

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

ОТЧЕТ
по учебной практике
Тема: Алгоритм Краскала

Студентка гр. 1303	_____	Куклина Ю.Н.
Студентка гр. 1303	_____	Сырцева Д.Д.
Студентка гр. 1303	_____	Хабибуллина А.М.
Руководитель	_____	Токарев А.П.

Санкт-Петербург
2023

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студентка Куклина Ю.Н. группы 1303

Студентка Сырцева Д.Д. группы 1303

Студентка Хабибуллина А.М. группы 1303

Тема практики: Алгоритм Краскала

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Kotlin с графическим интерфейсом.

Алгоритм: Краскала

Сроки прохождения практики: 30.06.2023 – 13.07.2023

Дата сдачи отчета: 10.07.2023

Дата защиты отчета: 10.07.2023

Студентка	_____	Куклина Ю.Н.
Студентка	_____	Сырцева Д.Д.
Студентка	_____	Хабибуллина А.М.
Руководитель	_____	Токарев А.П.

АННОТАЦИЯ

Данная работа направлена на создание приложения с графическим интерфейсом, где реализован и визуализирован алгоритм Краскала – поиска минимального остовного дерева.

Необходимо создать возможности различного задания входных данных, взаимодействия пользователя с приложением, возможность пошагового прохождения алгоритма.

В качестве итога получаем приложение с удобным для пользователя интерфейсом.

SUMMARY

This work is aimed at creating an application with a graphical interface, where the Kraskal algorithm is implemented and visualized – the search for the minimum spanning tree.

It is necessary to create opportunities for various input data assignments, user interaction with the application, the possibility of step-by-step passing of the algorithm.

As a result, we get an application with a user-friendly interface.

СОДЕРЖАНИЕ

Введение	5
1. Требования к программе	6
1.1. Исходные требования к программе*	6
1.2. Уточнение требований после сдачи прототипа	8
1.3. Уточнение требований после сдачи первой версии	11
2. План разработки и распределение ролей в бригаде	13
2.1. План разработки	13
2.2. Распределение ролей в бригаде	13
3. Особенности реализации	14
3.1. Общее описание	14
3.2. Описание класса Graph	14
3.3. Алгоритм Краскала	16
3.4. Описание класса GraphViewState	17
3.5. Верстка	21
3.6. Работа с файловой системой	24
4. Тестирование	26
4.1. Тестирование класса графа и алгоритма Краскала	26
4.2. Тестирование UI	26
Заключение	27
Список использованных источников	28
Приложение А. Исходный код	29
Приложение Б. Тестирование	49

ВВЕДЕНИЕ

Цель работы заключается в разработке приложения, которое будет визуализировать работу алгоритма Краскала – поиска минимального остовного дерева. А также создание удобного интерфейса для взаимодействия с пользователем.

Задачи:

- Разработка алгоритма Краскала на языке Kotlin
- Возможность задания входных данных разными способами
- Пошаговое отображение алгоритма

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

Наше приложение по реализации алгоритма Краскала обладает следующими возможностями.

Способы задания входных данных:

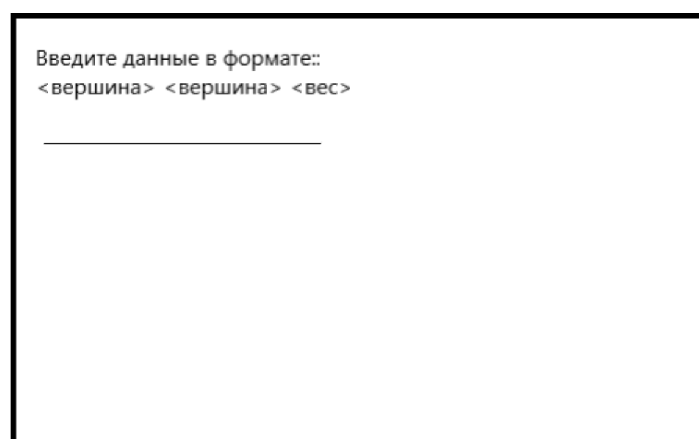
- Из файла
- Ввод с клавиатуры
- Вручную на холсте
- Смешанно (считать из файла/вручную и модифицировать на холсте)



Выберите способ задания входных данных

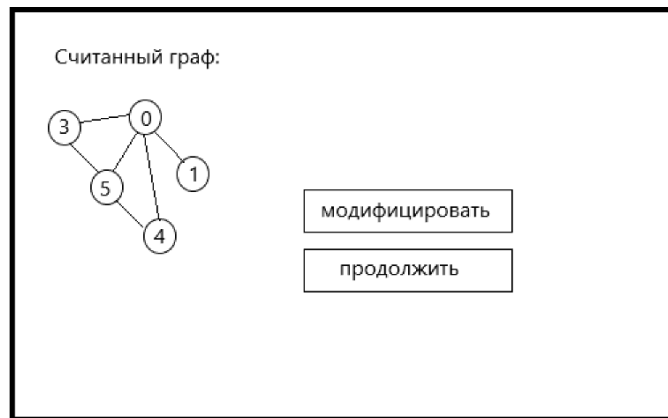
Из файла С клавиатуры Графически

При выборе ввода с клавиатуры будет открыта страница:



Введите данные в формате:
<вершина> <вершина> <вес>

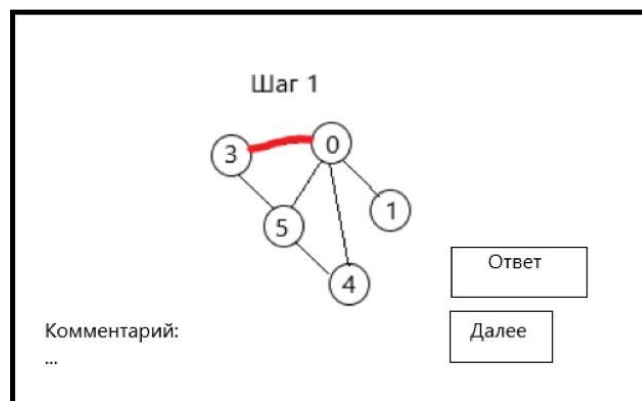
При выборе ввода из файла или после ручного ввода будет предложено модифицировать графически:



Визуализация работы алгоритма:

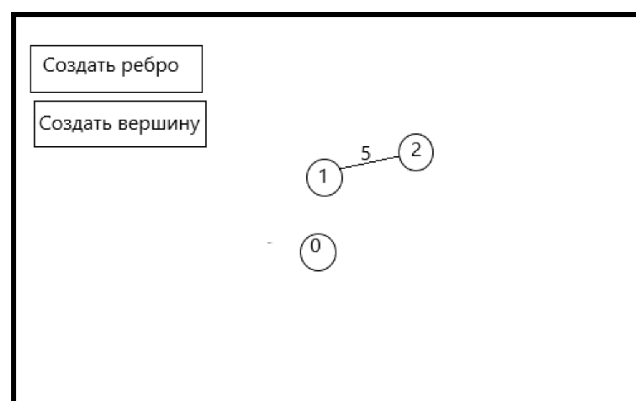
Управление будет осуществляться при помощи мыши, а именно нажатием на клавишу “Далее”, осуществляющую новый шаг алгоритма, либо нажатие кнопки с выводом конечного результата “Ответ”.

Также выбранное ребро закрашивается и выводится текстовый комментарий:

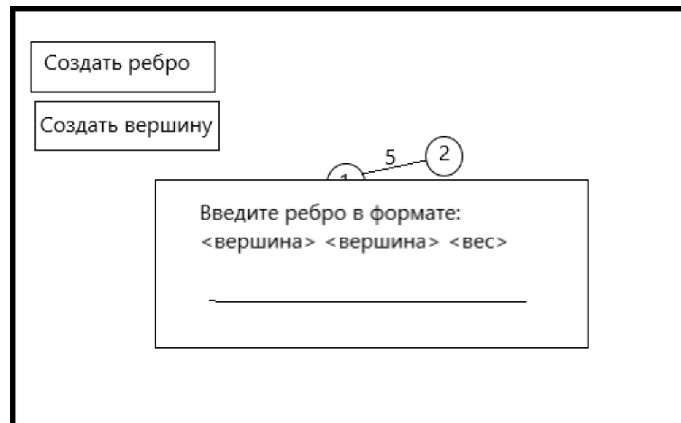


Создание графа графически:

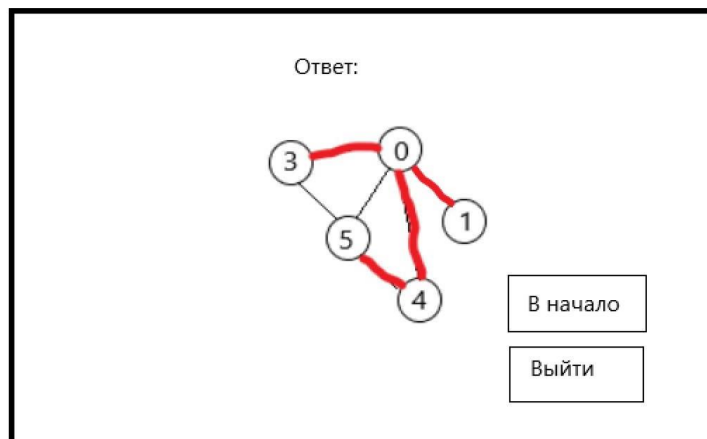
При создании вершины они появляются с номерами по очереди:



Если добавляется ребро, то в появившемся окошке необходимо вписать данные:



В итоге выведется ответ с вариантом прогнать алгоритм еще раз или выйти из приложения:



1.2. Уточнение требований после сдачи прототипа

Приложение должно удовлетворять следующим функциональным требованиям:

Способы задания входных данных:

- Из файла

(Данные в файле должны быть представлены в виде:

<вершина1> <вершина2> <вес ребра>)

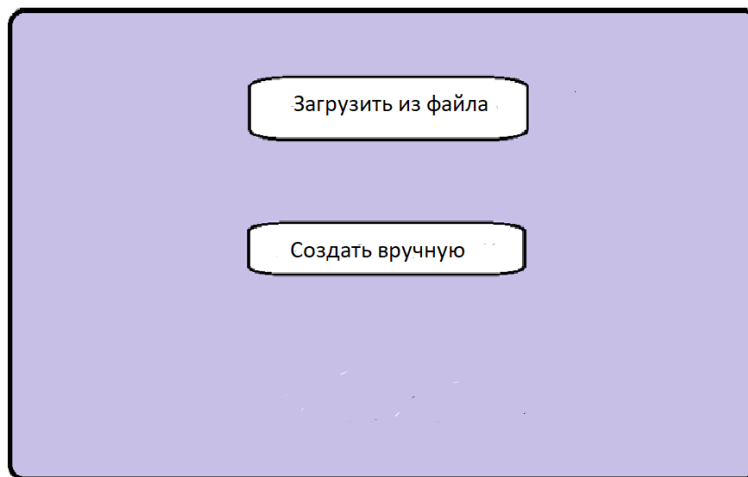
- Вручную на холсте

(Пользователь создает граф графически, добавляя визуализированные вершины и ребра)

