класса ***SquareLayout****:*

fun set_param(width: Double,height: Double) - метод, задающий координаты для отображения вершины

fun reset() - метод, обнуляющий координаты

fun execute() - метод, генерирующий новые координаты для размещения вершины на холсте в матричном виде

Методы класса ***Logger****:*

fun log(temp:String) - метод, выводящий работу алгоритма в консоль приложения

Методы класса ***VXCell***:

fun getcellId() - метод, возвращающий имя вершины

fun get_lable() - метод, возвращающий имя вершины, находящееся в её графическом представлении

fun getPosition() - метод, возвращающий координаты центра вершины

fun remove_connection() - метод, удаляющий упоминание текущей вершиной у других

fun setPosition(x: Double, y: Double) - метод, устанавливающий координаты центра вершины

fun edge_add_config() - метод, отключающий перемещение вершины по холсту и выделяющий её красной контуром

fun enableDrag() - метод, позволяющий перемещать вершину по холсту

fun disableDrag() - метод, запрещающий перемещать вершину по холсту

fun setLabel(label: Text) - метод, устанавливающий имя вершины

fun boundCenterCoordinate(value: Double, min: Double, max: Double) - вспомогательный метод, используемый для перемещения вершины

Методы класса ***VXEdge***:

fun getsource() - метод, возвращающий графическую вершину, являющуюся началом ребра

fun set_label(label: Text) - метод, задающий и форматирующий имя ребра

fun get_label() - метод, возвращающий имя ребра

fun gettarget() - метод, возвращающий графическую вершину, являющуюся концом ребра

fun getweight() - метод, возвращающий вес ребра

Методы класса ***HelloController***:

fun save_file(event: ActionEvent) - метод, открывающий окно для выбора файла, для сохранения графа

fun SaveGraph(path: String) - метод, сохраняющий граф расположенный на холсте в файл

fun getSingleCell() - метод, возвращающий список вершин, которые не связаны с другими вершинами рёбрами

fun initialize() - метод, логирующий промежуточные результаты в консоль приложения

fun open_file(event: ActionEvent) - метод, открывающий окно для выбора файла, для отрисовки графа

fun take_graph(path: String) - метод, строящий граф на холсте по данным из файла

fun clear_holst() - метод, очищающий холст от нарисованного на нем графа

fun off_parametrs() - метод, запрещающий взаимодействовать с графом на холсте

fun add_vert(event: ActionEvent) - метод, добавляющий вершину

fun del_vert(event: ActionEvent) - метод, удаляющий вершину

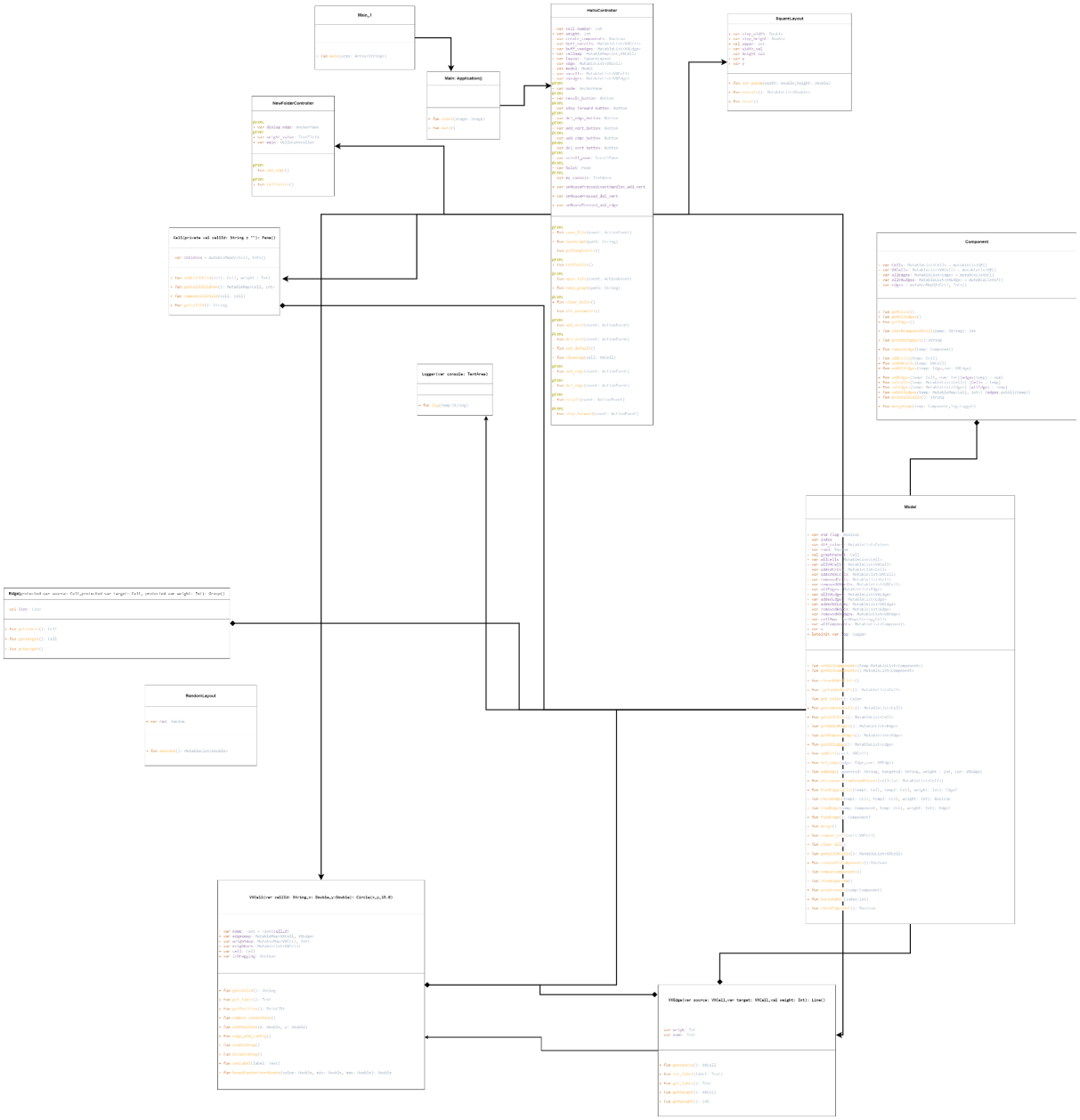
fun add_edge(event: ActionEvent) - метод, добавляющий ребро

fun del_edge(event: ActionEvent) - метод, удаляющий ребро

fun result(event: ActionEvent) - метод, вызывающий начало алгоритма

fun step_forward(event: ActionEvent) - метод, вызывающий один шаг алгоритма

3.3. UML - диаграмма



3.4. Описание алгоритма

Запускается метод *BoruvkaMST()*, который должен вернуть список всех ребер в получившемся минимальном остовном дереве. Если ребер в графе нет то метод вернет *null*.

Сначала для каждой вершины исходного графа создаются объекты класса *Component,* в котором для рассматриваемой вершины в поле словарь *edges* сохраняются все смежные с ней. **Дальше запускается цикл, в теле которого поочередно рассматривается каждая компонент связности, в которой ищется минимальное ребро соединяющее ее с другой компонентой. При нахождении такого ребра, рассматриваемая компонента и та с которой она связана этим ребром объединяются в одну, при этом поля рассматриваемого компонента - словарь *edges* и список *allEdges* изменяются: из словаря удаляются все ребра который связывали две предыдущие компоненты и добавляются новые ребра которые связывали вторую с другими; в список сохраняются все ребра из которых состоят эти компоненты и ребро связывающее их.** Эти действия повторяются до тех пор, пока не останется только одна компонент связанность содержащая все вершины.

4. ТЕСТИРОВАНИЕ

4.1. План тестирования

Для проверки корректной работы приложения необходимо проверить работоспособность всех кнопок графического интерфейса.

Общая проверка кнопок:

1)При первичном нажатии кнопки включается определенный режим взаимодействия с холстом для соответствующих кнопок (***добавить вершину***, ***удалить вершину***, ***добавить ребро***, ***удалить ребро***). При повторном нажатии на кнопку определённый режим взаимодействия с холстом выключается.

2)Нажать на кнопку и нажать на пространство вне холста. В этом случае ничего не произойдёт и у пользователя остается возможность использовать функцию этой кнопки, либо задействовать другие кнопки.

Проверка кнопки ***добавить вершину:***

1)Проверить, что кнопка выполняет необходимый функционал (добавляет вершину на холст при нажатии кнопкой мыши по холсту)

2) Проверить, что есть возможность добавить вершину на место уже расположенной вершины(наложение объектов возможно и их отделение тоже возможно)

Проверка кнопки ***удалить вершину***:

1)Проверить, что кнопка выполняет необходимый функционал (удаляет вершину и инцидентные ей ребра при нажатии кнопкой мыши по вершине на холсте)

Проверка кнопки ***добавить ребро***:

1)Проверка основного функционала кнопки (при нажатии двух вершин должно создаваться ребро и выводится окно для задания веса вершины)

2)Нажать на кнопку и указать на две вершины уже имеющие ребро. В таком случае должна быть возможность замены прошлого веса ребра на новый, заданный пользователем

3)Нажать на кнопку и указать только на одну вершину дважды, тогда ничего не произойдет.

4)Нажать на кнопку и промахнуться по второй или первой вершине, все равно останется возможность указать на нужные пользователю вершины.

Проверка кнопки ***удалить ребро***:

1)Проверка основного функционала кнопки (при нажатии двух смежных вершин должно удаляться ребро)

2)Нажать на кнопку и указать на две несмежные вершины, тогда ничего не произойдет.

3)Нажать на кнопку и промахнуться по второй или первой вершине, все равно останется возможность указать на нужные пользователю вершины.

Проверка кнопки ***открыть файл***:

1) Проверка основного функционала (возможность открыть txt файл)

2) В случае когда на холсте уже расположен граф, то при открытии файла граф, расположенный на холсте должен удалиться (в случае корректности данных файла) и за место него отрисоваться граф, полученный из файла

3) Если файл не был выбран или данные файла некорректны (в этом случае выводится сообщение о некорректности файла), то холст в приложении не должен меняться, т.к. новый граф не был считан.

Проверка кнопки ***сохранить файл***:

1)В случае когда на холсте не будет расположен граф, тогда ничего не произойдет

2) Если файл не был выбран, тогда граф не будет сохранен и ничего не произойдет.

3)Если выбран существующий файл то должно быть выведено окно, уточняющее намерения пользователя. В случае согласия - файл будет переписан, в противном случае пользователю предложат выбрать файл вновь.

Проверка кнопки ***очистить холст***:

1)При пустом холсте не будет ничего происходить.

Проверка кнопки ***результат***:

1) После нажатия кнопки выводится минимальное остовное дерево для заданного графа. При повторном нажатии ничего не произойдёт.

2) В случае пошагового выполнения алгоритма при помощи кнопки ***шаг вперёд***, кнопка ***результат*** должна вывести итог независимо от текущего шага.

Проверка кнопки ***шаг вперёд***:

1)После нажатия кнопки выводятся промежуточные данные. Как только алгоритм закончится, последующие нажатия на кнопку не производят никаких действий

Для проверки корректной работы приложения также необходимо проверить работоспособность слияния компонент связности.

1)При слиянии двух простых компонент связности (состоящих из одной вершины) к первой компоненте добавится вершина из второй, все ребра, ведущие из второй компоненты, а также к первой компоненте

добавится ребро, соединяющее две объединяемые компоненты. Последняя компонента удаляется из списка всех компонент связности.

2)При слиянии двух сложных компонент связности (состоящих из нескольких вершин) к первой компоненте добавятся все вершины ведущие из второй компоненты, все ребра из второй компоненты, а также к первой компоненте добавится ребро, соединяющее две объединяемые компоненты. Последняя компонента удаляется из списка всех компонент связности.

3)При слиянии одной простой и сложной компонент связности к первой компоненте добавятся все вершины из второй, все ребра ведущие из второй компоненты, а также к первой компоненте добавится ребро, соединяющее две объединяемые компоненты. Последняя компонента удаляется из списка всех компонент связности.

4)При слиянии одной сложной и одной простой компонент связности к первой компоненте добавится вершина из второй, все ребра ведущие из второй компоненты, а также к первой компоненте добавится ребро, соединяющее две объединяемые компоненты. Последняя компонента удаляется из списка всех компонент связности.

