МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: Визуализация алгоритма Краскала

Студентка гр. 1304 Чернякова А.Д.

Студент гр. 1383 Петров А.С.

Студентка гр. 1303 Хулап О.А.

Руководитель Токарев А.П.

Санкт-Петербург

2023

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студентка Чернякова А.Д. группы 1304

Студент Петров А.С. группы 1383

Студентка Хулап О.А. группы 1303

Тема практики: Визуализация алгоритма Краскала

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Краскала.

Сроки прохождения практики: 30.06.2023 – 13.07.2023

Дата сдачи отчета: 12.07.2023

Дата защиты отчета: 12.07.2023

Студентка гр. 1304	_____	Чернякова А.Д.
Студент гр. 1383	_____	Петров А.С.
Студентка гр. 1303	_____	Хулап О.А.
Руководитель	_____	Токарев А.П.

АННОТАЦИЯ

Целью практики являлось групповое создание проекта на языке программирования Java по визуализации алгоритма Краскала. Основными составляющими проекта являлись: разработка и реализация интерфейса алгоритма, написание самого алгоритма Краскала, тестирование интерфейса и алгоритма, обработка исключений.

SUMMARY

The aim of the practice was to create a group project in Java programming language on visualization of Kraskal's algorithm. The main components of the project were: development and implementation of the algorithm interface, writing the Kraskal algorithm itself, testing the interface and algorithm, exception handling.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью проекта по визуализации алгоритма Краскала является представить этот классический алгоритм в удобном, понятном и наглядном виде.

Алгоритм Краскала — это алгоритм для построения минимального остовного дерева в связном графе. Он основывается на принципе "сначала самые короткие ребра". В процессе работы алгоритма, ребра графа добавляются поочередно от самых коротких к самым длинным, при условии, что добавление ребра не создаст цикла в графе. В результате получается дерево, соединяющее все вершины графа, и обладающее наименьшей суммой весов ребер.

Визуализация алгоритма Краскала позволяет наглядно следить за шагами его работы и визуализировать процесс построения минимального остовного дерева. Она помогает пользователям лучше понять сам алгоритм, его применение и принцип работы.

Путем визуализации структуры данных и алгоритмических шагов становится возможным увидеть изменения графа на каждой итерации алгоритма и понять, каким образом формируется минимальное остовное дерево. Визуализация также позволяет проводить эксперименты с различными входными данными и анализировать эффективность алгоритма в разных сценариях.

Целью проекта является создание интерактивной визуализации алгоритма Краскала с использованием современных технологий, таких как JavaFX. Это позволит пользователям лучше понять и изучить алгоритм, а также использовать визуализацию в учебных или исследовательских целях.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

Основное задание состоит из следующих итераций:

1. Согласование спецификации и плана разработки. Необходимо залить спецификацию и план разработки в репозиторий отдельным файлом или в составе отчёта, или в виде вики - странички и задач в багтрекере. Затем оповестить преподавателя в discord или по почте.
2. Прототип (MVP - minimum viable product). MVP - это приложение, демонстрирующее интерфейс, но почти не реализующее основные функции. Необходимо залить скриншоты прототипа в репозиторий в текстовом файле или на вики страничку с краткими пояснениями (что за что отвечает в интерфейсе). Код прототипа также должен быть залит в репозиторий. Подсказка по дизайну: у пользователя всегда должна быть возможность отменить любое действие, вернуться на шаг назад и начать заново. Отсутствие данных фич - “design smells”
3. Версия 1.0 - все основные элементы приложения функциональны. Защита в дискорде.
4. Версия 2.0 (финальная) - все замечания по предыдущей итерации устранены, все заявленные фичи реализованы. Защита в дискорде.

1.2. Уточнение требований после 1-ой сдачи

1. Сделать комментарии к шагам более подробными

2. Сделать проверку на то, является ли граф связным (алгоритм должен работать только в том случае, если граф связный)

3. Добавить опцию «Шаг назад»

4. Сделать граф больше в размерах

5. Проверка веса ребра при импорте графа

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

1. Составление спецификации. Исполнители: Хулап О. А., Чернякова А. Д, Петров А. С. Срок сдачи: 06.07
2. Составление плана разработки. Исполнители: Хулап О. А., Чернякова А. Д, Петров А. С. Срок сдачи: 06.07
3. Дизайн стартового экрана. Исполнители: Хулап О.А. Срок сдачи: 06.07
4. MVP. Исполнители: Хулап О. А., Чернякова А. Д. Срок сдачи: 08.07
5. Импорт графа из файла. Исполнитель: Чернякова А. Д. Срок сдачи: 09.07
6. Реализация алгоритма Краскала. Исполнители: Чернякова А. Д. Срок сдачи: 09.07
7. Визуализация и графический интерфейс. Исполнители: Петров А. С. Срок сдачи: 10.07
8. Интерактивность и возможность контролировать пользователем. Исполнители: Хулап О. А. Срок сдачи: 10.07
9. Тестирование. Исполнители: Петров А. С. Срок сдачи: 11.07