- Смешанно (считать из файла и модифицировать на холсте)

После считывания данных из файла, пользователь имеет возможность модифицировать граф вручную, добавляя новые вершины и ребра (ребра будут создаваться при соединении двух вершин))

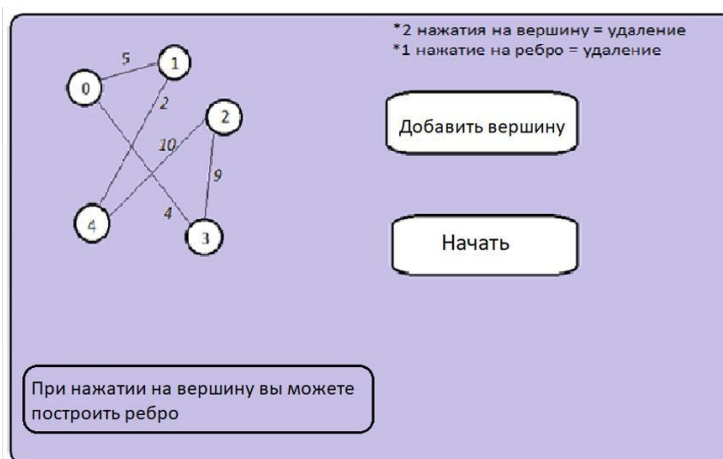
Вводная страница предлагает на выбор 2 варианта создания графа:



При выборе ввода из файла, содержимое файла должно соответствовать виду: <вершина1> <вершина2> <вес ребра>.

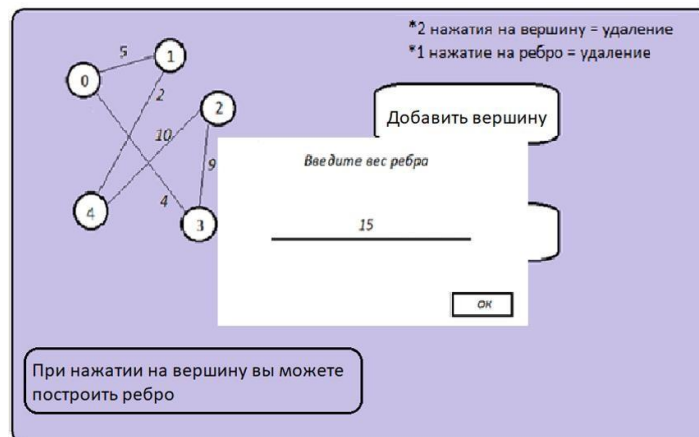
Также при выборе ввода данных из файла будет открываться диалоговое файловое окно, таким образом будет дана возможность выбрать нужный файл.

После выбора файла будет появляться считанный граф, который будет возможно модифицировать при желании.



Если есть необходимость в модификации, то нужно нажать на «Добавить вершину». Тогда появится отдельно стоящая вершина, которую можно перемещать. Чтобы добавить ребро, нужно нажать на вершину – тогда появится ребро, которое можно довести до любой другой вершины, тем самым построить

ребро между двумя данными вершинами, после чего откроется окно, в которое можно вписать вес построенного ребра.



После модификации, нажав на кнопку «Начать», начнется выполнение алгоритма.

Создание графа вручную будет происходить аналогично тому, как было описано выше. То есть будет возможность добавлять вершины, а при их соединении, будут появляться ребра. При создании, вершины появляются с номерами по очереди.

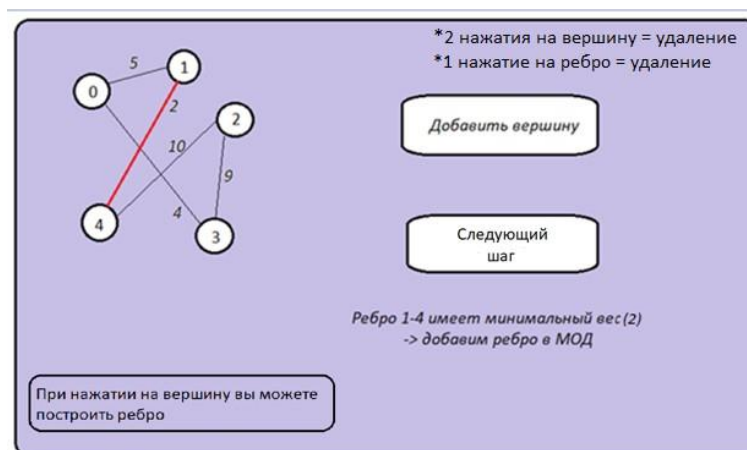
Для того, чтобы удалить ребро, необходимо нажать на него, а для удаления вершины, нажать на нее дважды. При удалении вершины, удаляются все инцидентные ей ребра. Для этого будет написана подсказка – «*2 нажатия на вершину = удаление,

*1 нажатие на ребро = удаление»

Визуализация работы алгоритма:

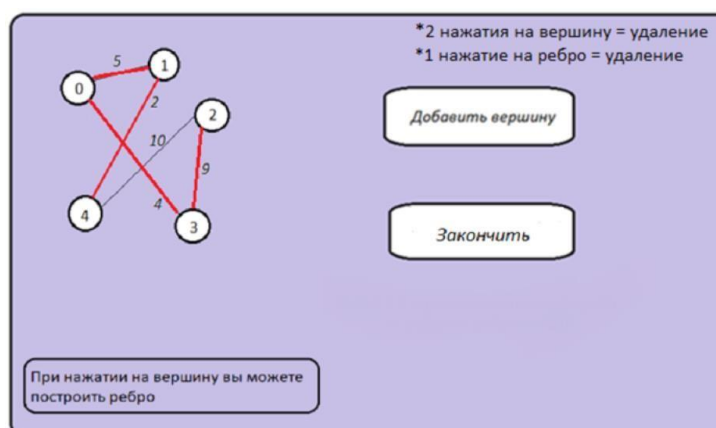
На холсте будет представлено по одному шагу алгоритма. Управление осуществляется при помощи мыши, а именно нажатием на клавишу “Следующий шаг”, переносящую на новый шаг алгоритма.

Выбранное ребро, внесенное в минимальное остовное дерево, закрашивается, и выводится текстовый комментарий, описывающий итерацию алгоритма:



На протяжении работы алгоритма можно будет модифицировать граф, но это прервет работу алгоритма, однако после внесенных изменений его можно будет начать заново. Если граф при построении оказывается не связанным, то невозможно продолжить работу алгоритма, пока не будет построено ребро (кнопка не будет активна).

После прохождения всех шагов будет представлен ответ – граф, в котором закрашенные ребра представляют МОД.



После нажатия кнопки «Закончить» раскраска сбросится, будет открыта и исходная страница (рис1) и можно будет задать другие входные данные и прогнать алгоритм заново.

1.3 Уточнение требований после сдачи первой версии.

- Дополнительно к вышеперечисленным требованиям в приложении должна быть возможность переходить на шаг назад при выполнении алгоритма.
- Также должна быть кнопка возврата на начальный экран, где можно будет

снова выбрать «Загрузить из файла» или «Создать вручную».

- Веса на ребрах должны быть хорошо читаемы.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

- 1) Написание спецификации и плана разработки. Исполнитель: Куклина Ю.Н., Сырцева Д.Д. Срок: 03.07
- 2) Написание алгоритма Краскала. Исполнитель: Хабибуллина А.М. Срок: 01.07
- 3) Написание класса графа. Исполнитель: Сырцева Д.Д. Срок: 01.07
- 4) Верстка стартового экрана. Исполнитель: Куклина Ю.Н. Срок: 04.07
- 5) Верстка экрана редактирования. Исполнитель: Куклина Ю.Н. Срок: 04.07
 - Диалоговое окно. Исполнитель: Хабибуллина А.М. Срок: 04.07
- 6) Взаимодействие с файловой системой. Исполнитель: Сырцева Д.Д. Срок: 04.07
- 7) Верстка графа. Исполнитель: Куклина Ю.Н. Срок: 07.07
- 8) Логика работы с UI. Исполнитель: Сырцева Д.Д. Хабибуллина А.М. Срок: 07.07
 - Изменение графа. Исполнитель: Сырцева Д.Д.
 - Отслеживание состояний, алгоритм. Исполнитель: Хабибуллина А.М.
- 9) Тестирование
 - Тестирование класса графа и алгоритма Краскала. Исполнитель: Хабибуллина А.М. Срок: 10.07
 - Тестирование UI. Исполнитель: Сырцева Д.Д., Куклина Ю.Н. Срок: 10.07

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Общее описание.

Проект был разработан с использованием Jetpack Compose для визуального интерфейса приложения. Compose – это декларативный и удобный в использовании фреймворк, предоставляющий инструменты для создания UI.

Для представления графа был создан класс Graph, который содержит основные методы для его создания и изменения.

Для визуализации графа реализованы Composable функции, представленные в файле GraphView.

Сам граф не знает, как его отображают, поэтому логика взаимодействия графики и объектов, которые нужно отобразить, реализована в GraphViewState.

Для запоминания и отслеживания состояний объекта используются функции remember и mutableStateOf. Remember позволяет создать объект единожды, запоминая его текущее состояние, а mutableStateOf создает объект MutableState, который позволяет отслеживать изменения хранимого в нем значения. Любые изменения этого значения приведут к обновлению любого компонента, который использует данное значение.

Теперь подробнее опишем каждую составляющую.

3.2 Описание класса Graph.

Для представления графа написан класс Graph.

Сам граф представим в виде матрицы весов (поле matrix). Переменная является изменяемым списком (MutableList), так как объект по мере выполнения программы изменяется. Размер матрицы указан в поле size, и вычисляется каждый раз, когда мы обращаемся к данному полю.

Функции класса Graph:

`addVertex()` – добавление вершины в граф. Чтобы было проще понять, как работает данная функция, можно представить квадратную матрицу. При

добавлении в нее значений для одной вершины, нужно добавить в нее столбец и строку, которые будут состоять из нулей (веса ребер=0). Прохождение по матрице осуществляется благодаря ListIterator, также он позволяет использовать функции add и hasNext () для того, чтобы заполнить нужные позиции.

`removeVertex()` – удаление вершины из графа. Снова представим квадратную матрицу весов. При удалении из нее вершины, нужно удалить соответствующий столбец и строку. Итератор также предоставляет функцию `remove ()`, которая и производит удаление значений.

`toString (): String` – переопределенная функция, для вывода матрицы в консоль. Чтобы вывод был ясным, чтобы представить число с определенной шириной и заполнителем (в нашем случае - нули), используется функция `format` и оператор форматирования `%`.

`set (src: Int, dest: Int, weight: Int)` – оператор `set`, позволяющий в матрице симметрично расставить веса.