Для проверки корректной работы приложения также необходимо проверить работоспособность самого алгоритма. На вход подается граф:

1) у которого несколько компонент связности соединены несколькими рёбрами с одинаковыми весами. В этом случае алгоритм сохранит первое рассматриваемое ребро и объединит эти компоненты через него.

2)у которого компоненты связности соединены ребрами разных весов. В этом случае алгоритм сохранит ребро имеющее наименьший вес и объединит эти компоненты через него.

3) у которого веса всех ребер одинаковы. В этом случае алгоритм сначала попарно объединит изначальные компонент связности по первому ребру. После этого алгоритм продолжит соединять компоненты связности через первые рассматриваемые ребра.

4) у которого присутствуют циклы. В этом случае алгоритм включит минимальные рёбра цикла в искомое минимальное остовное дерево.

5) у которого одна вершина. В этом случае алгоритм не запустится.

6) у которого между вершинами нет ребёр. В этом случае алгоритм не запустится.

4.2. Тестирование графического интерфейса

Общая проверка всех кнопок приложения показана на рис. 1-2.

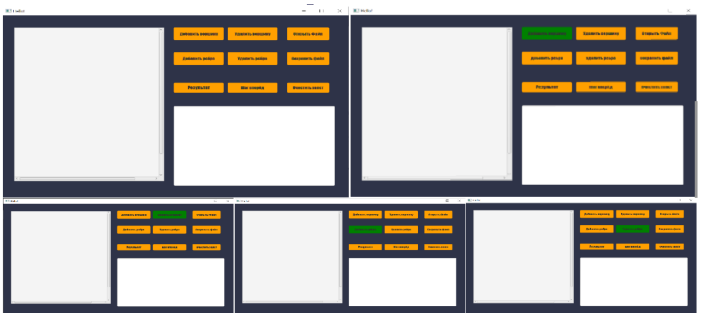


Рисунок 1 - Включение определенного режима после нажатия кнопки

(Графический интерфейс работает корректно)

Рисунок 2 - Нажатие на пространство вне холста после нажатия на кнопку

(Графический интерфейс работает корректно)

Проверка кнопки ***добавить вершину*** показана на рис. 3-4.

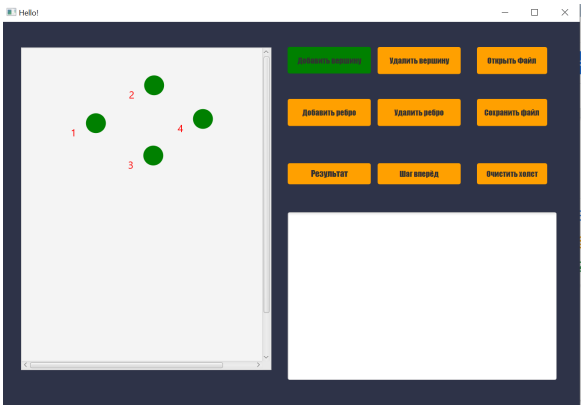


Рисунок 3 - Добавление вершин на холст

(Графический интерфейс работает корректно)

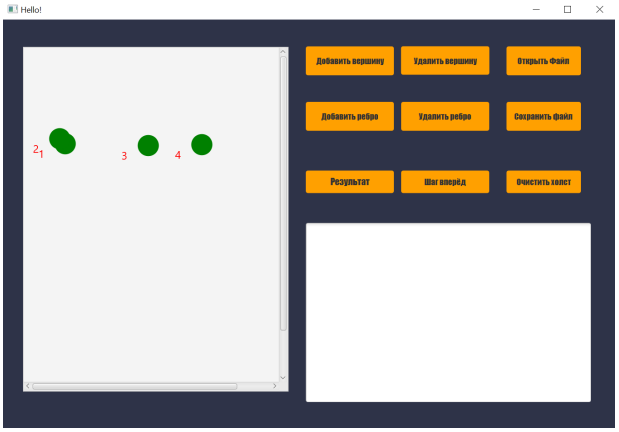


Рисунок 4 - Наложение и разъединение вершин

(Графический интерфейс работает корректно)

Проверка кнопки ***удалить вершину*** показана на рис. 5.

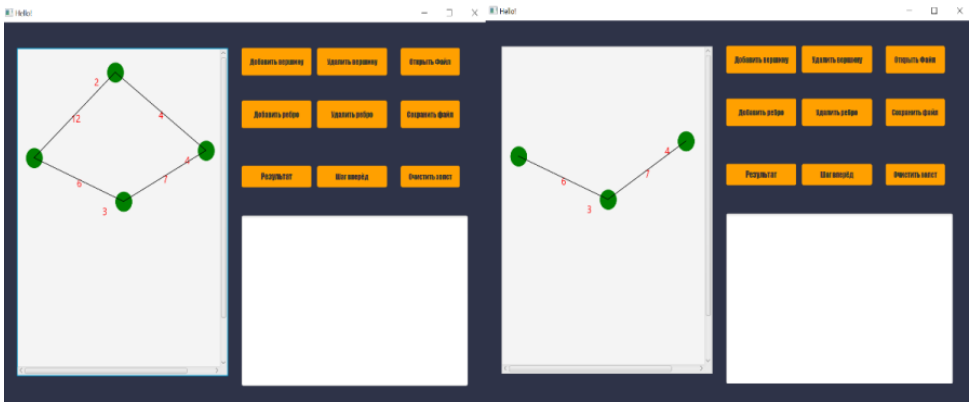


Рисунок 5 - Удаление вершины номер 2

(Графический интерфейс работает корректно)

Проверка кнопки ***добавить ребро*** показана на рис. 6-8.

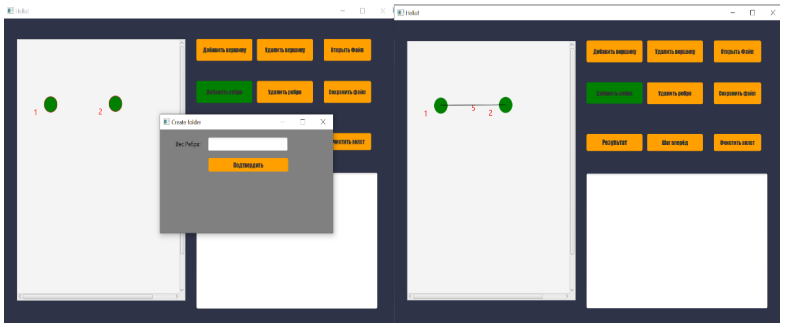


Рисунок 6 - Создание ребра(Графический интерфейс работает корректно)

Рисунок 7 - Нажатие на одну вершину несколько раз (Графический интерфейс работает корректно)

Рисунок 8 - Промах по второй вершине (Графический интерфейс работает корректно)

Проверка кнопки ***удалить ребро*** показана на рис. 9-11.

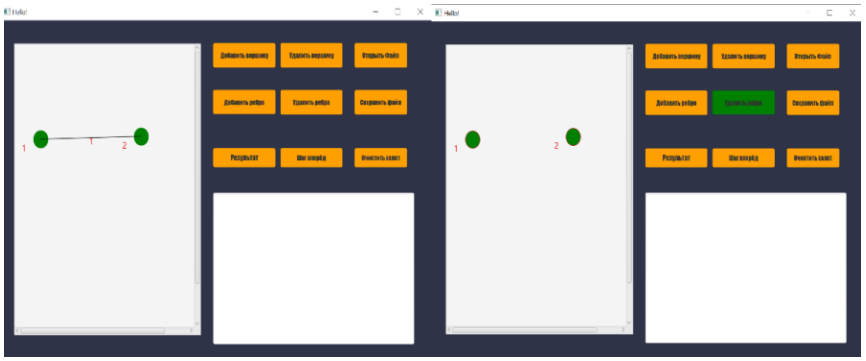


Рисунок 9 - Удаление ребра

(Графический интерфейс работает корректно)

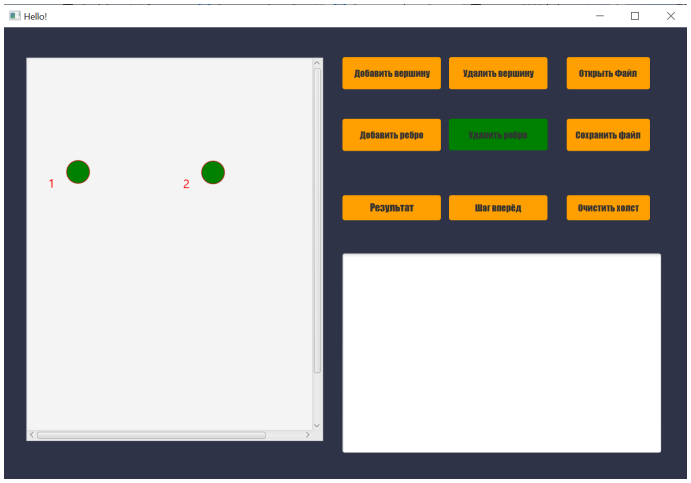


Рисунок 10 - Указание на 2 несмежные вершины

(Графический интерфейс работает корректно)

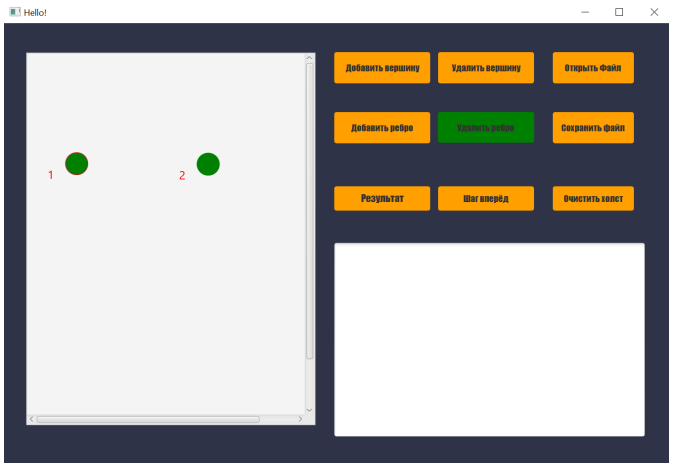


Рисунок 11 - Промах по второй вершине

(Графический интерфейс работает корректно)

Проверка кнопки ***очистить холст*** показана на рис. 12.

Рисунок 12 - Очистили пустой холст

(Графический интерфейс работает корректно)

Проверка кнопки ***результат*** показана на рис. 13-14.

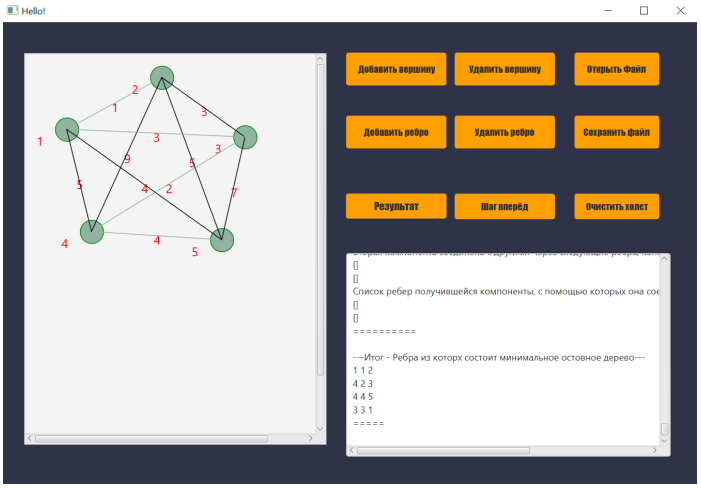


Рисунок 13 - Нашли минимальное остовное дерево

(Графический интерфейс работает корректно)

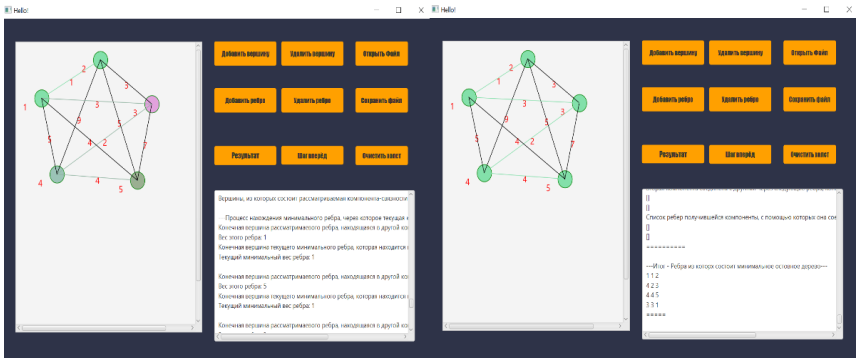


Рисунок 14 - Сразу нашли результат после нескольких шагов

(Графический интерфейс работает корректно)

Проверка кнопки ***шаг вперёд*** показана на рис. 15.

