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: 00.07.2020

Дата защиты отчета: 00.07.2020

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”.

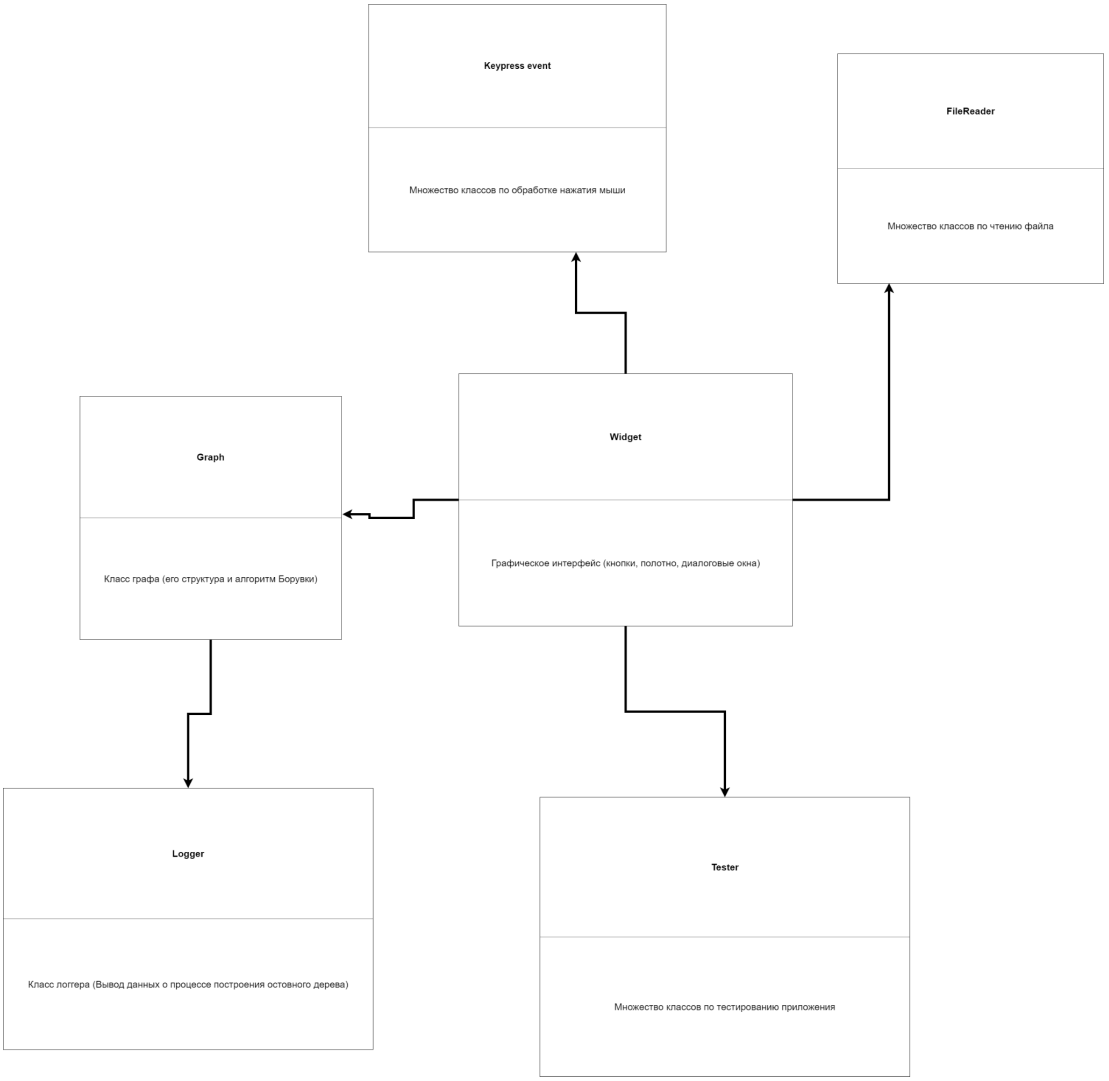
В исходном окне приложения пользователю будет предоставляться выбор для ввода данных: через нажатие кнопки мыши или через файл. Если пользователь выберет ввод данных через файл, то ему необходимо будет нажать соответствующую кнопку после чего откроется окно, в котором можно будет выбрать файл формата *.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:

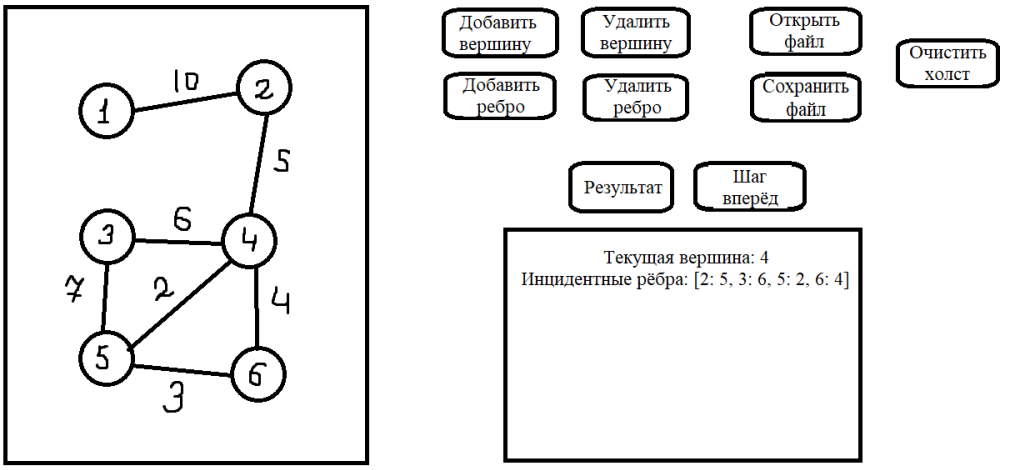
- Вершины выглядят как окружности, внутри которых будут написаны их номера.
- Рёбра представляют собой линии соединяющие вершины, над которыми будет написан их вес.

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 "==========".

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

allEdges: MutableList<Edge> - список всех список всех ребер 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:

***graphParent: Cell* - невидимый корневой узел**

allCells: MutableList<Cell> - список всех вершин графа

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

***removedCells: MutableList<Cell>* - список удаленных вершин**

allEdges: MutableList<Edge> - список всех ребер

***removedEdges: MutableList<Edge>* - список удаленных ребер**

***cellMap: HashMap<String,Cell>* -**

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 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 addAllEdges(temp: Edge) - метод, добавляет в список *allEdges* ребро *temp*

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, выводит в консоль 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

***fun setview(view: Node)* - метод**, **размещающий вершину на холсте**

***fun getview()* - метод**, **возвращающий отображение вершины**

Методы класса ***Model****:*

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

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

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(id: String)* - метод**

***fun addCell(cell: Cell)* - метод**

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

***fun disconnectFromGraphParent(cellList: MutableList<Cell>)* - метод**, **удаляющий вершину от корня**

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

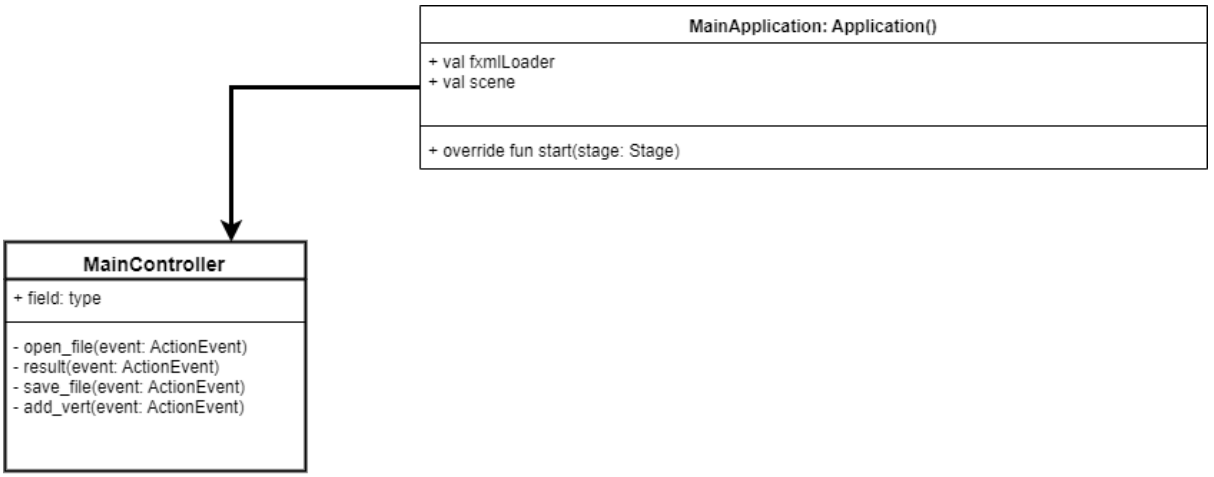
fun checkEdge(temp1: Cell, temp2: Cell, weight: Int) - метод, проверяющий находится ли ребро в списке *allEdges* по двум вершинам и весу.