`get (src: Int, dest: Int)` – оператор `get`, позволяющий получить значение из матрицы исходя из значений поданных вершинам.

`replace (graph: Graph)` – функция, позволяющая заменить существующий граф на поданный.

`edges (): Sequence<Edge>` - функция, возвращающая последовательность ребер. Осуществляется проход по графу и поочередно создаются ребра - объекты класса `Edge` (если вес больше нуля), которые впоследствии и возвращаются.

`isConnected ()` – функция, проверяющая то, связный ли граф в настоящий момент. Для этого вызывается функция – `kruskal ()`. Если выбросится исключение, то вернется `false` – граф несвязный, в противном случае `true`- граф связный.

3.3 Алгоритм Краскала.

Также класс `Graph` содержит функцию `kruskal()`, осуществляющую сам алгоритм Краскала. Для удобства все, что связано с алгоритмом вынесено в отдельный файл.

Для хранения ребер создан класс `Edge`, содержащий в себе три поля: вершина 1, вершина 2 и вес ребра, соединяющего данные вершины.

Теперь опишем функцию `fun Graph.kruskal()`. Данная функция реализовывает алгоритм Краскала и возвращает последовательность ребер, вошедших в минимальное остовное дерево. В переменную `edges` записываем список ребер исходного графа. Далее проверяем, есть ли ребра в нашем графе, если нет, то выбрасываем исключение. Сортируем ребра исходного графа по весу и создаем массив вершин, где номер вершины равен ее индексу, то есть каждая вершина сама себе родитель.

Теперь циклом `while` проходимся, пока все вершины не войдут в каркас. Далее определяется, какой компоненте принадлежат 2 вершины (концы ребра), и если компоненты не совпадают, то ребро добавляется в минимальное остовное дерево, а сами вершины объединяются в одно подмножество. Это необходимо, чтобы избежать циклов в МОД. При этом, если обнаруживается, что ребер не хватает для построения минимального остовного дерева, то выбрасывается исключение, так как это значит, что граф оказался несвязным.

Для определения родителя вершины (родитель определяет компоненту), написана функция `private fun find(subsets: IntArray, i: Int)`. Функция работает рекурсивно, пока индекс и значение не будут совпадать. Найденное значение возвращается и определяет родителя.

Для объединения вершин в одно подмножество, написана функция `private fun union(subsets: IntArray, x: Int, y: Int)`. Она присваивает одной вершине родителя другой вершины, тем самым объединяя их в одно подмножество.

3.4 GraphViewState.

Для реализации логики программы и обеспечения корректного взаимодействия объектов и их отображения был написан класс `GraphViewState`. Созданы вспомогательные переменные, которые будут использованы далее в функциях.

`const val vertexRadius = 0.1f` – переменная для масштабирования вершины (`f` – float)

`const val strokeWidth = 0.01f` – переменная для масштабирования ребра.

`val BoxWithConstraintsScope.minDimension get() = min(maxWidth, maxHeight)` – свойство только для чтения, которое определено в области `BoxWithConstraintsScope`, так как нужно использовать `maxWidth` и `maxHeight`(максимально доступная ширина и высота контейнера).

`val BoxWithConstraintsScope.vertexSize get() = minDimension * vertexRadius` – размер вершины, который рассчитывается каждый раз при обращении к данной переменной.

`val BoxWithConstraintsScope.vertexSizeHalf get() = vertexSize / 2` – половина размера вершины.

В самом классе `GraphViewState` объявлены следующие поля(переменные):

`private val angle: Double get() = 2 * Math.PI / graph.size` – угол, под которым будут расположены вершины(когда считывание из `graph.size`, чтобы расположить вершины графа как вершины правильного многоугольника)

`var coords by mutableStateOf(listOf<Offset>())` – координаты вершин.
(`by` - автоматизирует процесс `get()`, как бы подставляет `.value` к `coords`)

`var isConnected by mutableStateOf(false)` – связность графа

`var selectedVertex by mutableStateOf(-1)` – индекс выбранной вершины

`var edges by mutableStateOf(listOf<Edge>())` – список ребер графа

`var openDialog by mutableStateOf<DialogState?>(null) - nullable`
переменная, определяющая, открыто ли диалоговое окно. Если не null, то является объектом класса DialogState.

`var screen by mutableStateOf(Screen.Start) - Screen` – enum класс, содержащий перечисления: Start и Graph – какой экран открыт (начальный или основной)

`private var steps: ListIterator<Edge>? = null` – steps – итератор по ребрам в МОД, может быть null.

`set(value) { - установим значение`

`field = value - field` - реальное значение steps

`coloredEdges = listOf()` – обнуление всего списка закрасенных ребер

`}`

`var curWeight by mutableStateOf(0)` – текущий вес МОД

`var coloredEdges by mutableStateOf(listOf<Edge>()) { - - грани, которые`
нужно окрасить(уже точно вошли в каркас), при обновлении значения вызов функции – обновления состояний программы.

`onStepsUpdated()`

`}`

`var hasNextStep by mutableStateOf(false)` – есть ли следующий шаг
алгоритма

`var hasPreviousStep by mutableStateOf(false)` – есть ли предыдущий
шаг алгоритма

`var isActive by mutableStateOf(false)` –начался ли алгоритм

Также есть инициализирующий блок `init`, который вызывает функцию `onGraphUpdate()`.

Функции данного класса:

`onGraphUpdate()` – обновляет граф. Обновление значений `isConnected`, `coords`, `edges`, «очистка» МОД, вывод матрицы в консоль и обновление

состояния программы.

`replaceGraph(graph: Graph)` - так называемая «замена» графа. То есть обнуление построенного МОД, изменение матрицы, очистка координат и вызов функции `onGraphUpdate()`.

`complete()` – обнуление МОД, `steps` становится `null`, также текущий вес МОД становится равен 0

`resetCoords(radius: Float=0.5f)` – функция, принимающая на вход радиус, чтобы вершины были расположены как бы на правильном многоугольнике, который вписан в окружность соответствующего радиуса. Заполняет список `coords`.

`addVertex()` – функция, вызываемая при нажатии на кнопку «Добавить вершину». Изменяет матрицу весов и список координат. Если какие-то координаты уже есть в списке, то новую помещаем в центре относительно остальных. Если же список пуст, то вызовем `resetCoords()`.

`vertex(index: Int)` – функция, возвращающая координаты вершины по ее индексу.

`connect(src: Int, dest: Int, value: Int)`- при соединении двух вершин, запишем в матрицу соответствующий вес и обновим граф(`onGraphUpdate()`).

`removeEdge(i: Int, j: Int)` – при нажатии на ребро вызывается данная функция, которая изменяет матрицу(ставит на пересечении ноль) и обновляет граф.

`removeVertex(index: Int)` – удаление вершины, при двойном нажатии на нее. Для этого нужно обновить значения в матрице, то есть вызвать функцию `removeVertex(index)` у графа. Далее обновить координаты добавив в список все, кроме соответствующего индекса (с помощью `filterIndexed`) и обновить граф

Функция `private fun onStepsUpdated()` изменяет состояния программы. Переменной `hasNextStep` присваивается `true`, если значение `steps` не `null` и если функция итератора `hasNext()` вернет не `null`, иначе `false`. Переменной `has`

Previous присвоится true, если значение steps не null и функция hasPrevious() вернет true, иначе false. Переменной isActive присваивается true, если steps не равно null, иначе false. Изначально все переменные равны false. Данные переменные определяют какие кнопки будут отображены на экране приложения.

При нажатии на вершину вызывается функция `vertexClicked(index: Int)`. Функция присваивает `selectedVertex` индекс нажатой вершины, если `selectedVertex = -1`. Иначе, если значение `selectedVertex` равно нажатой вершине, то вызывается функция `removeVertex(index)`, удаляющая вершину, иначе вызывается функция `connect(selectedVertex, index, 1)` для построения ребра между двумя вершинами и изменяется значение переменной `openDialog`.

То есть, если кликнуть на вершину два раза она удалится, а если сначала кликнуть на одну, а потом на другую, то между данными вершинами построится ребро.

Для перемещения вершин написана функция `fun drag(selected: Int, dragOffset: Offset)`. Функция высчитывает координаты на которые перемещена выбранная вершина (исходные координаты + смещение относительно этих координат).

Также есть функция `fun cancelSelection()`, которая отменяет выбор вершины, присваивая полю `selectedVertex` значение -1.

Функция `fun closeDialog()` присваивает полю `openDialog` значение null.

Функция `fun openGraphScreen()` присваивает полю `screen` значение `Graph` из enum класса `Screen`.

Функция `fun openStartScreen()` присваивает полю `screen` значение `Start` из enum класса `Screen` и вызывает функцию `clear()`

Функция `clear()` позволяет очистить матрицу, обнулить МОД, очистить координаты и обновить граф.

`class DialogState(val src: Int, val dest: Int)` - класс содержащий поля вершина 1 и вершина 2. С его помощью определяется для каких двух

вершин будет записан вес ребра при открытии диалогового окна.

Функция `fun Offset.coerced()` делает координаты `Offset` относительными.

При нажатии на кнопку «Начать» вызывается функция `StartKruskal()`, а следом за ней функция `next()` чтобы сразу же отобразить первый шаг алгоритма. Далее опишем концепцию работы и зависимость данных функций.

Вызывается функция `StartKruskal()`, в которой вызывается функция `kruskal()`, возвращаемое значение которой преобразуется в список, с помощью функции `toList()`, так как функция «терминальная», то все вызовы функции `yield()` в функции `kruskal()` сразу отработают, в результате получим список, который преобразуется в итератор.

Функция `next()` вызывается каждый раз при нажатии на кнопку «Далее». Данная функция проверяет, есть ли возможность перехода к следующей итерации при покраске ребер, и если да, то в список `coloredEdges` добавляется очередное ребро, также к весу МОД прибавляется соответствующее значение.

Функция `previous()` противоположна `next()`. Она проверяет, можно ли сделать шаг назад по итерациям в покраске МОД, если да, то из списка `coloredEdges` удаляет ребро и из общего веса МОД вычитает соответствующее значение.

3.5 Верстка.

При открытии приложения появляется стартовое окно, которое содержит 2 кнопки «Загрузить из файла» и «Создать вручную», нажатие каждой из которых переводит на основной экран, где можно запустить и отследить алгоритм.

Логика работы стартового экрана описана в функции `StartScreen()`.

Экран реализован при помощи контейнера `Column` для размещения элементов друг под другом.

При нажатии кнопки «Загрузить из файла» откроется файловое окно для выбора файла, содержащего граф. Нажатие кнопки «Создать вручную»

перенесет сразу на основной экран.

Основной экран приложения реализован в функции `GraphScreen()`. Экран представляет собой строку – `Row` – размещает элементы горизонтально. Сама «строка» делится на 2 части, одна из которых отвечает за отображение графа – `GraphView`, а другая за отображение управления изменением графа – `ControlPanel`.