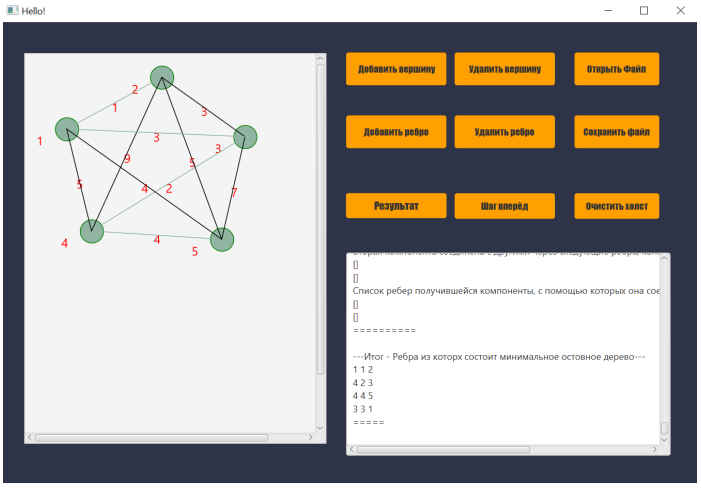


Рисунок 15 - Нашли минимальное остовное дерево

(Графический интерфейс работает корректно)

Проверка кнопки ***сохранить файл*** показана на рис. 16-18.

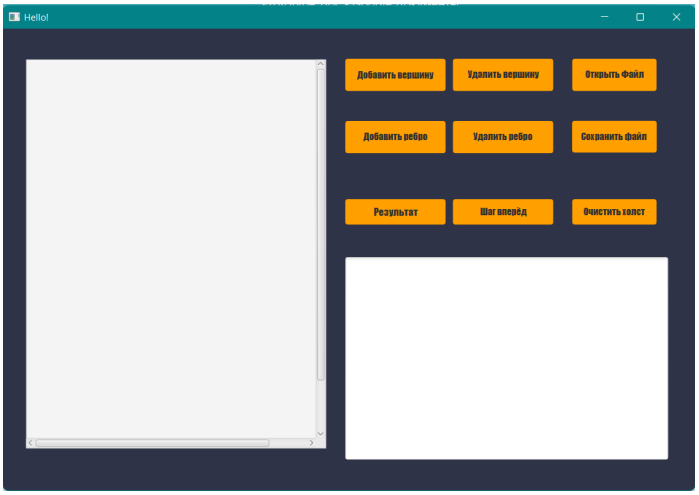


Рисунок 16 - Сохранение пустого графа

(Графический интерфейс работает корректно)

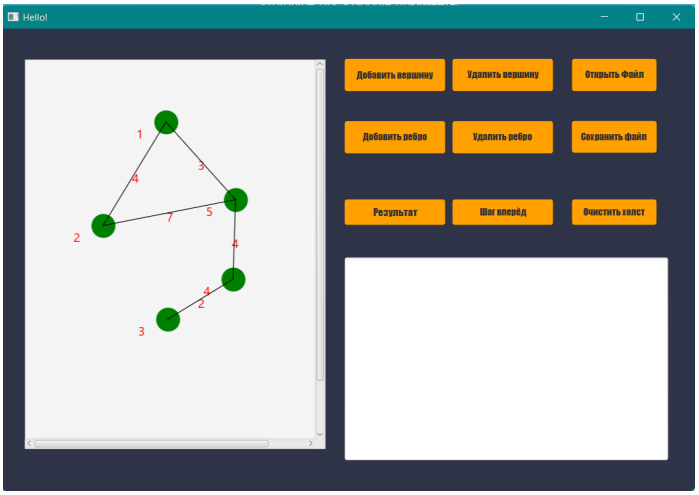


Рисунок 17 - Сохранение графа без выбора файла

(Графический интерфейс работает корректно)

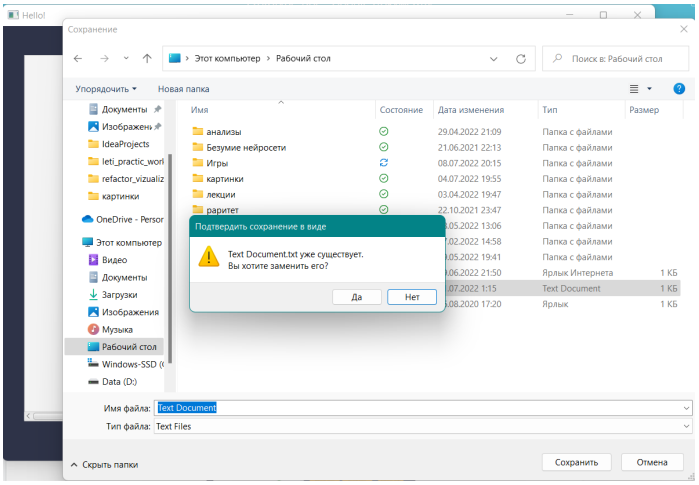


Рисунок 18 - Сохранение графа с выбором файла и предупреждение пользователя о перезаписи файла

(Графический интерфейс работает корректно)

Проверка кнопки ***открыть файл*** показана на рис. 19-20.

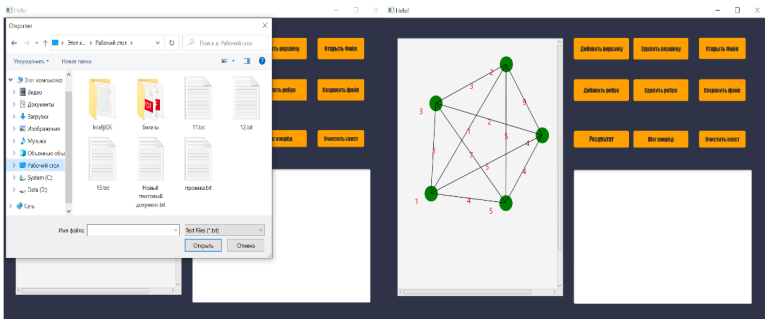


Рисунок 19 - Открытие файла с графом

(Графический интерфейс работает корректно)

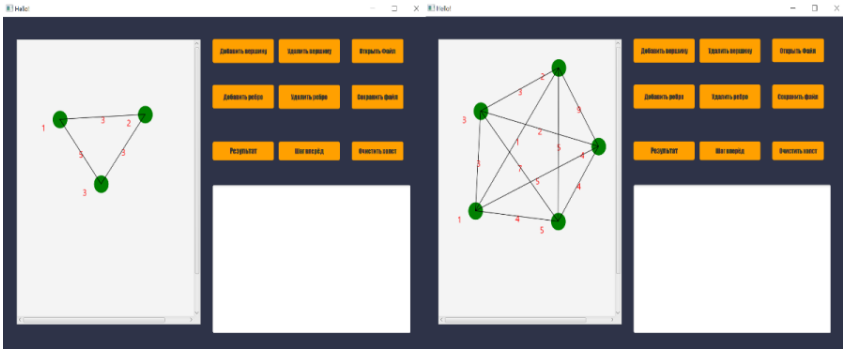


Рисунок 20- Открытие файла с графом, при другом нарисованном на холсте

(Графический интерфейс работает корректно)