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

***fun merge()* - метод**

fun BoruvkaMST() - метод, запускающий основной алгоритм(алгоритм Борувки)

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 петли. В этом случае алгоритм не включит их 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. Второй подраздел третьего раздела

ЗАКЛЮЧЕНИЕ

Кратко подвести итоги, проанализировать соответствие поставленной цели и полученного результата.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Ниже представлены примеры библиографического описания, В КАЧЕСТВЕ НАЗВАНИЯ ИСТОЧНИКА в примерах приводится вариант, в котором применяется то или иное библиографическое описание.

1. Иванов И. И. Книга одного-трех авторов. М.: Издательство, 2010. 000 с.

2. Книга четырех авторов / И. И. Иванов, П. П. Петров, С. С. Сидоров, В. В. Васильев. СПб.: Издательство, 2010. 000 с.

3. Книга пяти и более авторов / И. И. Иванов, П. П. Петров, С. С. Сидоров и др.. СПб.: Издательство, 2010. 000 с.

4. Описание книги под редакцией / под ред. И.И. Иванова СПб., Издательство, 2010. 000 с.

5. Иванов И.И. Описание учебного пособия и текста лекций: учеб. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2010. 000 с.

6. Описание методических указаний / сост.: И.И. Иванов, П.П. Петров. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2010. 000 с.

7. Иванов И.И. Описание статьи с одним-тремя авторами из журнала // Название журнала. 2010, вып. (№) 00. С. 000–000.

8. Описание статьи с четырьмя и более авторами из журнала / И. И. Иванов, П. П. Петров, С. С. Сидоров и др. // Название журнала. 2010, вып. (№) 00. С. 000–000.

9. Иванов И.И. Описание тезисов доклада с одним-тремя авторами / Название конференции: тез. докл. III международной науч.-техн. конф., СПб, 00–00 янв. 2000 г. / СПбГЭТУ «ЛЭТИ», СПБ, 2010, С. 000–000.

10. Описание тезисов доклада с четырьмя и более авторами / И. И. Иванов, П. П. Петров, С. С. Сидоров и др. // Название конференции: тез.

докл. III международной науч.-техн. конф., СПб, 00–00 янв. 2000 г. / СПбГЭТУ «ЛЭТИ», СПБ, 2010, С. 000–000.

11. Описание электронного ресурса // Наименование сайта. URL: http://east-front.narod.ru/memo/latchford.htm (дата обращения: 00.00.2010).

12. ГОСТ 0.0–00. Описание стандартов. М.: Изд-во стандартов, 2010.

13. Пат. RU 00000000. Описание патентных документов / И. И. Иванов, П. П. Петров, С. С. Сидоров. Опубл. 00.00.2010. Бюл. № 00.

14. Иванов И.И. Описание авторефератов диссертаций: автореф. дисс. канд. техн. наук / СПбГЭТУ «ЛЭТИ», СПБ, 2010.

15. Описание федерального закона: Федер. закон [принят Гос. Думой 00.00.2010] // Собрание законодательств РФ. 2010. № 00. Ст. 00. С. 000–000.

16. Описание федерального постановления: постановление Правительства Рос. Федерации от 00.00.2010 № 00000 // Опубликовавшее издание. 2010. № 0. С. 000–000.

17. Описание указа: указ Президента РФ от 00.00.2010 № 00 // Опубликовавшее издание. 2010. № 0. С. 000–000.

ПРИЛОЖЕНИЕ А

НАЗВАНИЕ ПРИЛОЖЕНИЯ

полный код программы должен быть в приложении, печатать его не надо