`ControlPanel` реализован в контейнере `Column`. В нем содержится `Box`, в котором размещена подсказка для действий над графом. Кнопка «Добавить вершину», при нажатии которой вызывается функция `addVertex()` – данную логику обеспечивает поле `onClick`. Следующая кнопка меняется в зависимости от статуса работы алгоритма. Если алгоритм в работе – то отображается кнопка «Дальше», если пройдены все шаги, то «Закончить», иначе алгоритм не начат – кнопка «Начать». Также имеется кнопка «Назад», которая переносит на один шаг назад при выполнении алгоритма.

Если алгоритм в работе, то на экране появляется элемент `LazyColumn`, который обеспечивает «ленивую» загрузку элементов, то есть загрузку по мере их создания.

`GraphView` является контейнером `BoxWithConstraints` для того, чтобы иметь доступ к ограничениям контейнера – `maxWidth`, `maxHeight`. Также в него мы устанавливаем `onPointerEvent`, чтобы отслеживать позицию указателя, а именно мыши, так как здесь для рисовки ребра и перемещения вершины, это необходимая составляющая. `OnClick` обрабатывает ситуацию нажатия на `Box` вне его дочерних элементов, тогда отменится нажатие на какую-либо вершину.

В этом боксе содержатся функции рисования ребра, вершины и рисования неполного ребра.

Функция `EdgeView()` рассчитывает все необходимые значения для рисовки ребра. Вычисляет толщину ребра, начальная и конечная координата, цвет и надпись на ребре и, используя эти данные, вызывает функцию `EdgeLine()`.

Функция `EdgeLine()` по полученным данным рисует ребро, которое

представляется контейнером `Box`. Дополнительно здесь рассчитывается угол поворота ребра относительно горизонтальной оси и длина ребра.

`EdgeLine()` принимает также `modifier`, который мы можем передать или не передать. В случае, когда рисуется полное ребро от одной вершины до другой, то мы передаем `Modifier.onClick {state.removeEdge(src, dest)}`, чтобы при нажатии на ребро оно удалялось. Но в случае, когда мы «тащим ребро», то требуется просто нарисовать его от исходной вершины до текущей позиции мыши и функция удаления не требуется.

Функция `VertexView()` рисует вершину, как контейнер `Box`. Контейнеру задается размер, положение, `drawBehind` позволяет нарисовать на боксе круг заданного цвета. `OnClick` здесь обеспечивает то, что при нажатии на вершину она становится выбранной. `OnDrag` отвечает за обработку событий перетаскивания элемента, выполняя нужную логику, в данном случае мы меняем ее координаты.

Функция `SelectLine()` позволяет нарисовать ребро от выбранной вершины. Данная функция вызывается тогда, когда на было разовое нажатие на вершину. Выполняет рисовка ребра от вершины до текущей позиции мыши.

Функция `WeightDialog()` срабатывает тогда, когда нужно указать вес построенного ребра. Используется конструкция `AlertDialog`, чтобы при необходимости всплывало окно, прерывающее пользователя необходимым действием – в данном случае требуется ввести вес ребра. `onDismissRequest` отвечает за то, что если нажатие произошло мимо диалогового окна, то вызовется функция закрытия диалога.

Далее описывается разметка внутри диалогового окна. Задается переменная `focusRequester` – экземпляр `FocusRequester`, который потребуется для запроса фокуса для текстового поля. `LaunchedEffect` применяем, чтобы автоматически запустить запрос фокуса для текстового поля – то есть, чтобы не нужно было нажимать на него для получения возможности вписать вес.

Так же реализована проверка введенного текста на ошибки, а именно проверка на то, что строка не состоит из одних пробелов, не является

отрицательным числом или буквой.

Окно реализовано при помощи контейнера `Column`. Текстовое поле получает какое-то значение – текст, также устанавливается возможность вписывать данные только в одну линию. Далее задана возможность установить вес и выйти из окна нажатием `Enter`.

Если обнаружено, что введенное значение не корректно, то это будет обозначено оранжевым цветом и выводом сообщения «Введите число».

При нажатии кнопки «Подтвердить» введенный вес установится на построенное ребро.

3.6 Работа с файловой системой.

В функции `StartScreen()` создается отслеживаемая булевая переменная `showFilePicker`, которая по умолчанию равна `false`. Данная переменная отвечает за открытие диалогового окна.

При нажатии на кнопку «Загрузить из файла» переменной присваивается `true`. Поскольку переменная `mutableState`, то каждый раз когда она изменяется выполняется `if`, который проверяет равно ли значение переменной `true`. Если да, то параллельно программе с помощью встроенных функций открывается диалоговое окно с названием "Select File to Open". Диалоговое окно открывается в режиме загрузки. Пользователь имеет возможность выбрать любой файл, который находится на его компьютере.

Директория и имя файла записываются в переменные, однако если они `null`, то диалоговое окно запустится повторно для выбора файла. Если директория и имя файла записались успешно, то вызываем функцию `fromFile()`, в которую передаем путь к файлу, предварительно преобразованный в строку.

Если не удалось корректно считать данные из файла, то диалоговое окно откроется снова. При успешном считывании выполняется функции `replaceGraph(graph)` и `openGraphScreen()`, которая переводит приложение на основной экран.

При нажатии на кнопку “Загрузить из файла” откроется окно и будет предоставлен выбор для загрузки конкретного файла. После чего произойдет вызов функции `fromFile(path: String)` - блок `try-catch`, используемый для отлавливания ошибок при считывании данных. `Catch` обрабатывает все исключения, которые являются подклассами `Extention`. Если возникло исключение, то вернется `null`.

В блоке `try` происходит создание объекта `Graph`, после чего благодаря `apply`, к объекту применяются следующие действия: файл с именем `path` открывается и все строки, что записаны в данном файле преобразуются в поток.

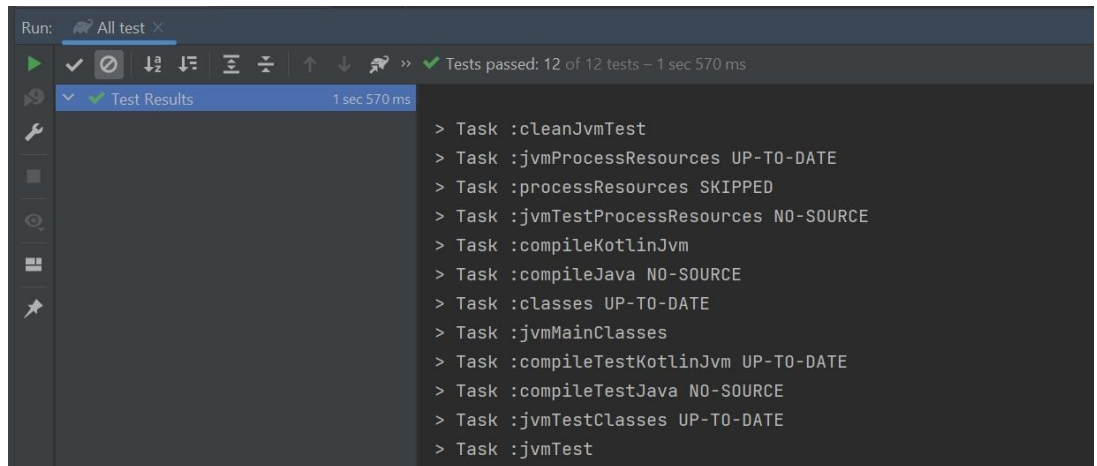
Далее вызывается функция, относящаяся к классу `Graph` – `fillGraph()`, для занесения в матрицу соответствующих значений. Функция `fillGraph()` принимает на вход созданный поток, создает список троек-чисел, которые были записаны в файле. Если в файле были записаны некорректные данные, то не получится привести их к `int` и произойдет исключение, которое будет отловлено. Если же список корректно создался, то на его основе создастся новый список, который будет содержать уже двойки чисел, то есть индексы вершин. Найдем максимальный из индексов и прибавим единицу – таким образом получим количество вершин в графе. В конечном итоге заполним граф на основании первого созданного списка. Выведем граф в консоль.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование функций класса графа и алгоритма Краскала

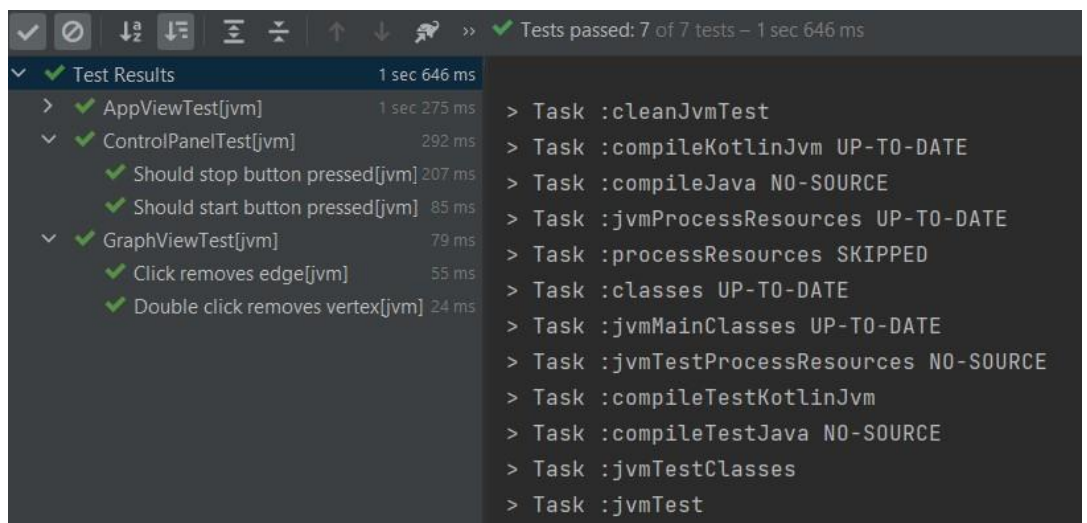
Бали написаны JUnit тесты для проверки корректности добавления, удаления вершин из графа.

Также протестирован алгоритм Краскала. Рассмотрены ситуации, когда граф пуст или не связный.



4.2. Тестирование UI

Тестами покрыты все основные кейсы. То есть проверяется навигация, кликабельность основных элементов графа, изменение кнопок в зависимости от состояния программы.



Код тестирования представлен в Приложении Б.

ЗАКЛЮЧЕНИЕ

В итоге, в качестве проекта на ЯП Kotlin было реализовано приложение выполняющее алгоритм Краскала. Был освоен Jetpack Compose, который отвечает за визуальный интерфейс приложения. В качестве заявленных требований к программе создано два экрана (начальный и основной) с различными кнопками, осуществляющими работу с приложением. Реализовано считывание графа из файла, рисование его вручную - удаление и добавление вершин и ребер. Каждый шаг алгоритма представлен отдельно, переход по итерациям осуществляется с помощью кнопки «Далее», также при нажатии кнопки «Назад» имеется возможность вернуться на один шаг назад при выполнении алгоритма.