4.3 Тестирование слияния компонент связности

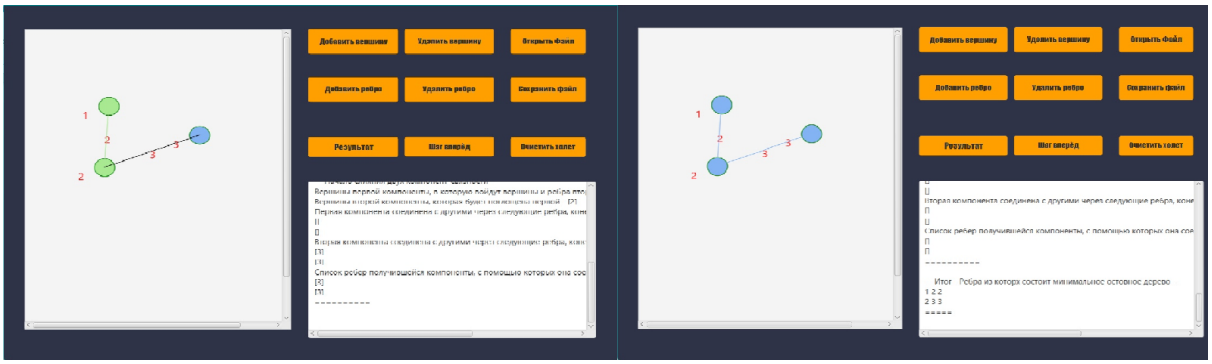


Рисунок 21 - Присоединение к простой компоненте сложной

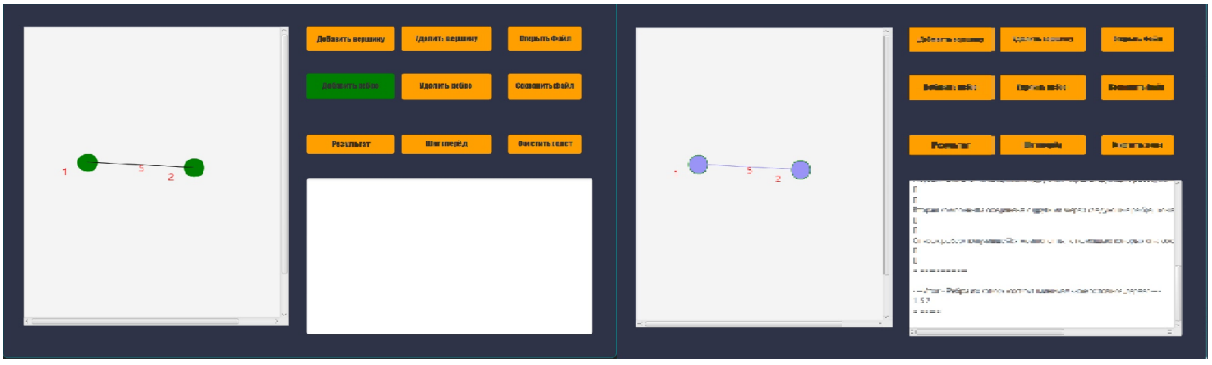


Рисунок 22 - Слияние двух простых компонент связности

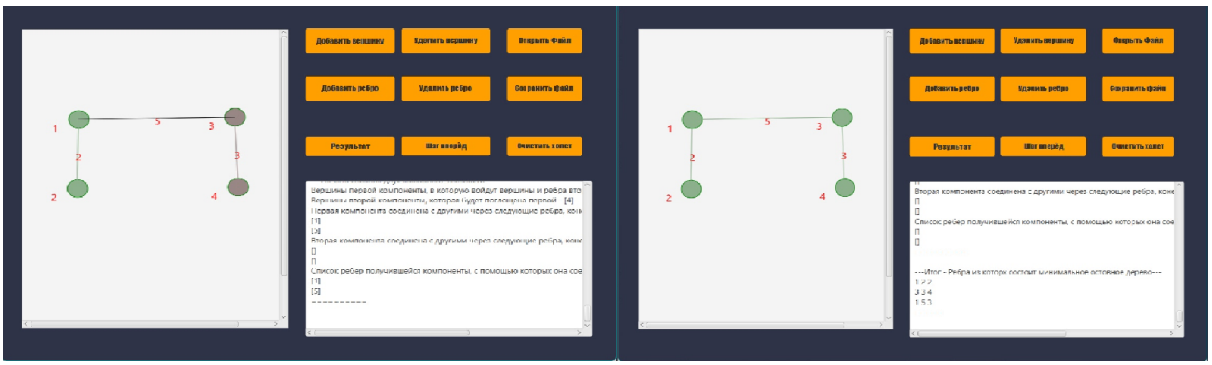


Рисунок 23 - Слияние двух сложных компонент связности

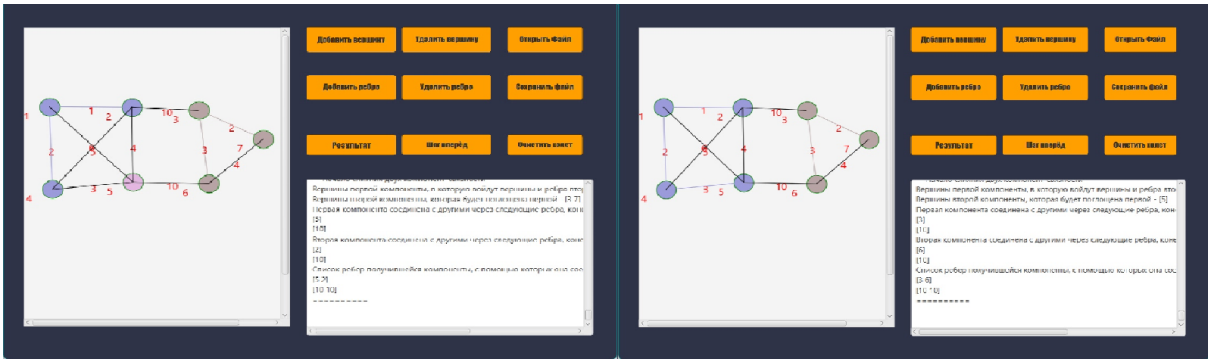


Рисунок 24 - Присоединение к сложной компоненте простой

4.4 Тестирование кода алгоритма

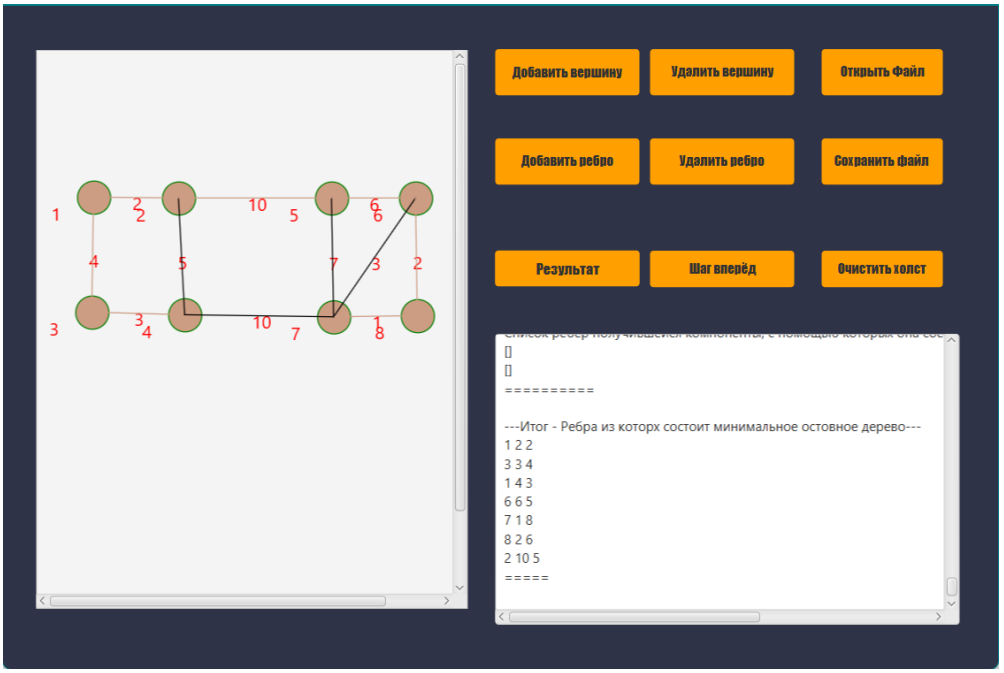


Рисунок 25 - Тестирование случая, когда несколько компонент связности соединены несколькими рёбрами с одинаковыми весами.

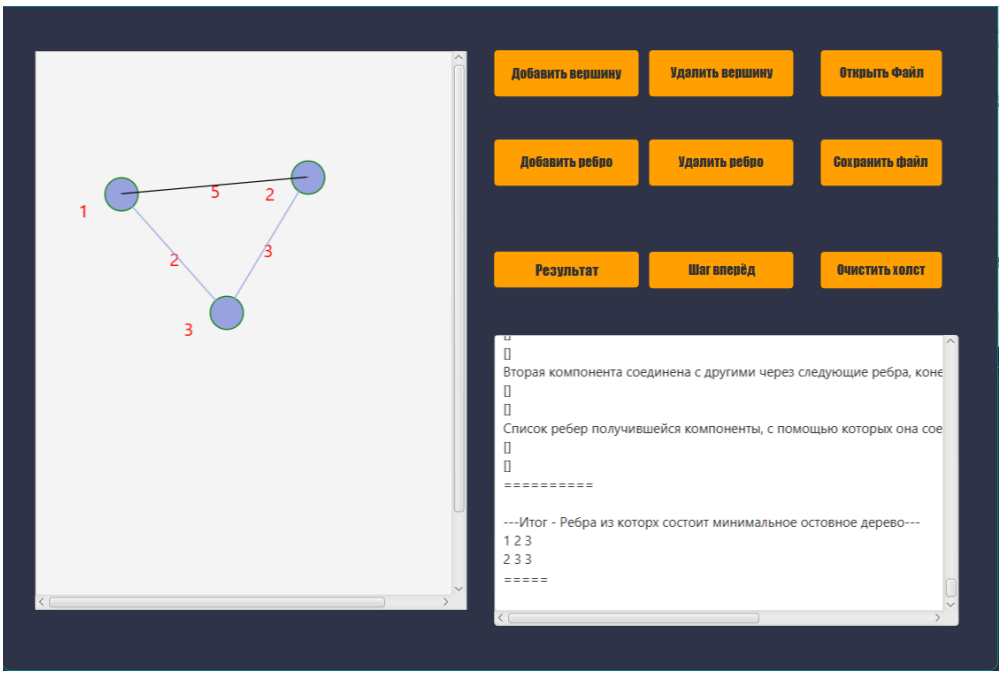


Рисунок 26 - Тестирование случая, когда компоненты связности соединены ребрами разных весов.

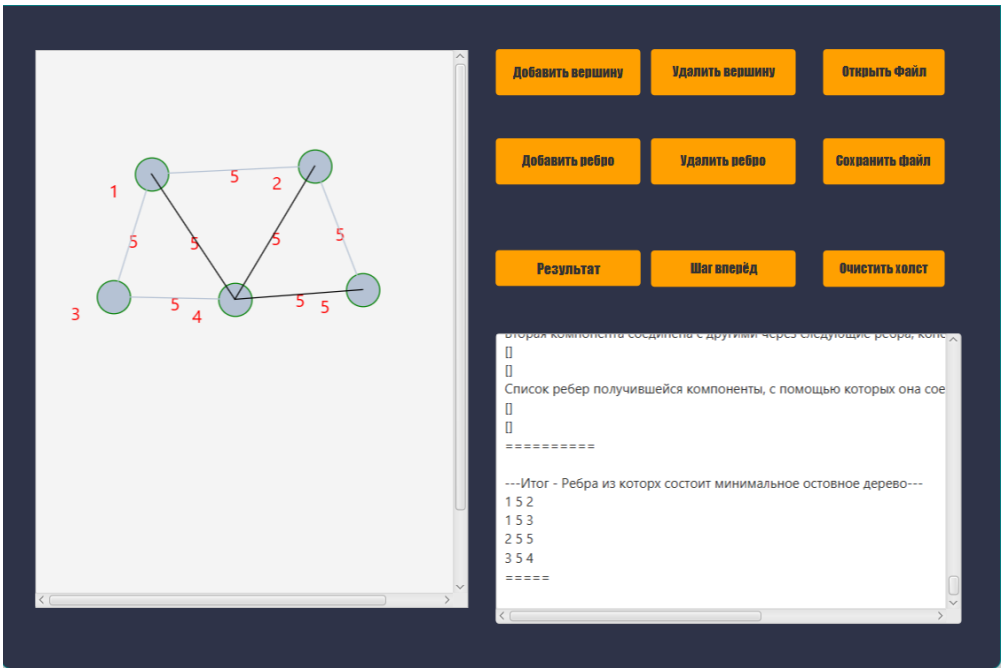


Рисунок 27- Тестирование случая, когда веса всех ребер одинаковы.

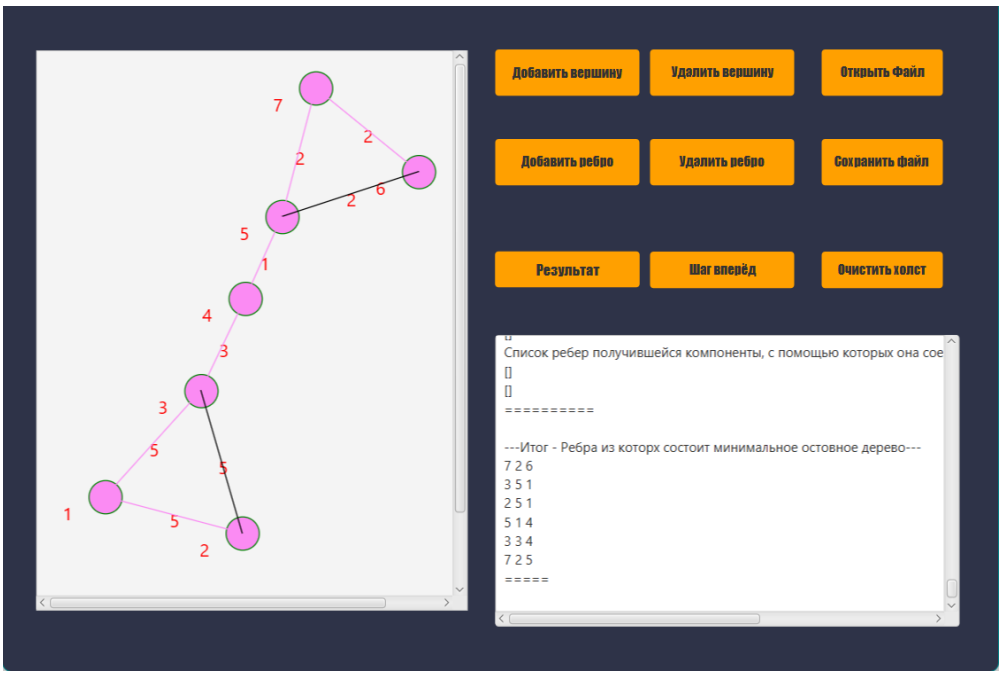


Рисунок 28 - Тестирование случая, когда у графа присутствуют циклы.

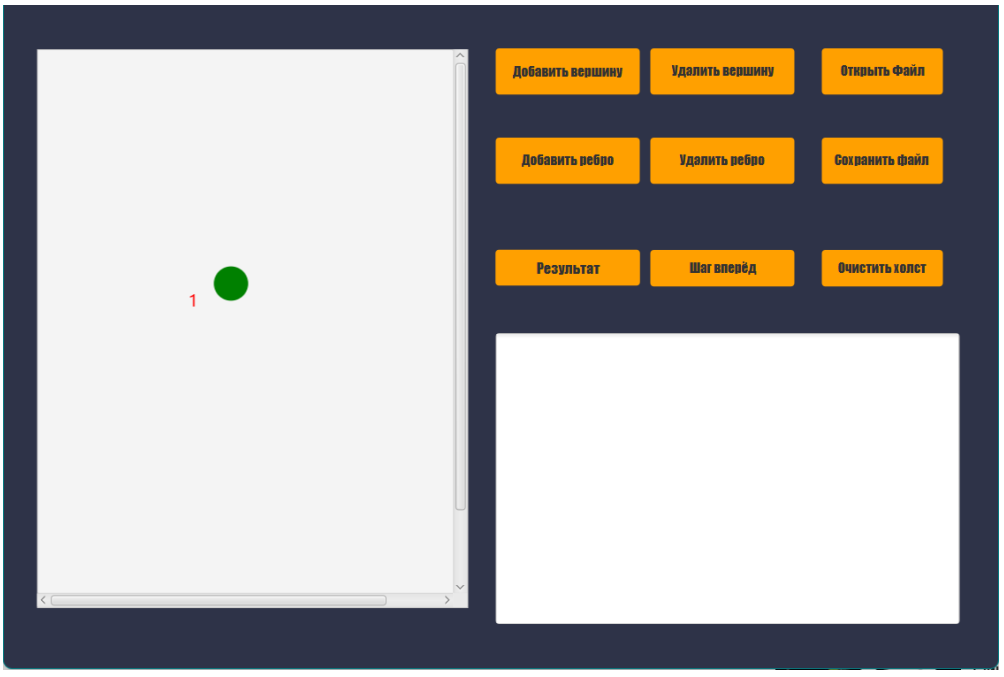


Рисунок 29 - Тестирование случая, когда у графа одна вершина.