Таким образом, все заявленные требования к проекту были выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Визуальные компоненты. Работа с Text. Название сайта: METANIT.COM. URL: <https://metanit.com/kotlin/jetpack/4.1.php>
2. Введение в Jetpack Compose. Философия построения интерфейса. Название сайта: Хабр. URL: <https://habr.com/ru/companies/rncb/articles/669374/>
3. Введение в состояние компонентов.MutableState и remember. Название сайта: METANIT.COM. URL: <https://metanit.com/kotlin/jetpack/5.2.php>
4. Комбинация remember + mutableStateOf. Название сайта: Startandroid. URL:<https://startandroid.ru/ru/courses/compose/30-course/compose/673-urok-10-remember-mutablestateof.html>
5. Функции области видимости (Scope Functions). Название сайта: Bimlibik. URL: <https://bimlibik.github.io/posts/kotlin-scope-functions/>
6. Диалоговые окна. Название сайта: METANIT.COM. URL: <https://metanit.com/kotlin/jetpack/4.16.php>
7. Как работает yield. Название сайта: Хабр. URL: <https://habr.com/ru/articles/132554/>
8. Мультиплатформенный виджет compose для выбора файлов. Название сайта: Android Example 365. URL: <https://androidexample365.com/a-multiplatform-compose-widget-for-picking-files/>
9. Кнопка Button. Название сайта: METANIT.COM. URL: <https://metanit.com/kotlin/jetpack/4.2.php?ysclid=ljug0atd3p279438184>
10. Графический элемент. Название сайта: SVG Repo. URL: <https://www.svgrepo.com/svg/483909/exit-2>
11. Создание UI тестов на Jatrpack Comprouse. Название сайта: Android Developers. URL: <https://developer.android.com/codelabs/jetpack-compose-testing#2>
12. Compose тестирование. Название сайта: Хабр. URL: <https://habr.com/ru/articles/674112/>

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Main.kt

```
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.unit.dp
import androidx.compose.ui.window.Window
import androidx.compose.ui.window.WindowPosition
import androidx.compose.ui.window.application
import androidx.compose.ui.window.rememberWindowState
import view.App
import view.GraphViewState

fun main() = application {
    val windowState = rememberWindowState(
        width = 1000.dp,
        height = 600.dp,
        position = WindowPosition.Aligned(Alignment.Center)
    )
    val state = remember { GraphViewState() }
    Window(state = windowState, onCloseRequest = ::exitApplication) {
        App(state)
    }
}
```

Graph.kt

```
import kotlin.math.sqrt

class Graph {
    private val matrix = mutableListOf<Int>()
    val size get() = sqrt(matrix.size.toDouble()).toInt()

    fun addVertex() {
        val initialEdge = 0
        val size = size
        val iterator = matrix.listIterator()
        var index = 0
        if (!iterator.hasNext()) {
            iterator.add(initialEdge)
            return
        }
        while (iterator.hasNext()) {
            iterator.next()
            index++
        }
        if (index % size == 0) {
            iterator.add(initialEdge)
        }
        repeat(size + 1) {
            iterator.add(initialEdge)
        }
    }
}
```

```

override fun toString(): String { val
builder = StringBuilder() val size =
size
if (size == 0) return "empty graph"
val maxDigits = matrix.maxOf { it.toString().length }
val formatter = "%0${maxDigits}d"for (i in
0 until size) {
builder.append("| ")
for (j in 0 until size) {
val edge = matrix[i * size + j] val cell =
formatter.format(edge)builder.append(cell)
if (j < size - 1) { builder.append(" ")
}
}
builder.append(" |\n")
}
return builder.toString()
}

operator fun set(src: Int, dest: Int, weight: Int) {if
(src == dest) return // no self reference
val size = size
matrix[src * size + dest] = weightmatrix[src
+ dest * size] = weight
}

operator fun get(src: Int, dest: Int) = matrix[src * size + dest]fun

edges(): Sequence<Edge> = sequence {
val size = size
for (j in 0 until size) { for (i in
0 until j) {
val weight = this@Graph[i, j]if (weight > 0) {
yield(Edge(i, j, weight))
}
}
}
}

fun replace(graph: Graph) { matrix.clear()
matrix.addAll(graph.matrix)
}

fun isConnected() =try {
kruskal().lastOrNull()true
} catch (e: GraphNotConnectedException) {false
}

fun removeVertex(vertex: Int) {

```

```

val iterator = matrix.listIterator() var index
= 0
if (!iterator.hasNext()) {return
}
val size = size
while (iterator.hasNext()) {
iterator.next()
if (index % size == vertex || index in size * vertex until
size * (vertex + 1)) {
iterator.remove()
}
index++
}

}

fun clear() {
matrix.clear()
}

}

```

Kruskal.kt

```

data class Edge(val src: Int, val dest: Int, val weight: Int)

private fun find(subsets: IntArray, i: Int): Int {if
(subsets[i] != i) {
subsets[i] = find(subsets, subsets[i])
}
return subsets[i]
}

private fun union(subsets: IntArray, x: Int, y: Int) {
val xRoot = find(subsets, x)
val yRoot = find(subsets, y)
subsets[xRoot] = yRoot
}

fun Graph.kruskal() = sequence {
val result = mutableListOf<Edge>() val
edges = edges().toList()
if (edges.isEmpty()) {
throw GraphNotConnectedException()
}
val sortedEdges = edges.sortedBy(Edge::weight) val
vertices = size
val subsets = IntArray(vertices) { it }
var i = 0 var e
= 0
while (e < vertices - 1) {
if (i >= sortedEdges.size) {
throw GraphNotConnectedException()
}
val nextEdge = sortedEdges[i++]

```

```

val x = find(subsets, nextEdge.src) val y =
find(subsets, nextEdge.dest)

if (x != y) {
yield(nextEdge)
result.add(nextEdge) union(subsets,
x, y)e++
}
}

println("Minimum Spanning Tree:") for
(edge in result) {
println("${edge.src} -- ${edge.dest} weight: ${edge.weight}")
}
}

class GraphNotConnectedException : IllegalStateException()

```

Application.kt

```

package view

import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.Image
import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.itemsIndexed
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Button
import androidx.compose.material.MaterialTheme.colors
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.ColorFilter
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.platform.testTag
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import theme.MainTheme import
utils.clickableHidden import
utils.strings
import utils.fromFile import
java.awt.FileDialog import
java.awt.Frame import
java.nio.file.Paths import
kotlin.io.path.Path

@Composable
fun App(state: GraphViewState) {

```



```

MainTheme {
    Surface {
        if (state.screen == Screen.Start) {
            StartScreen(state)
        }
        if (state.screen == Screen.Graph) {
            GraphScreen(state)
        }
    }
}

```

```

@Composable
private fun StartScreen(state: GraphViewState) {
    var showFilePicker by remember { mutableStateOf(false) }
    Column(
        modifier = Modifier.fillMaxSize().testTag("StartScreen"),
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        Spacer(Modifier.weight(1f))

        if (showFilePicker) {
            LaunchedEffect(Unit) {
                showFilePicker = false
                FileDialog(null as Frame?, "Select File to Open").apply {
                    mode = FileDialog.LOADisVisible = true
                    directory ?: file ?: return@apply
                    val graph = fromFile(Paths.get(directory,
                        file).toString()) ?: return@apply
                    state.replaceGraph(graph) state.openGraphScreen()
                }
            }
        }
        Button(onClick = { showFilePicker = true }) {
            Text(strings.buttonFromFile)
        }
        Button(onClick = { state.openGraphScreen() }) {
            Text(strings.buttonManual)
        }
        Spacer(Modifier.weight(1f))
    }
}

```

```

@Composable
private fun GraphScreen(state: GraphViewState) {
    Column(modifier = Modifier.fillMaxSize().testTag("GraphScreen")) {
        Box(modifier.clickableHidden { state.openStartScreen() }) {
            Image(
                modifier = Modifier.padding(16.dp).size(36.dp), painter =
                painterResource("exit-2-svgrepo-com.svg"), contentDescription = "back",
                colorFilter = ColorFilter.tint(colors.onSurface),
            )
        }
        Row(modifier = Modifier.fillMaxSize().padding(horizontal = 16.dp),

```

```

vertical = 16.dp)) {
Box(Modifier.weight(4f)) { GraphView(state) }
ControlPanel(Modifier.weight(4f), state)
}
}
}

@Composable
fun ControlPanel(modifier: Modifier, state: GraphViewState) {
Column(
modifier = modifier,
horizontalAlignment = Alignment.CenterHorizontally,
) {
Box(
Modifier.fillMaxWidth().border(BorderStroke(2.dp,
SolidColor(colors.onSurface)), RoundedCornerShape(16.dp))
) {
Text(
text = strings.instructions,
modifier = Modifier.fillMaxWidth().padding(16.dp),textAlign =
TextAlign.End
)
}
Spacer(Modifier.weight(1f))

Button(onClick = { state.addVertex() }) {
Text(strings.buttonAddVertex) }
if (state.isRunning) {
Button(onClick = { state.next() }) { Text(strings.buttonNext)
}
} else if (state.isStarted) {
Button(onClick = { state.complete() }) {
Text(strings.buttonComplete) }
} else {
Button(
onClick = {
state.startKruskal()state.next()
},
enabled = state.isConnected,
) { Text(strings.buttonStart) }
}
if (state.isStarted) {
LazyColumn(Modifier.weight(2f).padding(vertical = 24.dp)) {
itemsIndexed(state.coloredEdges) { index, edge ->
Text(strings.stepDescription(edge))
if (index == state.coloredEdges.size - 1) {
Text(text = strings.stepBottom)
}
}
}
} else {
Spacer(Modifier.weight(2f))
}
}
}
}

```

GraphView.kt

```
@file:OptIn(
ExperimentalFoundationApi::class,
ExperimentalMaterialApi::class,
ExperimentalComposeUiApi::class,
)

package view

import Edge
import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.gestures.onDrag
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.material.MaterialTheme.colors
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.ExperimentalComposeUiApi
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.drawBehind import
androidx.compose.ui.focus.FocusRequesterimport
androidx.compose.ui.focus.focusRequesterimport
androidx.compose.ui.geometry.Offset import
androidx.compose.ui.geometry.lerp
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.graphicsLayer
import androidx.compose.ui.input.pointer.PointerEventType.Companion.Move
import androidx.compose.ui.input.pointer.onPointerEvent
import androidx.compose.ui.platform.testTag
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.times
import utils.*

@Composable
fun GraphView(state: GraphViewState) {
    var mousePosition by remember { mutableStateOf(Offset(0f, 0f)) }
    BoxWithConstraints(Modifier
        .fillMaxSize()
        .onPointerEvent(Move) { mousePosition = it.changes[0].position }
        .clickableHidden(onClick = state::cancelSelection)
    ) {
        state.edges.forEach { EdgeView(state, it) }
        SelectLine(state, mousePosition)
        state.coords.forEachIndexed { index, offset -> VertexView(state,
            index, offset) }
    }
    WeightDialog(state)
}

@Composable
private fun BoxWithConstraintsScope.EdgeView(state: GraphViewState, edge:
Edge) {
```

```

val strokeSize = minDimension * strokeWidth

val (src, dest, weight) = edge
val vertex1 = state.vertex(src).toAbsolute() - Offset(0f,
strokeSize.value / 2)
val vertex2 = state.vertex(dest).toAbsolute() - Offset(0f,
strokeSize.value / 2)

if (weight > 0) {
val isColored = state.coloredEdges.any { (s, d) -> (s == src && d
== dest) || (d == src && s == dest) }
val color = if (isColored) Color.Red else colors.primary
EdgeLine(vertex1, vertex2, color, Modifier
.clickableHidden { state.removeEdge(src, dest) }
.testTag("EdgeView $edge")
)
val textOffset = lerp(vertex2, vertex1, 0.3f).toInt()
Text(strings.edgeWeight(weight), Modifier.offset { textOffset }, color =
colors.onSurface)
}
}

@Composable
private fun BoxWithConstraintsScope.EdgeLine(
vertex1: Offset,
vertex2: Offset, color:
Color,
modifier: Modifier = Modifier,
) {
val strokeSize = minDimension * strokeWidth val
angle = vertex1.angle(vertex2).toDegrees() val scaleX
= vertex1.distance(vertex2)
Box(
modifier = modifier
.offset { vertex1.toInt() }
.graphicsLayer(rotationZ = angle, scaleX = scaleX,
transformOrigin = TransformOriginStart)
.background(color)
.height(strokeSize)
.width(1.dp)
.then(modifier)
)
}

@Composable
private fun BoxWithConstraintsScope.VertexView(
state: GraphViewState,
index: Int, offset:
Offset,
) {
val color = colors.secondaryVariant
val pressedColor = remember { Color(0, 0, 0, 0x40) }
Box(
modifier = Modifier
.size(vertexSize)
.testTag("VertexView $index")
.offset(offset.x * maxWidth - vertexSizeHalf, offset.y *

```

```

maxHeight - vertexSizeHalf)
    .drawBehind {
drawCircle(color)
if (index == state.selectedVertex)
drawCircle(pressedColor)
    }
    .clickableHidden { state.vertexClicked(index) }
    .onDrag { state.drag(index, Offset(it.x / maxWidth.value,
it.y / maxHeight.value)) }
    ) {
Text(
text = strings.vertexNumber(index),
modifier = Modifier.fillMaxWidth().align(Alignment.Center),textAlign =
TextAlign.Center,
color = colors.onSecondary,
style = MaterialTheme.typography.button,
)
    }
}

@Composable
private fun BoxWithConstraintsScope.SelectLine(state: GraphViewState,
mousePosition: Offset) {
val selected = state.selectedVertexif
(selected != -1) {
val vertex = state.vertex(selected).toAbsolute()
EdgeLine(vertex, mousePosition, colors.primary)
}
}

@Composable
private fun WeightDialog(parentState: GraphViewState) {
val state = parentState.openDialog ?: return AlertDialog(
onDismissRequest = { parentState.closeDialog() },buttons =
{
val focusRequester = remember { FocusRequester() }
LaunchedEffect(Unit) { focusRequester.requestFocus() }var text by
remember { mutableStateOf("") }
val isError = text.isNotBlank() && text.toIntOrNull().let {it == null ||
it < 0 }
Column(Modifier.padding(24.dp)) {
TextField(
value = text, isError = isError,
singleLine = true,modifier =
Modifier.fillMaxWidth().focusRequester(focusRequester).onEnter {
if (!isError) { parentState.connect(state.src, state.dest,
text.toInt())
parentState.closeDialog()
}
},
onValueChange = { text = it },
)
}
}

```

```

        if (isError) {
            Text(
                text = strings.enterNumber,color =
                    colors.error,
                style = MaterialTheme.typography.caption, modifier =
                    Modifier.padding(start = 12.dp, top =
8.dp)
            )
        }
        Button(modifier = Modifier.padding(top = 8.dp), enabled =text.isNotBlank()
            && !isError, onClick = {
            parentState.connect(state.src, state.dest,
                text.toInt())
                parentState.closeDialog()
            }) { Text(strings.buttonAccept) }
        },
    )
}

```

GraphViewState.kt

```
package view
```

```

import Edge
import Graph
import androidx.compose.foundation.layout.BoxWithConstraintsScope
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue import
import androidx.compose.ui.geometry.Offset import
import androidx.compose.ui.unit.min
import kruskal
import kotlin.math.cos
import kotlin.math.sin

const val vertexRadius = 0.1f
const val strokeWidth = 0.01f
val BoxWithConstraintsScope.minDimension get() = min(maxWidth, maxHeight)
val BoxWithConstraintsScope.vertexSize get() = minDimension * vertexRadius

val BoxWithConstraintsScope.vertexSizeHalf get() = vertexSize / 2

class GraphViewState(private val graph: Graph = Graph()) {

    private val vertices get() = graph.size
    private val angle: Double get() = 2 * Math.PI / graph.sizevar

    coords by mutableStateOf(listOf<Offset>())

    var coloredEdges by mutableStateOf(listOf<Edge>())var

    isConnected by mutableStateOf(false)

    var selectedVertex by mutableStateOf(-1)

    var edges by mutableStateOf(listOf<Edge>())

```

```

var openDialog by mutableStateOf<DialogState?>(null) var

screen by mutableStateOf(Screen.Start)

private var steps: Iterator<Edge>? = null
set(value) {
field = value coloredEdges = listOf()
onStepsUpdated()
}

var isRunning by mutableStateOf(false) var

isStarted by mutableStateOf(false)

init {
onGraphUpdated()
}

fun replaceGraph(graph: Graph) {
complete() this.graph.replace(graph)
coords = listOf() onGraphUpdated()
}

fun startKruskal() {
steps = graph.kruskal().onEach { edge ->
coloredEdges += edge onStepsUpdated()
}.iterator()
}

fun next() {
val s = steps
if (s?.hasNext() == true) {s.next()}
}

fun complete() {steps
= null
}

private fun onStepsUpdated() {val
steps = steps
isRunning = steps.let { it != null && it.hasNext() }
isStarted = steps != null
}

private fun resetCoords(radius: Float = 0.5f) {coords
= (0 until vertices).map { index ->
val angleReal = index * angle
Offset(
(radius * cos(angleReal) + radius).toFloat()
.coerceIn(2 * vertexRadius, 2 * (radius -

```

```

vertexRadius)),
        (radius * sin(angleReal) + radius).toFloat()
        .coerceIn(2 * vertexRadius, 2 * (radius -
vertexRadius))),
)
}
}

fun addVertex() {
graph.addVertex()
if (coords.isEmpty()) {
resetCoords()
} else {
coords += Offset(
(coords.sumOf { it.x.toDouble() } /
coords.size).toFloat(),
(coords.sumOf { it.y.toDouble() } /
coords.size).toFloat(),
)
}
onGraphUpdated()
}

fun vertex(index: Int) = coords[index]

fun connect(src: Int, dest: Int, value: Int) {
graph[src, dest] = value
onGraphUpdated()
}

fun removeEdge(i: Int, j: Int) {
graph[i, j] = 0 // delete edge
onGraphUpdated()
}

private fun onGraphUpdated() { isConnected
= graph.isConnected() complete()
if (coords.isEmpty()) {
resetCoords()
}
edges = graph.edges().toList()
println(graph) onStepsUpdated()
}

private fun removeVertex(index: Int) {
graph.removeVertex(index)
coords = coords.filterIndexed { i, _ -> index != i }
onGraphUpdated()
}

fun drag(selected: Int, dragOffset: Offset) { coords =
coords.mapIndexed { index, offset ->
if (index == selected) (offset + dragOffset).coerced() else offset
}
}

```



```

}

fun vertexClicked(index: Int) {
    selectedVertex = if (selectedVertex == -1) index else {
        if (selectedVertex == index) {
            removeVertex(index)
        } else {
            connect(selectedVertex, index, 1)
            openDialog = DialogState(selectedVertex, index)
        }
    }
    -1
}

fun cancelSelection() {
    selectedVertex = -1
}

fun closeDialog() {
    openDialog = null
}

fun openGraphScreen() { screen
    = Screen.Graph
}

fun openStartScreen() { screen
    = Screen.Startclear()
}

private fun clear() {
    graph.clear() complete()
    coords = listOf()
    onGraphUpdated()
}

}

data class DialogState(val src: Int, val dest: Int)

fun Offset.coerced() = Offset(
    x.coerceIn(0f, 1f),
    y.coerceIn(0f, 1f),
)

```

Screen.kt

```

package view

enum class Screen {
    Start, Graph,
}

```

_Initializer.kt

```
package utils

import Graph
import androidx.compose.ui.res.useResource
import kruskal
import java.io.File
import java.util.stream.Stream
import kotlin.streams.asStream
import kotlin.streams.toList

fun fromFile(path: String) = try {
    Graph().apply {
        File(path).useLines { lines ->
            fillGraph(lines.asStream())
        }
    }
} catch (_: Exception) {
    null
}

fun Graph.fillGraph(lines: Stream<String>) {
    val connections = lines.map { line ->
        line.split(" ").take(3).map(kotlin.String::toInt)
    }.toList()
    val vertices = connections.flatMap { it.take(2) }.max() + 1
    repeat(vertices) { addVertex() }
    connections.forEach { (from, to, edge) -> this[from, to] = edge }
    println(this)
}
```