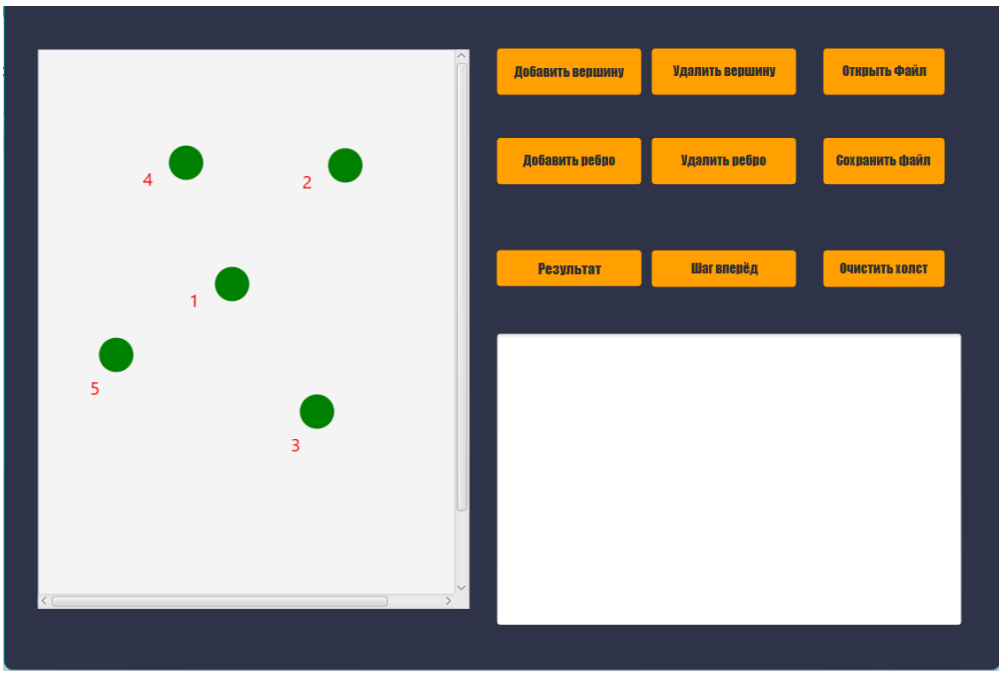


Рисунок 30 - Тестирование случая, когда у графа между вершинами нет ребёр.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы, были выполнены все поставленные задачи в короткие сроки с минимальным количеством проблем. Некоторые идеи и способы реализации были переделаны в процессе разработки, опираясь на реальность их реализации за поставленное время. В конечном итоге получилась программа с графическим интерфейсом выполняющая поставленную задачу, а именно находящая минимальное остовное дерево алгоритмом Борувки. К сожалению, в дальнейшем в программе может обнаружиться некоторое количество багов, которые не были обнаружены при реализации и дальнейшем тестировании.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.Курс обучения Kotlin // Stepik: https://stepik.org/lesson/66286/step/1?unit=43162

2.Документация языка Kotlin: https://kotlinlang.org/

3.Форум программистов, где можно получить помощь по любому интересующему вопросу // Stack overflow. URL: https://stackoverflow.com

4.Описание работы алгоритма Борувки: https://progler.ru/blog/algoritm-boruvki-zhadnyy-algo-9

5.Работа с файлами в Kotlin: https://www.fandroid.info/7-osnovy-kotlin-fajlovye-operatsii/

6.Инструкция для работы с Scene Builder: https://code.makery.ch/ru/library/javafx-tutorial/part1/

7. Документация по JavaFX // Oracle: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm

ПРИЛОЖЕНИЕ А
НАЗВАНИЕ ПРИЛОЖЕНИЯ

Название файла: HelloController.kt

```
package com.example.refactor_vizualize_graph_ktln

import com.fxgraph.graph.Cell
import com.fxgraph.graph.Edge
import com.fxgraph.graph.Logger
import com.fxgraph.graph.Model
import com.fxgraph.layout.SquareLayout
import com.fxgraph.vizualization.VXCell
import com.fxgraph.vizualization.VXEdge
import javafx.event.ActionEvent
import javafx.event.EventHandler
import javafx.fxml.FXML
import javafx.fxml.FXMLLoader
import javafx.scene.Node
import javafx.scene.Parent
import javafx.scene.Scene
import javafx.scene.control.Button
import javafx.scene.control.ScrollPane
import javafx.scene.control.TextArea
import javafx.scene.control.TextField
import javafx.scene.input.MouseEvent
import javafx.scene.layout.AnchorPane
import javafx.scene.layout.Pane
import javafx.scene.paint.Color
import javafx.scene.text.Text
import javafx.stage.FileChooser
import javafx.stage.Modality
import javafx.stage.Stage
import java.io.BufferedWriter
import java.io.File
import java.io.IOException
```

```
import java.io.InputStream
import java.util.Collections.max

class HelloController {
private var cell_number: Int = 1
var weight: Int = 0
private var create_components: Boolean = false
private var buff_vxcells: MutableList<VXCell> =
mutableListOf()
private var buff_vxedges: MutableList<VXEdge> =
mutableListOf()
private var cellmap: MutableMap<Int,VXCell> = hashMapOf()
private var layout: SquareLayout = SquareLayout()
private var edge: MutableList<VXCell> = mutableListOf()
private var model: Model = Model()
private var vxcells: MutableList<VXCell> = mutableListOf()
private var vxedges: MutableList<VXEdge> = mutableListOf()
@FXML
private lateinit var node: AnchorPane
@FXML
private lateinit var result_button: Button
@FXML
private lateinit var step_forward_button: Button
@FXML
private lateinit var del_edge_button: Button
@FXML
private lateinit var add_vert_button: Button
@FXML
private lateinit var add_edge_button: Button
@FXML
private lateinit var del_vert_button: Button
@FXML
private lateinit var scroll_pane: ScrollPane
@FXML
```

```
private lateinit var holst: Pane
@FXML
private lateinit var my_console: TextArea
@FXML
private fun save_file(event: ActionEvent){
 val ex1: FileChooser.ExtensionFilter =
FileChooser.ExtensionFilter("Text Files", "*.txt")
 event.consume()
 val filechooser: FileChooser = FileChooser()
 filechooser.extensionFilters.add(ex1)
 val selectedfile: File? =
filechooser.showSaveDialog(node.scene.window)
 if(selectedfile != null){
  SaveGraph(selectedfile.path)
 }

}

private fun SaveGraph(path: String){ ///доработать
 var file: BufferedWriter = File(path).bufferedWriter()
 var size = model.getAllEdges().size
 var singlecells: MutableList<Int> = getSingleCell()
 size += singlecells[singlecells.size - 1]
 file.write("${size}\n")
 for(i in  model.getAllEdges()){
  file.write("${i.getsource().getcellId()}
${i.gettarget().getcellId()} ${i.getweight()}\n")
 }
 for(i in 0 until size - model.getAllEdges().size){
  file.write("${singlecells[i]} ${singlecells[i]} ${0}")
 }
 file.close()
}
private fun getSingleCell(): MutableList<Int>{
```

```
 var cellslist: MutableList<Int> = mutableListOf()
 var res = 0
 for(i in model.getallCells()){
  if(i.getCellChildren().isEmpty()){
   cellslist.add(i.getcellId().toInt())
   res++
  }

 }
 cellslist.add(res)
 return cellslist
}

@FXML
fun initialize() {
 model.log = Logger(my_console)
}
@FXML
private fun open_file(event: ActionEvent){
 val ex1: FileChooser.ExtensionFilter =
FileChooser.ExtensionFilter("Text Files", "*.txt")

 event.consume()
 val filechooser: FileChooser = FileChooser()
 filechooser.extensionFilters.add(ex1)
 val selectedfile: File? =
filechooser.showOpenDialog(node.scene.window)
 if(selectedfile != null){
  take_graph(selectedfile.path)
 }
}
private fun take_graph(path: String){
try {
 layout.set_param(scroll_pane.width,scroll_pane.height)
```

```
 val inputStream: InputStream = File(path).inputStream()
 val lineList = mutableListOf<String>()
 inputStream.bufferedReader().forEachLine { lineList.add(it) }
 val num_edge: Int = lineList[0].toInt()
 var source: Int = 0
 var target: Int = 0
 var weight: Int = 0
 var coordinate: MutableList<Double> = mutableListOf()
 var edge: VXEdge
 var sourcecell: VXCell
 var targetcell: VXCell
 for(i in 1..(num_edge)){
  var strs = lineList[i].split(" ").toTypedArray()
  source = strs[0].toInt()
  target = strs[1].toInt()
  weight = strs[2].toInt()
  if(source != target){    //разные вершины

      if(!cellmap.containsKey(source)) {
       coordinate = layout.execute()
       sourcecell = VXCell(source.toString(), coordinate[0],
coordinate[1]) //coordinate[0] coordinate[1]
       cellmap[source] = sourcecell
       sourcecell.setLabel(Text(source.toString()))
       buff_vxcells.add(sourcecell)
      }
      else{
         sourcecell = cellmap[source]!!
      }
      if(!cellmap.containsKey(target)){
       coordinate = layout.execute()
       targetcell = VXCell(target.toString(),coordinate[0],
coordinate[1]) //coordinate[0] coordinate[1]
       cellmap[target] = targetcell
```

```
        targetcell.setLabel(Text(target.toString()))
        buff_vxcells.add(targetcell)
       }
       else{
        targetcell = cellmap[target]!!
       }
       edge = VXEdge(sourcecell,targetcell,weight) //
добавление ребра
       edge.set_label(Text(weight.toString()))
       sourcecell.edgesmap[targetcell] = edge
       sourcecell.neighbors.add(targetcell)
       sourcecell.weightmap[targetcell] = weight
       targetcell.edgesmap[sourcecell] = edge
       targetcell.neighbors.add(sourcecell)
       targetcell.weightmap[sourcecell] = weight
       buff_vxedges.add(edge)
  }
  else{
   if(!cellmap.containsKey(source)) {
    coordinate = layout.execute()
    sourcecell = VXCell(source.toString(), coordinate[0],
coordinate[1]) // coordinate[0] coordinate[1]
    cellmap[source] = sourcecell
    sourcecell.setLabel(Text(source.toString()))
    buff_vxcells.add(sourcecell)
   }
  }
 }
 cell_number = max(cellmap.keys) + 1
 clear_holst()
 for(i in buff_vxcells) {
  vxcells.add(i)
  holst.children.add(i)
  holst.children.add(i.get_lable())
```

```
  model.addCell(i)
  model.merge()
 }
 for(i in buff_vxedges){
  vxedges.add(i)
  holst.children.add(i.get_label())
  holst.children.add(i)

model.addEdge(i.getsource().getcellId(),i.gettarget().getcellI
d(),weight,i)
  model.merge()
 }
 layout.reset()
 inputStream.close()
}
catch(e: IOException){
 val stage = Stage()
 val loader =
FXMLLoader(javaClass.getResource("readfile_exception.fxml"))
 val root = loader.load<Parent>()
 stage.title = "Exception"
 stage.scene = Scene(root, 400.0, 200.0)
 stage.initModality(Modality.APPLICATION_MODAL)
 stage.showAndWait()
}
catch(e: java.lang.Exception){
 val stage = Stage()
 val loader =
FXMLLoader(javaClass.getResource("wrong_data.fxml"))
 val root = loader.load<Parent>()
 stage.title = "Exception"
 stage.scene = Scene(root, 400.0, 200.0)
 stage.initModality(Modality.APPLICATION_MODAL)
 stage.showAndWait()
```

```
 for(i in buff_vxcells){
   cellmap.remove(i.cell.getcellId().toInt())
 }
}
finally {
    buff_vxcells.clear()
    buff_vxedges.clear()
}

}
@FXML
fun clear_holst() {
 holst.children.removeAll(vxedges)
 holst.children.removeAll(vxcells)
 for(e in vxedges){
  holst.children.remove(e.get_label())
 }
 vxedges.clear()
 for(cell in vxcells){
  holst.children.remove(cell.get_lable())
 }
 vxcells.clear()
 model.clear_all()
 cellmap.clear()
 cell_number = 1
 model.removecomponents()
 off_parametrs()
 my_console.clear()
}

private fun off_parametrs(){
 for (cell in vxcells){
  cell.disableDrag()
 }
```

```
  del_edge_button.style = "-fx-background-color: #ffa000"
add_vert_button.style = "-fx-background-color: #ffa000"
add_edge_button.style = "-fx-background-color: #ffa000"
del_vert_button.style = "-fx-background-color: #ffa000"
 edge.clear()
 scroll_pane.onMouseClicked = null
}
@FXML
private fun add_vert(event: ActionEvent) {
 event.consume()
 if (add_vert_button.style == "-fx-background-color: green") {
// если включена
  off_parametrs() // сброс всего
  add_vert_button.style = "-fx-background-color: #ffa000"

  for(cell in vxcells){
   cell.enableDrag()
  }
 } else {                                                    //
включаем

  off_parametrs()
//сброс всех настроек
  add_vert_button.style = "-fx-background-color: green"
  scroll_pane.onMouseClicked =
onMousePressedEventHandler_add_vert
 }
}
var onMousePressedEventHandler_add_vert =
 EventHandler<MouseEvent> { event ->
  val node = event.source as Node
  val x = event.x  // присвоение координат со смещением
радиуса
  val y = event.y
```

```
  val cell: VXCell = VXCell(cell_number.toString(),x,y)
  cell.setLabel(Text(cell_number.toString()))
  vxcells.add(cell)
  cell_number += 1 // новое имя для другой вершины
  holst.children.add(cell) // добавить на холст
  holst.children.add(cell.get_lable())
  model.addCell(cell) // добавить в структуру
  model.merge()
 }
////////////////////////////////////////////////////////////
//////////////////////
@FXML
private fun del_vert(event: ActionEvent) {
 event.consume()
 if(del_vert_button.style == "-fx-background-color: green") {
// если включена
  off_parametrs() // сброс всего
  del_vert_button.style = "-fx-background-color: #ffa000"
  for(cell in vxcells){
   cell.enableDrag()
  }
 }
 else{
  off_parametrs()
  del_vert_button.style = "-fx-background-color: green"
  for(cell in vxcells){
   cell.onMousePressed = onMousePressed_del_vert
  }
 }
}


var onMousePressed_del_vert = EventHandler<MouseEvent> {event
->
```

```
 val node = event.source as VXCell
 model.remove_cell(node)
 cleaning(node)
 node.remove_connection()
 for(e in node.edgesmap.values){

  holst.children.remove(e)
  holst.children.remove(e.get_label())
 }
 vxcells.remove(node)
 holst.children.remove(node.get_lable())
 holst.children.remove(node)
}
private fun set_default(){
  for(i in vxcells){
    i.fill = Color.GREEN
  }
 for(i in vxedges){
   i.stroke = Color.BLACK
  i.style = "-fx-stroke-width: 1px"
 }
 model.removecomponents()
 my_console.clear()
 create_components = false
 result_button.setOnAction { result(event = ActionEvent())}
 }
private fun cleaning(cell: VXCell){

 for(neighbor in cell.neighbors){
  val cur = cell.weightmap[neighbor]
  val e: VXEdge? = cell.weightmap[neighbor]?.let {
VXEdge(cell,neighbor, it) }
  val second_e: VXEdge? = cell.weightmap[neighbor]?.let {
VXEdge(neighbor,cell, it) }
```

```
  cur?.let { Edge(cell.cell,neighbor.cell, it) }?.let {
   if (e != null) {
    model.del_edge(it,e)
   }
  }
  cur?.let { Edge(neighbor.cell,cell.cell, it) }?.let {
   if (second_e != null) {
    model.del_edge(it,second_e)
   }
  }
 }
}
////////////////////////////////////////////////////////////
///////////////
@FXML
private fun add_edge(event: ActionEvent) {
 //event.consume()
 if (add_edge_button.style == "-fx-background-color: green") {
  off_parametrs()
  add_edge_button.style = "-fx-background-color: #ffa000"
  for(cell in vxcells){
   cell.enableDrag()
  }
 } else {
  off_parametrs()
  add_edge_button.style = "-fx-background-color: green"
  for(cell in vxcells){
   cell.disableDrag()
   cell.onMousePressed = onMousePressed_add_edge
   cell.onMouseDragged = onMousePressed_add_edge
  }
 }
}
```

```
var onMousePressed_add_edge = EventHandler<MouseEvent> {event
->
 val node = event.source as VXCell
 node.stroke = Color.FIREBRICK
 edge.add(node)
 if(edge.size == 2){
  if(edge[0].getcellId().toInt() !=
edge[1].getcellId().toInt()){
   val stage = Stage()
   val loader =
FXMLLoader(javaClass.getResource("edge_weight_dialog.fxml"))
   val root = loader.load<Parent>()
   val lc: NewFolderController = loader.getController<Any>()
as NewFolderController
   lc.main = this
   stage.title = "Create folder"
   stage.scene = Scene(root, 400.0, 200.0)
   stage.initModality(Modality.APPLICATION_MODAL)
   stage.showAndWait()
   if(edge[0].getcellId().toInt() !=
edge[1].getcellId().toInt()){
    val cur: Int? = edge[0].weightmap[edge[1]]
    val e: VXEdge? = edge[0].weightmap[edge[1]]?.let {
VXEdge(edge[0],edge[1], it) }
    val second_e: VXEdge? = edge[0].weightmap[edge[1]]?.let {
VXEdge(edge[1],edge[0], it) }
    edge[0].neighbors.remove(edge[1])
    holst.children.remove(edge[0].edgesmap[edge[1]])
    holst.children.remove(edge[1].edgesmap[edge[0]])

holst.children.remove(edge[0].edgesmap[edge[1]]?.get_label())

holst.children.remove(edge[1].edgesmap[edge[0]]?.get_label())
    edge[0].edgesmap.remove(edge[1])
```

```
 edge[0].weightmap.remove(edge[1])
 edge[1].neighbors.remove(edge[0])
 edge[1].edgesmap.remove(edge[0])
 edge[1].weightmap.remove(edge[0])
 cur?.let { Edge(edge[0].cell,edge[1].cell, it) }?.let {
  if (e != null) {
   model.del_edge(it,e)
  }
 }
 cur?.let { Edge(edge[1].cell,edge[0].cell, it) }?.let {
  if (second_e != null) {
   model.del_edge(it,second_e)
  }
 }

}

val e: VXEdge = VXEdge(edge[0],edge[1],weight)
edge[0].stroke = Color.GREEN
edge[1].stroke = Color.GREEN
e.set_label(Text(weight.toString()))
vxedges.add(e)
edge[0].edgesmap[edge[1]] = e
// edge[0].edges.add(e)
edge[0].neighbors.add(edge[1])
edge[0].weightmap[edge[1]] = weight
edge[1].edgesmap[edge[0]] = e
// edge[1].edges.add(e)
edge[1].neighbors.add(edge[0])
edge[1].weightmap[edge[0]] = weight
holst.children.add(e.get_label())
holst.children.add(e)
```

```
model.addEdge(e.getsource().getcellId(),e.gettarget().getcellI
d(),weight,e)
   model.merge()
  }
  edge.clear()
 }
 event.consume()
}
@FXML
private fun del_edge(event: ActionEvent) {
 event.consume()
 if (del_edge_button.style == "-fx-background-color: green") {
  off_parametrs()
  del_edge_button.style = "-fx-background-color: #ffa000"
  for (cell in vxcells) {
   cell.enableDrag()
  }
 } else {
  off_parametrs()
  del_edge_button.style = "-fx-background-color: green"
  for(cell in vxcells){
   cell.disableDrag()
   cell.onMousePressed = onMousePressed_del_edge
  }
 }
}

var onMousePressed_del_edge = EventHandler<MouseEvent> {event
->
 val node = event.source as VXCell
 node.stroke = Color.FIREBRICK
 edge.add(node)
 if(edge.size == 2){
```

```
  println("del")
  if(edge[0].getcellId().toInt() !=
edge[1].getcellId().toInt()){
   val cur: Int? = edge[0].weightmap[edge[1]]
   val e: VXEdge? = edge[0].weightmap[edge[1]]?.let {
VXEdge(edge[0],edge[1], it) }
   val second_e: VXEdge? = edge[0].weightmap[edge[1]]?.let {
VXEdge(edge[1],edge[0], it) }
   edge[0].neighbors.remove(edge[1])
   holst.children.remove(edge[0].edgesmap[edge[1]])
   holst.children.remove(edge[1].edgesmap[edge[0]])

holst.children.remove(edge[0].edgesmap[edge[1]]?.get_label())

holst.children.remove(edge[1].edgesmap[edge[0]]?.get_label())
   edge[0].edgesmap.remove(edge[1])
   edge[0].weightmap.remove(edge[1])
   edge[1].neighbors.remove(edge[0])
   edge[1].edgesmap.remove(edge[0])
   edge[1].weightmap.remove(edge[0])
   cur?.let { Edge(edge[0].cell,edge[1].cell, it) }?.let {
    if (e != null) {
     model.del_edge(it,e)
     edge[0].cell.removeCellChild(edge[1].cell)
     edge[1].cell.removeCellChild(edge[0].cell)
     for(i in model.getallCells()){
      if(i.getcellId() == edge[0].cell.getcellId() ){
       i.removeCellChild(edge[1].cell)
      }
     }

    }
   }
   cur?.let { Edge(edge[1].cell,edge[0].cell, it) }?.let {
```

```
    if (second_e != null) {
     model.del_edge(it,second_e)
    }
   }

  }
  edge.clear()
 }
 event.consume()
}

@FXML
private fun result(event: ActionEvent) {
 off_parametrs()
 del_edge_button.onAction = null
 add_vert_button.onAction = null
 add_edge_button.onAction = null
 del_vert_button.onAction = null
 for(i in vxcells){
  i.enableDrag()
 }
 if(model.getAllComponents().isEmpty()){

  create_components = model.createAllComponents()
 }
 if(create_components){
  model.BoruvkaMST(1)
  if(model.end_flag == true) {                //алгоритм
завершился
   model.end_flag = false
   result_button.text = "Заново"
   result_button.setOnAction(EventHandler<ActionEvent?> {
    step_forward_button.setOnAction { step_forward(event =
ActionEvent()) }
```

```
    result_button.text = "Результат"
    set_default()
    add_edge_button.setOnAction { add_edge(event =
ActionEvent()) }
    del_edge_button.setOnAction {del_edge(event =
ActionEvent())  }
    add_vert_button.setOnAction {add_vert(event =
ActionEvent())  }
    del_vert_button.setOnAction { del_vert(event =
ActionEvent()) }
   })


  }
 }

}
@FXML
private fun step_forward(event: ActionEvent)  {
 off_parametrs()
 del_edge_button.onAction = null
 add_vert_button.onAction = null
 add_edge_button.onAction = null
 del_vert_button.onAction = null
 for(i in vxcells){
  i.enableDrag()
 }
 if(model.getAllComponents().isEmpty()){
  create_components = model.createAllComponents()
 }
 if(create_components){
  model.BoruvkaMST(0)
  if(model.end_flag == true) {                  //алгоритм
завершился
```

```
    step_forward_button.onAction = null
    model.end_flag = false
    result_button.text = "Заново"
    result_button.setOnAction(EventHandler<ActionEvent?> {
     result_button.text = "Результат"
     set_default()
     step_forward_button.setOnAction { step_forward(event =
ActionEvent())}
     add_edge_button.setOnAction { add_edge(event =
ActionEvent()) }
     del_edge_button.setOnAction {del_edge(event =
ActionEvent())  }
     add_vert_button.setOnAction {add_vert(event =
ActionEvent())  }
     del_vert_button.setOnAction { del_vert(event =
ActionEvent()) }
     })
   }
  }
}
}
```

Название файла: Main.kt

```
package com.example.refactor_vizualize_graph_ktln

import com.fxgraph.graph.Model
import com.fxgraph.layout.RandomLayout
import javafx.application.Application
import javafx.fxml.FXMLLoader
import javafx.scene.Scene
import javafx.scene.layout.BorderPane
import javafx.stage.Stage

class Main : Application() {
```

```
    override fun start(stage: Stage) {
        val fxmlLoader =
FXMLLoader(Main::class.java.getResource("hello-view.fxml"))
        val scene = Scene(fxmlLoader.load(), 900.0, 600.0)
        stage.title = "Hello!"
        stage.scene = scene
        stage.show()
    }


    fun main(){
        Application.launch(Main::class.java)
    }

}
```

Название файла: Main_1.kt

```
package com.example.refactor_vizualize_graph_ktln

class Main_1 {

   companion object {

       @JvmStatic
       fun main(args: Array<String>) {
           Main().main()
       }
   }

}
```

Название файла: Cell.kt

```
package com.fxgraph.graph

import javafx.scene.Node
import javafx.scene.layout.Pane
```

```
import javafx.scene.shape.Circle

open class Cell(private val cellId: String = ""): Pane() {
   private var children = mutableMapOf<Cell, Int>()
   lateinit var view: Node
   fun addCellChild(cell: Cell, weight : Int) {
       children[cell] = weight // добавить ребро исходящее
       cell.children[this] = weight
   }
   fun getCellChildren(): MutableMap<Cell, Int> {
       return children // вернуть список исходящих вершин
   }

   fun removeCellChild(cell: Cell) {
       cell.children.remove(this)
       children.remove(cell); // удалить исходящее ребро
   }


   fun setview(view: Node) {

       this.view = view //присвоить отображение
       getChildren().add(view) // разместить на холсте

   }

   fun getview(): Node {
       return this.view; // вернуть отображение
   }

   fun getcellId(): String {
       return cellId; // вернуть имя
   }
}
```

Название файла: Component.kt

```
package com.fxgraph.graph


import com.fxgraph.vizualization.VXCell
import com.fxgraph.vizualization.VXEdge
import kotlin.math.min

class Component {
   private var Cells: MutableList<Cell> = mutableListOf() //
список всех вершин
   private var VXCells: MutableList<VXCell> = mutableListOf()
   private var allEdges: MutableList<Edge> = mutableListOf()
// список рёбер внутри компоненты
   private var allVXEdges: MutableList<VXEdge> =
mutableListOf()
   private var edges = mutableMapOf<Cell, Int>() // список
рёбер соединяющий разные компоненты

   fun getCells() = Cells
   fun getAllEdges() = allEdges
   fun getEdges() = edges

   fun checkComponentCell(name: String): Int {
       var num = 0
       for (i in Cells) {
           if (name == i.getcellId())
               return num
           num++
       }
       return Cells.size
   }
```

```
    fun printneigbors():String{
        var str1 = ""
        var str2 = ""
        for (i in edges.keys) {
            str1 += i.getcellId() + " "
        }
        for (i in edges.values) {
            str2 += "$i "
        }
        return "\n[${str1.trim()}]\n[${str2.trim()}]"
    }


    fun removeEdge(temp: Component){
        for (i in temp.Cells)
            this.edges.remove(i)}

    fun addCells(temp: Cell){Cells.add(temp)}
    fun addVXCells(temp: VXCell){VXCells.add(temp)}
    fun addAllEdges(temp: Edge,cur: VXEdge){
        allEdges.add(temp)
        cur.stroke = VXCells[0].fill
        cur.style = "-fx-stroke-width: 4px"
        allVXEdges.add(cur)
    }
    fun addEdges(temp: Cell, num: Int){edges[temp] = num}

    fun setCells(temp: MutableList<Cell>) {Cells = temp}
    fun setEdges(temp: MutableList<Edge>) {allEdges = temp}
    fun setAllEdges(temp: MutableMap<Cell, Int>)
{edges.putAll(temp)}

    fun printallCells(): String {
        var res: String = ""
```

```
        for (i in Cells){
            res += i.getcellId()+" "
        }
        return res.trim()
    }

    fun mergeComp(temp: Component,log:Logger){
        log.log("\n---Начало слияния двух
компонент-связности---")//-------
        log.log("Вершины первой компоненты, в которую войдут
вершины и ребра второй - [${this.printallCells()}]")//  Вывод
компонент свзяности
        log.log("Вершины второй компоненты, которая будет
поглощена первой - [${temp.printallCells()}]")//------
        for (i in temp.Cells){
            this.Cells.add(i)
        }
        for(i in temp.VXCells){
            i.fill = this.VXCells[0].fill
            this.VXCells.add(i)
        }
        for (i in temp.allEdges){
            this.allEdges.add(i)
        }
        for(i in temp.allVXEdges){
            i.stroke = this.VXCells[0].fill
            this.allVXEdges.add(i)
        }

        var keys_1 = this.edges.keys
        var keys_2 = temp.edges.keys
```

```
        log.log("Первая компонента соединена с другими через
следующие ребра, конечные вершины и их
вес:${this.printneigbors()}")
        log.log("Вторая компонента соединена с другими через
следующие ребра, конечные вершины и их
вес:${temp.printneigbors()}")

        for (i in 0 until keys_2.size){
            if (keys_2.elementAt(i) in keys_1){
                if (this.edges.containsKey(keys_2.elementAt(i))
&& temp.edges.containsKey(keys_2.elementAt(i))){
                    var min1 =
min(this.edges[keys_2.elementAt(i)]!!,
temp.edges[keys_2.elementAt(i)]!!)
                    this.edges.remove(keys_2.elementAt(i))
                    this.edges[keys_2.elementAt(i)] = min1}
            } else {
                temp.edges[keys_2.elementAt(i)]?.let {
this.edges.put(keys_2.elementAt(i), it) }
            }
        }

        log.log("Список ребер получившейся компоненты, с
помощью которых она соединена с другими, их конечные вершины и
вес:${this.printneigbors()}")

    }

}
```

Название файла: Edge.kt

```
package com.fxgraph.graph

import com.fxgraph.vizualization.VXCell
import javafx.scene.Group
```

```
import javafx.scene.shape.Line


class Edge(protected var source: Cell,protected var target:
Cell, protected var weight: Int): Group() {
   private val line: Line = Line()
   init{
       source.addCellChild(target, weight)
   }
   fun getsource(): Cell { //получить стартовую
       return source;
   }

   fun gettarget(): Cell { // получить конечную
       return target;
   }

   fun getweight(): Int {
       return weight
   }

}
```

Название файла: Logger.kt

```
package com.fxgraph.graph
import javafx.scene.control.TextArea
import java.io.*
import java.nio.charset.Charset
class Logger(var console: TextArea) {
   fun log(temp:String){
       println(temp)
       console.appendText(temp + "\n")
   }
}
```

Название файла: Model.kt

```
package com.fxgraph.graph

import java.util.Random
import com.fxgraph.vizualization.VXCell
import com.fxgraph.vizualization.VXEdge
import javafx.scene.paint.Color
import kotlin.collections.HashMap

class Model {
   var end_flag: Boolean = false
   private var index = 0
   private var dif_colors: MutableList<Color> =
mutableListOf()
   private var rand: Random = Random()
   private val graphParent: Cell = Cell("_ROOT_") // корневой
узел невидимый
   private var allCells: MutableList<Cell> = mutableListOf()
// список всех вершин
   private var allVXCells: MutableList<VXCell> =
mutableListOf()
   private var addedCells: MutableList<Cell> = mutableListOf()
// добавленные вершины
   private var addedVXCells: MutableList<VXCell> =
mutableListOf()
   private var removedCells: MutableList<Cell> =
mutableListOf() // удалённые вершины
   private var removedVXCells: MutableList<VXCell> =
mutableListOf()
   private var allEdges: MutableList<Edge> = mutableListOf()
// список рёбер
   private var allVXEdges: MutableList<VXEdge> =
mutableListOf()
```

```
   private var addedEdges: MutableList<Edge> = mutableListOf()
// список добавленных рёбер
   private var addedVXEdges: MutableList<VXEdge> =
mutableListOf()
   private var removedEdges: MutableList<Edge> =
mutableListOf() // список удалённых рёбер
   private var removedVXEdges: MutableList<VXEdge> =
mutableListOf()
   private var cellMap: HashMap<String,Cell> = hashMapOf()  //
<id,cell> доступ к вершине по имени
   private var allComponents: MutableList<Component> =
mutableListOf() // список компонент связности
   var n = 0
   lateinit var log: Logger
   fun
setAllComponents(temp:MutableList<Component>){allComponents =
temp.toMutableList()}
   fun getAllComponents():MutableList<Component>{return
allComponents}

   fun clearAddedLists() { //очистить список рёбер и вершин
(полное удаление)
       addedCells.clear();
       addedEdges.clear();
   }

   fun  getaddedCells(): MutableList<Cell> { // получить
список добавленных вершин
       return addedCells;
   }
   private fun get_color(): Color{

       var r: Double = rand.nextFloat() / 2f + 0.5
       var g: Double = rand.nextFloat() / 2f + 0.5
```

```
        var b: Double = rand.nextFloat() / 2f + 0.5
        var col: Color = Color.color(r,g,b)
        var first_expr: Boolean = col == Color.GREEN
        var second_expr: Boolean = col == Color.WHITE
        while (dif_colors.contains(col) == true  or first_expr
or second_expr ){
            r = rand.nextFloat() / 2f + 0.5
            g = rand.nextFloat() / 2f + 0.5
            b = rand.nextFloat() / 2f + 0.5
            col = Color.color(r,g,b)
        }
        dif_colors.add(col)
      //  var col: Color = Color(r,g,b)

        return  col
    }
    fun getremovedCells(): MutableList<Cell> { // получить
список удалённых вершин
        return removedCells;
    }

    fun getallCells(): MutableList<Cell> { // получить список
всех вершин
        return allCells;
    }

    fun getAddedEdges(): MutableList<Edge> { // получить список
добавленных рёбер
        return addedEdges;
    }

    fun getRemovedEdges(): MutableList<Edge> { // получить
список удалённых рёбер
        return removedEdges;
```

```
    }
    fun getAllEdges(): MutableList<Edge> { // получить список
всех рёбер
        return allEdges;
    }

    fun addCell(cell: VXCell) {  // приватный метод

        addedCells.add(cell.cell) // добавить в список
добавленных
        addedVXCells.add(cell)
        cellMap[cell.getcellId()] = cell.cell // присвоить пару
ключ-значение
        merge()
    }
    fun del_edge(edge: Edge,cur: VXEdge){
        removedEdges.add(edge)
        removedVXEdges.add(cur)
        for(i in 0..getallCells().size - 1){
            if(allCells[i].getcellId() ==
edge.getsource().getcellId()){
                allCells[i].removeCellChild(edge.gettarget())
            }
            if(allCells[i].getcellId() ==
edge.gettarget().getcellId()){
                allCells[i].removeCellChild(edge.getsource())
            }
        }
        edge.getsource()
        merge()
    }

    fun addEdge( sourceId: String, targetId: String, weight :
Int, cur: VXEdge) { // добавить ребро
```

```
        val sourceCell: Cell = cellMap[sourceId]!!; //
получение вершин по имени
        val targetCell: Cell = cellMap[targetId]!!;

        val edge: Edge = Edge( sourceCell, targetCell, weight);
// создание ребра

        addedEdges.add(edge); // добавление ребра в список
        addedVXEdges.add(cur)
    }


    fun disconnectFromGraphParent(cellList: MutableList<Cell>)
{ // удалить вершину от корня

        for(cell in cellList) {
            graphParent.removeCellChild( cell);
        }
    }




    fun findEdgeinList(temp1: Cell, temp2: Cell, weight: Int):
Edge? {
        for(i in getAllEdges()){
            if ((i.getsource() == temp1 && i.gettarget() ==
temp2 || i.getsource() == temp2 && i.gettarget() == temp1) &&
weight == i.getweight())
                return i
            index += 1
        }
        return null
```

```
    }

    fun checkEdge(temp1: Cell, temp2: Cell, weight: Int):
Boolean{
        for(i in getAllEdges()){
            if ((i.getsource() == temp1 && i.gettarget() ==
temp2 || i.getsource() == temp2 && i.gettarget() == temp1) &&
weight == i.getweight())
                return true
        }
        return false
    }

    fun findEdge(comp: Component, temp: Cell, weight: Int):
Edge? {
        for (i in comp.getCells())
            if (checkEdge(i,temp, weight))
                return findEdgeinList(i, temp, weight)
        return null
    }

    fun findComp(): Component?{
        if(getAllComponents().size==1) {
            return getAllComponents()[0]
        }
        for (i in getAllComponents())
            if (i.getEdges().isEmpty())
                return i
        return null
    }
    fun merge() {

        // cells
```

```
        allCells.addAll( addedCells); // добавить список
добавленных вершин во все
        allCells.removeAll( removedCells); // удалить все
удалённые
        allVXCells.addAll(addedVXCells)
        allVXCells.removeAll(removedVXCells)
        addedCells.clear();
        addedVXCells.clear()
        removedCells.clear(); // очистить буфферы
        removedVXCells.clear()
        // edges
        allEdges.addAll( addedEdges);
        allEdges.removeAll( removedEdges); // сделать также с
рёбрами
        allVXEdges.addAll(addedVXEdges)
        allVXEdges.removeAll(removedVXEdges)
        addedEdges.clear();
        addedVXEdges.clear()
        removedEdges.clear(); // очистить буфферы
        removedVXEdges.clear()

    }
    fun remove_cell(cell:VXCell){
        removedCells.add(cell.cell)
        removedVXCells.add(cell)
        merge()
    }

    fun clear_all(){
        allCells.clear()// список всех вершин
        allVXCells.clear()
        addedCells.clear()  // добавленные вершины
        addedVXCells.clear()
        removedCells.clear() // удалённые вершины
```

```
        removedVXCells.clear()
        allEdges.clear() // список рёбер
        allVXEdges.clear()
        addedEdges.clear()// список добавленных рёбер
        addedVXEdges.clear()
        removedEdges.clear() // список удалённых рёбер
        removedVXEdges.clear()
        cellMap.clear()  // <id,cell> доступ к вершине по имени
    }
    fun getallVXCells(): MutableList<VXCell>{
        return allVXCells
    }
    fun createAllComponents():Boolean {
        if (this.getAllEdges().isNotEmpty() &&
getAllComponents().size!=1) {
            if (getAllComponents().isEmpty()) {
                for (i in 0 until allCells.size) {
                    var temp = Component()
                    temp.addCells(getallCells()[i]) // создал
компоненты (перекрасить все вершины)
                    ///
                    getallVXCells()[i].fill = get_color() //
покраска вершин
                    ///
                    temp.addVXCells(getallVXCells()[i])


temp.setAllEdges(getallCells()[i].getCellChildren())
                    getAllComponents().add(temp)
                }
                return true
            }
            return true
        }
```

```
        return false
    }
    fun removecomponents(){
        allComponents.clear()
        n = 0
    }
    fun stepAlgoritm(){
        var p = getAllComponents()[n]
        log.log("Вершины, из которых состоит рассматриваемая
компонента-связности - [${p.printallCells()}]\n")
        var destination = Cell()
        var num: Int = 0
        log.log("---Процесс нахождения минимального ребра,
через которое текущая компонента свазана с другой---")
        for (i in p.getEdges()) {
            log.log("Конечная вершина рассматримаевого ребра,
находящаяся в другой компоненте: ${i.key.getcellId()} \nВес
этого ребра: ${i.value}")
            if (destination.getcellId().isEmpty()) {
                destination = i.key
                num = i.value
            } else if (i.value < num) {
                num = i.value
                destination = i.key
            }
            log.log("Конечная вершина текущего минимального
ребра, которая находится в другой компоненте:
${destination.getcellId()} \nТекущий минимальный вес ребра:
$num \n")
        }
        log.log("Найденная конечная вершина минимального ребра:
${destination.getcellId()}\nВес этого ребра: $num\n")
        log.log("---Нахождение компоненты, с которой произойдет
слияние с текущей, через найденное минимальное ребро---")
```

```
        for (t in getAllComponents()) {
            log.log("Рассматриваемая для слияния компонента
состоит из следующих вершин: [${t.printallCells()}]")
            if (t.checkComponentCell(destination.getcellId())
!= t.getCells().size) {
                log.log("Последняя рассматриваемая компонента
содержит нужную вершину - \"${destination.getcellId()}\"")
                t.removeEdge(p)
                p.removeEdge(t)
                p.mergeComp(t,log) // перекрасить всё множество
t в цвет множества p
                findEdge(p, destination, num)?.let {
p.addAllEdges(it,allVXEdges[index]) } // перекраска ребра
                index = 0
                getAllComponents().remove(t)
                break
            }
        }
        n++
        if (n > getAllComponents().size - 1) n = 0
        log.log("==========\n")
    }
    fun printresult(comp:Component){
        log.log("---Итог - Ребра из которх состоит минимальное
остовное дерево---")
        for (i in comp.getAllEdges())
            log.log("${i.getsource().getcellId()}
${i.getweight()} ${i.gettarget().getcellId()}")
        log.log("=====")
        end_flag = true
    }
    fun BoruvkaMST(index:Int){
        if(index == 1) {
```

```
            while (getAllComponents().size > 1 &&
checkEdgesOut()) {
                stepAlgoritm()
            }
            findComp()?.let { printresult(it) }
        }else if(index == 0 ){
            if (checkEdgesOut()){
                stepAlgoritm()
            }
            if (getAllComponents().size == 1 ||
!checkEdgesOut())
                findComp()?.let { printresult(it) }
        }
    }
    fun checkEdgesOut(): Boolean {
        for (i in getAllComponents())
            if (i.getEdges().isEmpty())
                return false
        return true
    }
}
```