_Ui.kt

```
package utils

import androidx.compose.foundation.clickable
import androidx.compose.foundation.interaction.MutableInteractionSource
import androidx.compose.foundation.layout.BoxWithConstraintsScope import
    androidx.compose.runtime.remember
import androidx.compose.ui.ExperimentalComposeUiApi
import androidx.compose.ui.Modifier
import androidx.compose.ui.composed
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.graphics.TransformOrigin
import androidx.compose.ui.input.key.*
import androidx.compose.ui.unit.IntOffset
import kotlin.math.PI
import kotlin.math.atan2
import kotlin.math.pow
import kotlin.math.sqrt

fun Offset.toInt() = IntOffset(x.toInt(), y.toInt())
```

```

context(BoxWithConstraintsScope)
fun Offset.toAbsolute() = Offset(x * maxWidth.value, y * maxHeight.value)

fun Offset.angle(other: Offset) = atan2(other.y - y, other.x - x)

fun Float.toDegrees() = (this * 180 / PI).toFloat()

fun Offset.distance(other: Offset) = sqrt((other.x - x).pow(2) + (other.y - y).pow(2))

@OptIn(ExperimentalComposeUiApi::class) fun
Modifier.onEnter(
    onEnter: () -> Unit
): Modifier = onPreviewKeyEvent {
    if (it.key == Key.Enter && it.type == KeyEvent.Type.KeyDown) {
        onEnter()
    }
    false
}

fun Modifier.clickableHidden(
    onClick: () -> Unit,
): Modifier = composed {
    Modifier.clickable( onClick
    = onClick,
    indication = null, //откл подсветка
    interactionSource = remember { MutableInteractionSource() }
    )
}

val TransformOriginStart = TransformOrigin(0.0f, 0.5f)

strings.kt
package utils

import Edge object

strings {
    const val buttonManual = "Создать вручную" const val
    buttonFromFile = "Загрузить из файла"
    const val instructions = "2 нажатия на вершину = удаление\n" + "1
    нажатие на ребро = удаление\n" +
    "При нажатии на вершину вы можете построить ребро"const val
    buttonAddVertex = "Добавить вершину"
    const val buttonNext = "Дальше"
    const val buttonComplete = "Закончить"const
    val buttonStart = "Начать"
    fun stepDescription(edge: Edge) = "Ребро ${edge.src} - ${edge.dest} с
    минимальным весом ${edge.weight}"
    const val stepBottom = "Поэтому добавляем в каркас"fun
    edgeWeight(index: Int) = index.toString()
    fun vertexNumber(index: Int) = index.toString()const
    val enterNumber = "Введите число"
    const val buttonAccept = "Подтвердить"
}

```

Color.kt

```
package theme
```

```
import androidx.compose.ui.graphics.Color
```

```
val md_theme_light_primary = Color(0xFF8307F0) val
md_theme_light_onPrimary = Color(0xFFFFFFFF)
val md_theme_light_primaryContainer = Color(0xFFEEDCFF) val
md_theme_light_onPrimaryContainer = Color(0xFF2A0054) val
md_theme_light_secondary = Color(0xFF655A6F)
val md_theme_light_onSecondary = Color(0xFFFFFFFF)
val md_theme_light_secondaryContainer = Color(0xFFEBDDF7) val
md_theme_light_onSecondaryContainer = Color(0xFF20182A) val
md_theme_light_tertiary = Color(0xFF80525A)
val md_theme_light_onTertiary = Color(0xFFFFFFFF)
val md_theme_light_tertiaryContainer = Color(0xFFFFFD9DE) val
md_theme_light_onTertiaryContainer = Color(0xFF321018) val
md_theme_light_error = Color(0xFFBA1A1A)
val md_theme_light_errorContainer = Color(0xFFFFDAD6)
val md_theme_light_onError = Color(0xFFFFFFFF)
val md_theme_light_onErrorContainer = Color(0xFF410002)
val md_theme_light_background = Color(0xFFFFFBBF)
val md_theme_light_onBackground = Color(0xFF1D1B1E)
val md_theme_light_surface = Color(0xFFFFFBBF)
val md_theme_light_onSurface = Color(0xFF1D1B1E)
val md_theme_light_surfaceVariant = Color(0xFFE8E0EB) val
md_theme_light_onSurfaceVariant = Color(0xFF4A454E) val
md_theme_light_outline = Color(0xFF7B757F)
val md_theme_light_inverseOnSurface = Color(0xFFFF5EFF4)
val md_theme_light_inverseSurface = Color(0xFF1D1B1E) val
md_theme_light_inversePrimary = Color(0xFFD9B9FF) val
md_theme_light_shadow = Color(0xFF000000)
val md_theme_light_surfaceTint = Color(0xFF8307F0)

val md_theme_dark_primary = Color(0xFFD9B9FF) val
md_theme_dark_onPrimary = Color(0xFF450085)
val md_theme_dark_primaryContainer = Color(0xFF6300BA) val
md_theme_dark_onPrimaryContainer = Color(0xFFEEDCFF)
```

```

val md_theme_dark_secondary = Color(0xFFCFC1DA) val
md_theme_dark_onSecondary = Color(0xFF352D40)
val md_theme_dark_secondaryContainer = Color(0xFF4C4357) val
md_theme_dark_onSecondaryContainer = Color(0xFFEBDDF7) val
md_theme_dark_tertiary = Color(0xFFFF2B7C1)
val md_theme_dark_onTertiary = Color(0xFF4B252D)
val md_theme_dark_tertiaryContainer = Color(0xFF653B43) val
md_theme_dark_onTertiaryContainer = Color(0xFFFFD9DE) val
md_theme_dark_error = Color(0xFFFFB4AB)
val md_theme_dark_errorContainer = Color(0xFF93000A)
val md_theme_dark_onError = Color(0xFF690005)
val md_theme_dark_onErrorContainer = Color(0xFFFFDAD6)
val md_theme_dark_background = Color(0xFF1D1B1E)
val md_theme_dark_onBackground = Color(0xFFE7E1E5)
val md_theme_dark_surface = Color(0xFF1D1B1E)
val md_theme_dark_onSurface = Color(0xFFE7E1E5)
val md_theme_dark_surfaceVariant = Color(0xFF4A454E) val
md_theme_dark_onSurfaceVariant = Color(0xFFCCC4CF) val
md_theme_dark_outline = Color(0xFF958E99)
val md_theme_dark_inverseOnSurface = Color(0xFF1D1B1E)
val md_theme_dark_inverseSurface = Color(0xFFFFFBFF) val
md_theme_dark_inversePrimary = Color(0xFF8307F0) val
md_theme_dark_shadow = Color(0xFF000000)
val md_theme_dark_surfaceTint = Color(0xFFD9B9FF)

val seed = Color(0xFF8100EF)

```

Shape.kt

```

package theme

import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Shapes
import androidx.compose.ui.unit.dp

val Shapes = Shapes(
    small = RoundedCornerShape(4.dp), medium =
    RoundedCornerShape(12.dp),
)

```

Theme.kt

```

package theme

import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material.MaterialTheme
import androidx.compose.material.darkColors
import androidx.compose.material.lightColors
import androidx.compose.runtime.Composable

const val stronglyDeemphasizedAlpha = 0.6f
const val slightlyDeemphasizedAlpha = 0.87f

private val LightColors = lightColors(
    primary = md_theme_light_primary, secondary =
    md_theme_light_secondary, background =
    md_theme_light_background, surface =
    md_theme_light_surface, error =
    md_theme_light_error,
    onPrimary = md_theme_light_onPrimary, onSecondary
    = md_theme_light_onSecondary, onBackground =
    md_theme_light_onBackground, onSurface =
    md_theme_light_onSurface, onError =
    md_theme_light_onError,
)

private val DarkColors = darkColors( primary
    = md_theme_dark_primary, onPrimary =
    md_theme_dark_onPrimary, secondary =
    md_theme_dark_secondary, onSecondary =
    md_theme_dark_onSecondary, error =
    md_theme_dark_error,
    onError = md_theme_dark_onError, background =
    md_theme_dark_background, onBackground =
    md_theme_dark_onBackground, surface =
    md_theme_dark_surface, onSurface =
    md_theme_dark_onSurface,
)

@Composable fun
MainTheme(
    useDarkTheme: Boolean = isSystemInDarkTheme(),
    content: @Composable () -> Unit
) {
    val colors = if (useDarkTheme) DarkColors else LightColors

    MaterialTheme( colors =
    colors, shapes = Shapes,
    typography = Typography, content =
    content,
    )
}

```

Typography.kt

```

package theme

```

```

import androidx.compose.material.Typography
import androidx.compose.ui.text.TextStyle import
import androidx.compose.ui.text.font.FontFamilyimport
import androidx.compose.ui.text.font.FontWeightimport
import androidx.compose.ui.text.platform.Font import
import androidx.compose.ui.unit.sp

val MontserratFontFamily = FontFamily(
    listOf(
        Font("font/Montserrat-Regular.ttf"), Font("font/Montserrat-
        Medium.ttf", FontWeight.Medium), Font("font/Montserrat-
        SemiBold.ttf", FontWeight.SemiBold)
    )
)

val Typography = Typography(
    // Display Large - Montserrat 57/64 . -0.25pxh1 =
    TextStyle(
        fontFamily = MontserratFontFamily,fontWeight
        = FontWeight.W400, fontSize = 57.sp,
        lineHeight = 64.sp, letterSpacing =
        (-0.25).sp,
    ),

    // Display Medium - Montserrat 45/52 . 0pxh2 =
    TextStyle(
        fontFamily = MontserratFontFamily,fontWeight
        = FontWeight.W400, fontSize = 45.sp,
        lineHeight = 52.sp,
        letterSpacing = 0.sp,
    ),

    // Display Small - Montserrat 36/44 . 0pxh3 =
    TextStyle(
        fontFamily = MontserratFontFamily,fontWeight
        = FontWeight.W400, fontSize = 36.sp,
        lineHeight = 44.sp,
        letterSpacing = 0.sp,
    ),

    // Headline Large - Montserrat 32/40 . 0pxh4 =
    TextStyle(
        fontFamily = MontserratFontFamily,fontWeight
        = FontWeight.W400, fontSize = 32.sp,
        lineHeight = 40.sp,
        letterSpacing = 0.sp,
    ),

    // Headline Medium - Montserrat 28/36 . 0pxh5 =
    TextStyle(
        fontFamily = MontserratFontFamily,fontWeight
        = FontWeight.W400, fontSize = 28.sp,

```

```

lineHeight = 36.sp,
letterSpacing = 0.sp,
),

// Headline Small - Montserrat 24/32 . 0pxh6 =
TextStyle(
fontFamily = MontserratFontFamily,fontWeight
= FontWeight.W400, fontSize = 24.sp,
lineHeight = 32.sp,
letterSpacing = 0.sp,
),

// Title Large - Montserrat 22/28 . 0px
subtitle1 = TextStyle(
fontFamily = MontserratFontFamily,fontWeight
= FontWeight.W400, fontSize = 22.sp,
lineHeight = 28.sp,
letterSpacing = 0.sp,
),

// Title Medium - Montserrat 16/24 . 0.15px
subtitle2 = TextStyle(
fontFamily = MontserratFontFamily,fontWeight
= FontWeight.W500, fontSize = 16.sp,
lineHeight = 24.sp, letterSpacing
= 0.15.sp,
),

// Body Large - Montserrat 16/24 . 0.5pxbody1
= TextStyle(
fontFamily = MontserratFontFamily,fontWeight
= FontWeight.W400, fontSize = 16.sp,
lineHeight = 24.sp, letterSpacing
= 0.5.sp,
),

// Body Medium - Montserrat 14/20 . 0.25pxbody2
= TextStyle(
fontFamily = MontserratFontFamily,fontWeight
= FontWeight.W400, fontSize = 14.sp,
lineHeight = 20.sp, letterSpacing
= 0.25.sp,
),
// Body Small - Montserrat 12/16 . 0.4px
caption = TextStyle(
fontFamily = MontserratFontFamily,fontWeight
= FontWeight.W400, fontSize = 12.sp,
lineHeight = 16.sp, letterSpacing
= 0.4.sp,
),
)

```


ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

AppViewTest.kt

```
@file:OptIn(ExperimentalCoroutinesApi::class)

import androidx.compose.material.Button
import androidx.compose.runtime.mutableStateOf
import androidx.compose.ui.test.*
import androidx.compose.ui.test.junit4.createComposeRule
import kotlinx.coroutines.ExperimentalCoroutinesApi
import kotlinx.coroutines.test.StandardTestDispatcher
import kotlinx.coroutines.test.TestScope
import kotlinx.coroutines.test.runTest
import org.junit.Assert.assertEquals
import org.junit.Assert.assertTrue
import org.junit.Before
import org.junit.Rule
import org.junit.Test
import utils.strings
import view.App
import view.GraphViewState
import kotlin.properties.Delegates
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import utils.fromFile
import view.ControlPanel

class AppViewTest {

    @get:Rule
    val compose = createComposeRule() //создание правил одно на всех

    lateinit var scope: TestScope //переменные будут проинициализированы
    позже
    lateinit var state: GraphViewState
    var showFilePicker by Delegates.notNull<Boolean>()

    val startScreen = hasTestTag("StartScreen")
    val graphScreen = hasTestTag("GraphScreen")
    val buttonManual = hasText(strings.buttonManual) and hasClickAction()
    val buttonStart = hasText(strings.buttonStart) and hasClickAction()
    val buttonNext = hasText(strings.buttonNext) and hasClickAction()

    @Before
    fun before() {
        val dispatcher = StandardTestDispatcher() //позволяет эмулировать
        пользовательские действия в тестах
        scope = TestScope(dispatcher)
        state = GraphViewState()
        showFilePicker = false
    }
}
```

```

@Test
fun `Should display start screen on init`() = scope.runTest {
    compose.run {
        setContent { App(state) } // устанавливает содержимое compose
экрана
        awaitIdle() //ожидает пока все элементы интерфейса будут
доступны для проверки
        onNode(startScreen).assertIsDisplayed() //onNode - возвращает
узел элемента в compose дереве
        onNode(graphScreen).assertDoesNotExist()
    }
}

@Test
fun `Should display graph when manual button pressed`() =
scope.runTest {
    compose.run {
        setContent { App(state) }
        onNode(buttonManual).performClick()
        awaitIdle()
        onNode(startScreen).assertDoesNotExist()
        onNode(graphScreen).assertIsDisplayed()
    }
}
}

```

ControlPanelTest.kt

```

@file:OptIn(ExperimentalCoroutinesApi::class)

import androidx.compose.material.Button
import androidx.compose.runtime.mutableStateOf
import androidx.compose.ui.test.*
import androidx.compose.ui.test.junit4.createComposeRule
import kotlinx.coroutines.ExperimentalCoroutinesApi
import kotlinx.coroutines.test.StandardTestDispatcher
import kotlinx.coroutines.test.TestScope
import kotlinx.coroutines.test.runTest
import org.junit.Assert.assertEquals
import org.junit.Assert.assertTrue
import org.junit.Before
import org.junit.Rule
import org.junit.Test
import utils.strings
import view.App
import view.GraphViewState
import kotlin.properties.Delegates
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import utils.fromFile
import view.ControlPanel
import kotlin.io.path.Path

```

```

class ControlPanelTest {

    @get:Rule
    val compose = createComposeRule() //создание правил одно на всех

    lateinit var scope: TestScope //переменные будут проинициализированы
    позже
    lateinit var state: GraphViewState

    val buttonStart = hasText(strings.buttonStart) and hasClickAction()
    val buttonNext = hasText(strings.buttonNext) and hasClickAction()
    val buttonComplete = hasText(strings.buttonComplete) and
hasClickAction()

    @Before
    fun before() {
        val dispatcher = StandardTestDispatcher() //позволяет эмулировать
пользовательские действия в тестах
        scope = TestScope(dispatcher)
        state = GraphViewState()
    }

    @Test
    fun `Should start button pressed`() = scope.runTest {
        compose.run {
            val graph =
fromFile("C:\\Users\\dns\\projects\\kruskal1\\src\\jvmMain\\resources\\gr
aph2.txt") ?: return@run
            state.replaceGraph(graph)
            setContent { ControlPanel(Modifier, state) }
            awaitIdle()
            onNode(buttonStart).assertIsDisplayed()
            onNode(buttonNext).assertDoesNotExist()
            onNode(buttonStart).performClick()
            awaitIdle()
            onNode(buttonNext).assertIsDisplayed()
            onNode(buttonStart).assertDoesNotExist()
        }
    }

    @Test
    fun `Should stop button pressed`() = scope.runTest {
        compose.run {
            val graph =
fromFile("C:\\Users\\dns\\projects\\kruskal1\\src\\jvmMain\\resources\\gr
aph2.txt") ?: return@run
            state.replaceGraph(graph)
            setContent { ControlPanel(Modifier, state) }
            awaitIdle()
            onNode(buttonStart).performClick()
            awaitIdle()
            onNode(buttonNext).performClick()
            awaitIdle()
            onNode(buttonNext).performClick()
            awaitIdle()
        }
    }
}

```

```

        onNode(buttonComplete).assertIsDisplayed()
    }
}

}

```

GraphViewTest.kt

```

@file:OptIn(ExperimentalCoroutinesApi::class)

import androidx.compose.ui.test.assertIsDisplayed
import androidx.compose.ui.test.hasTestTag
import androidx.compose.ui.test.junit4.createComposeRule
import androidx.compose.ui.test.performClick
import kotlinx.coroutines.ExperimentalCoroutinesApi
import kotlinx.coroutines.test.StandardTestDispatcher
import kotlinx.coroutines.test.TestScope
import kotlinx.coroutines.test.runTest
import org.junit.Assert.assertTrue
import org.junit.Before
import org.junit.Rule
import org.junit.Test
import view.GraphView
import view.GraphViewState
import kotlin.test.assertNotNull

class GraphViewTest {

    @get:Rule
    val compose = createComposeRule() //создание правил одно на всех

    lateinit var scope: TestScope //переменные будут проинициализированы
    позже
    lateinit var state: GraphViewState

    val vertex0 = hasTestTag("VertexView 0")
    val edgeNode = hasTestTag("EdgeView ${Edge(0, 1, 2)}")

    @Before
    fun before() {
        val dispatcher = StandardTestDispatcher() //позволяет эмулировать
пользовательские действия в тестах
        scope = TestScope(dispatcher)
        state = GraphViewState()
    }

    @Test
    fun `Double click removes vertex`() = scope.runTest {
        compose.run {
            state.addVertex()
            setContent { GraphView(state) }
            awaitIdle()
            onNode(vertex0).assertIsDisplayed()
            onNode(vertex0).performClick()
            onNode(vertex0).performClick()
            awaitIdle()
        }
    }
}

```

```

        onNode(vertex0).assertDoesNotExist()
    }
}

@Test
fun `Click removes edge`() = scope.runTest {
    compose.run {
        state.addVertex()
        state.addVertex()
        state.connect(0, 1, 2)
        setContent { GraphView(state) }
        awaitIdle()
        onNode(edgeNode).assertIsDisplayed()
        onNode(edgeNode).performClick()
        awaitIdle()
        onNode(edgeNode).assertDoesNotExist()
    }
}
}

```

GraphTest.kt

```

import org.junit.Assert.assertEquals
import org.junit.Assert.assertTrue
import org.junit.Before
import org.junit.Test

class GraphTest {
    private lateinit var graph: Graph

    @Before
    fun setUp() {
        graph = Graph()
    }

    @Test
    fun addVertex_addsVertexToMatrix() {
        val expectedMatrixSize = 1

        graph.addVertex()
        val actualSize = graph.size
        assertEquals(expectedMatrixSize, actualSize)
    }

    @Test
    fun addVertex_addsCorrectNumberOfEdgesToMatrix() {
        val expectedMatrixSize = 2

        graph.addVertex()
        graph.addVertex()
        val actualMatrixSize = graph.size

        assertEquals(expectedMatrixSize, actualMatrixSize)
    }
}

```

```

@Test
fun testSetAndGet() {
    graph.addVertex()
    graph.addVertex()
    graph[0, 1] = 5
    assertEquals(5, graph[0, 1])
}

@Test
fun testRemoveVertex() {
    graph.addVertex()
    graph.addVertex()
    graph.removeVertex(1)
    assertEquals(1, graph.size)
}

@Test
fun testIsConnected() {
    graph.isConnected() //GraphNotConnectedException
    graph.addVertex()
    graph.addVertex()
    graph[0, 1] = 5
    assertTrue(graph.isConnected())
}
}

```

KruskalTest.kt

```

import org.junit.Assert.*
import org.junit.Before
import org.junit.Test

class KruskalTests {
    private lateinit var graph: Graph

    @Before
    fun setUp() {
        graph = Graph()
    }

    @Test
    fun testKruskalEmptyGraph() {
        try {
            graph.kruskal().toList()
            fail("Expected GraphNotConnectedException")
        } catch (e: GraphNotConnectedException) {
            // Exception
        }
    }

    @Test
    fun testKruskalConnectedGraph() {
        graph.addVertex()
        graph.addVertex()
    }
}

```

```

graph.addVertex()
graph[0, 1] = 5
graph[0, 2] = 3
graph[1, 2] = 2

val minimumSpanningTree = graph.kruskal().toList()
assertEquals(2, minimumSpanningTree.size)
assertEquals(2, minimumSpanningTree[0].weight)
assertEquals(3, minimumSpanningTree[1].weight)
assertEquals(1, minimumSpanningTree[0].src)
assertEquals(2, minimumSpanningTree[0].dest)
assertEquals(0, minimumSpanningTree[1].src)
assertEquals(2, minimumSpanningTree[1].dest)
}

@Test
fun testKruskalDisconnectedGraph() {
    graph.addVertex()
    graph.addVertex()
    graph.addVertex()
    graph.addVertex()
    graph[0, 1] = 5
    graph[3, 2] = 3

    try {
        graph.kruskal().toList()
        fail("Expected GraphNotConnectedException")
    } catch (e: GraphNotConnectedException) {
        // Exception
    }
}

@Test
fun testKruskal() {
    graph.addVertex()
    graph.addVertex()
    graph[0, 1] = 5
    graph.addVertex()
    graph[0, 2] = 7
    graph[1, 2] = 3

    val minimumSpanningTree = graph.kruskal().toList()
    assertEquals(2, minimumSpanningTree.size)
    assertEquals(3, minimumSpanningTree[0].weight)
    assertEquals(5, minimumSpanningTree[1].weight)
}

@Test(expected = GraphNotConnectedException::class)
fun testKruskalException() {
    graph.addVertex()
    graph.addVertex()
    graph.addVertex()

    graph.kruskal().toList() // генерирует GraphNotConnectedException
}

```

}