Название файла: RandomLayout.kt

```
package com.fxgraph.layout

import java.util.Random


class RandomLayout() { //  граф
    var rnd: Random = Random() // класс рандома

    fun execute(): MutableList<Double> {
            var coordinate: MutableList<Double> =
mutableListOf()
```

```
            var x: Double = rnd.nextDouble() * 500
            var y: Double = rnd.nextDouble() * 500
            coordinate.add(x)
            coordinate.add(y)
            return coordinate
    }
    }
```

Название файла: SquareLayout.kt

```
package com.fxgraph.layout

class SquareLayout {  // 6x6
    var step_width: Double = 0.0
    var step_height: Double = 0.0
    val separ: Int = 5
    private var width_val = 0.0
    private var height_val = 0.0
    var x = 0.0
    var y = 0.0
    fun set_param(width: Double,height: Double){
        width_val = width
        height_val = height
        step_width = width / separ - 15
        step_height = height / separ + 15
        y = step_height
    }
    fun execute(): MutableList<Double> {
        var coordinate: MutableList<Double> = mutableListOf()
        if(x > width_val - 20 ){
            x = step_width
            y += step_height
        }
        else{
            x += step_width
```

```
        }
        coordinate.add(x)
        coordinate.add(y)
        return coordinate
    }
    fun reset(){
        x = 0.0
        y = 0.0
    }
}
```

Название файла: VXCell.kt

```
package com.fxgraph.vizualization

import com.fxgraph.graph.Cell
import javafx.event.EventHandler
import javafx.geometry.Point2D
import javafx.scene.Cursor
import javafx.scene.input.MouseEvent
import javafx.scene.paint.Color
import javafx.scene.shape.Circle
import javafx.scene.text.Font
import javafx.scene.text.FontPosture
import javafx.scene.text.FontWeight
import javafx.scene.text.Text

class VXCell(var cellId: String,x: Double,y:Double):
Circle(x,y,15.0) {
    private var name: Text = Text(cellId)
    var edgesmap: MutableMap<VXCell, VXEdge> = hashMapOf()
    var weightmap: MutableMap<VXCell, Int> = hashMapOf()
    var neighbors: MutableList<VXCell> = mutableListOf()
    var cell: Cell = Cell(cellId)
    var isDragging: Boolean = false
```

```
init {
    stroke = Color.GREEN
    fill = Color.GREEN
    enableDrag()
    relocate(x,y)


}

fun getcellId(): String {
    return cellId
}

fun get_lable(): Text {
    return name
}

/**
 * Gets position of the center of this vertex view
 *
 * @return position of the center of this vertex view
 */
fun getPosition(): Point2D? {
    return Point2D(centerX, centerY)
}

fun remove_connection() {
    for (neighbr in neighbors) {
        val cur = this.weightmap[neighbr]
        neighbr.edgesmap.remove(this)
        neighbr.neighbors.remove(this)
        this.weightmap.remove(neighbr)
```

```
        }

    }

    fun setPosition(x: Double, y: Double) {
        if (isDragging) {
            return
        }
        centerX = x
        centerY = y
    }

    fun edge_add_config() {
        disableDrag()
        onMousePressed = EventHandler { event: MouseEvent ->
            this.stroke = Color.FIREBRICK
        }

    }

    /**
     * Enables user drag
     */

    fun enableDrag() {
        class Point(var x: Double, var y: Double)

        val dragDelta = Point(0.0, 0.0)
        onMousePressed = EventHandler { event: MouseEvent ->
            if (event.isPrimaryButtonDown) {


name.xProperty().bind(centerXProperty().subtract(name.layoutBo
unds.width +radius))
```

```
name.yProperty().bind(centerYProperty().add(name.layoutBounds.
height + radius))
                dragDelta.x = centerX - event.x
                dragDelta.y = centerY - event.y
                scene.cursor = Cursor.MOVE
                isDragging = true
                event.consume()
            }
        }
        onMouseReleased = EventHandler { event: MouseEvent ->
            scene.cursor = Cursor.HAND
            isDragging = false
            event.consume()
        }
        onMouseDragged = EventHandler { event: MouseEvent ->
            if (event.isPrimaryButtonDown) {
                val newX = event.x + dragDelta.x
                val x = boundCenterCoordinate(newX, 0.0,
parent.layoutBounds.width)
                centerX = x
                val newY = event.y + dragDelta.y
                val y = boundCenterCoordinate(newY, 0.0,
parent.layoutBounds.height)
                centerY = y
                this.relocate(x, y) /////
            }
        }
        onMouseEntered = EventHandler { event: MouseEvent ->
            if (!event.isPrimaryButtonDown) {
                scene.cursor = Cursor.HAND
            }
        }
        onMouseExited = EventHandler { event: MouseEvent ->
```

```
            if (!event.isPrimaryButtonDown) {
                scene.cursor = Cursor.DEFAULT
            }
        }
    }

    /**
     * Disables user drag
     */
    fun disableDrag() {
        onMousePressed = EventHandler { event: MouseEvent? -> }
        onMouseReleased = EventHandler { event: MouseEvent? ->
}
        onMouseDragged = EventHandler { event: MouseEvent? -> }
        onMouseEntered = EventHandler { event: MouseEvent? -> }
        onMouseExited = EventHandler { event: MouseEvent? -> }
    }

    fun setLabel(label: Text) {
        name = label
        name.style = "-fx-text-inner-color: red; -fx-font-size:
16px;"
        name.fill = Color.RED

name.xProperty().bind(centerXProperty().subtract(name.layoutBo
unds.centerX))
        name.yProperty().bind(centerYProperty().subtract(
name.layoutBounds.centerY))
    }

    /**
     * Sets value to fit in boundaries: (min + radius; max -
radius)
     *
```

```
     * @param value - current value
     * @param min   - min value
     * @param max   - max value
     * @return bound value
     */
    private fun boundCenterCoordinate(value: Double, min:
Double, max: Double): Double {
        val radius = radius
        return if (value < min + radius) {
            min + radius
        } else if (value > max - radius) {
            max - radius
        } else {
            value
        }
    }
}
```

Название файла: VXEdge.kt

```
package com.fxgraph.vizualization

import javafx.event.EventHandler
import javafx.scene.Node
import javafx.scene.control.TextField
import javafx.scene.input.KeyCode
import javafx.scene.input.KeyEvent
import javafx.scene.input.MouseEvent
import javafx.scene.paint.Color
import javafx.scene.shape.Line
import javafx.scene.text.Text


class VXEdge(var source: VXCell,var target: VXCell,val weight:
Int): Line() {
    private var weigh: Int = weight
```

```
   private var name: Text = Text(weight.toString())
   init{

       this.style = "-fx-stroke-width: 1px"
       name.setFill(Color.BLUE); // setting color of the text
to blue
       name.setStroke(Color.BLUE)
       startXProperty().bind(source.centerXProperty().add(
source.radiusProperty()))
       startYProperty().bind(source.centerYProperty().add(
source.radiusProperty()))
       endXProperty().bind( target.centerXProperty().add(
target.radiusProperty()))
       endYProperty().bind(target.centerYProperty().add(
target.radiusProperty())) // задать при перемещении
   }
   fun getsource(): VXCell { //получить стартовую
       return source;
   }


   fun set_label(label: Text){
       name = label
       name.style = "-fx-text-inner-color: red; -fx-font-size:
16px;"
       name.fill = Color.RED

name.layoutXProperty().bind(startXProperty().add(endXProperty(
)).divide(2).subtract(label.layoutBounds.width / 2))

name.layoutYProperty().bind(startYProperty().add(endYProperty(
)).divide(2).add(label.layoutBounds.height / 1.5));
   }
```

```
    fun get_label(): Text{
        return name
    }

    fun gettarget(): VXCell { // получить конечную
        return target;
    }

    fun getweight(): Int {
        return weight
    }
}
```