2.2. Распределение ролей в бригаде

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Главным классом проекта является KruscalApplication. Класс KruscalApplication определяет все необходимые переменные и структуры данных для визуализации. Здесь объявлен граф, который представляет связный граф с весами ребер. Также есть списки и карты для хранения вершин, ребер, цветов и комментариев по шагам алгоритма. Также объявлены некоторые вспомогательные переменные и объекты Stage и Scene для главного окна. Рассмотрим методы из данного класса, расположенного в файле KruscalApplication.java.

Метод checkConnectivityComponents() проверяет связность компонентов графа. Он создает два множества - одно из всех вершин графа и одно, в котором будут добавляться вершины связных компонентов. Затем он проходит циклом, пока количество вершин во множестве связных компонентов не перестанет расти. Внутри цикла происходит поиск вершин, смежных с текущими вершинами связных компонентов, и добавление их в множество связных компонентов. В конце проверяется, что количество вершин во множестве всех вершин графа и множестве связных компонентов совпадает, что говорит о связности всего графа.

Метод edgeExist() проверяет, существует ли ребро между двумя вершинами в графе. Он использует метод getEdge() из библиотеки JGraphT, чтобы получить ребро между двумя вершинами, и возвращает true, если оно существует, и false в противном случае.

Метод stepBack() используется для отката на один шаг назад в алгоритме. Он удаляет последний комментарий из списка, удаляет последнее ребро из списка раскрашенных ребер, удаляет последние вершины из списка вершин

минимального остовного дерева, уменьшает счетчик шагов и уменьшает стоимость дерева на вес удаленного ребра. Затем он обновляет метки на экране и перерисовывает граф.

В методе start() происходит запуск приложения JavaFX. Он создает объект FXMLLoader для загрузки макета из файла hello-view.fxml. Затем он создает, устанавливает сцену в объект Stage. Затем создается панель и добавляется в нее графический элемент Canvas, который будет использоваться для рисования графа. Затем панель добавляется в объект AnchorPane, который в свою очередь добавляется в сцену. Наконец, вызывается метод show() объекта Stage, чтобы отобразить окно на экране.

Методы addVertex() и deleteVertex() добавляют и удаляют вершину в графе соответственно. Они вызывают методы графа addVertex() и removeVertex() для добавления и удаления вершин и затем вызывают метод drawGraph() для перерисовки графа на экране.

Методы addEdge() и deleteEdge() добавляют и удаляют ребро между вершинами в графе соответственно. Они вызывают методы графа addEdge() и removeEdge() для добавления и удаления ребер, а затем вызывают метод drawGraph() для перерисовки графа на экране.

Метод importGraph() импортирует граф из файла. Он считывает строки из файла и разделяет их на элементы по пробелам. Затем он проверяет, что каждая строка содержит 3 элемента, разделенных пробелами. Затем он проверяет, что третий элемент является положительным числом. Если одно из этих условий не выполняется, он показывает предупреждение с соответствующим сообщением. Если все условия выполняются, он продолжает считывать строки и добавлять вершины в список вершин, если их там еще нет.

Функция clearWindow() очищает поле и обнуляет счетчик.

Функция drawGraph() создает объект класса Canvas для отображения графа, устанавливает параметры отображения графа (координаты вершин и ребер), отображает вершины графа на канвасе, отображает ребра графа на канвасе, создает панель и добавляет на нее канвас, устанавливает фон панели и ее позицию на главной сцене, отображает главное окно

Функция sortEdge() выполняет следующие действия:

- Создает двумерный массив spanningTreeCost размером, равным количеству ребер в графе

- Заполняет массив в виде вес ребра, начальная вершина, конечная вершина

- Сортирует массив в порядке возрастания веса ребер

- Возвращает отсортированный массив

Метод doAlgorithm() выполняет алгоритм Крускала для построения минимального остовного дерева графа. Он принимает на вход несколько параметров: label (метка) для отображения информации о процессе выполнения алгоритма, ostovTreeCost - двумерный массив, содержащий информацию о весе ребер, и costLabel - метка для отображения общего веса минимального остовного дерева.

Сначала метод проверяет, достигнут ли конец алгоритма. Если все ребра добавлены в остовное дерево, текст на метке label устанавливается в "Spanning tree built" и метод завершается. Затем метод обнуляет тексты на метках label и costLabel.

Затем из массива ostovTreeCost выбираются вершины v1 и v2, соответствующие текущему шагу. Если оба этих вершины уже содержатся в остовном дереве, увеличиваем шаг на 1 и вызываем метод doAlgorithm() снова.

Затем вершины v1 и v2 добавляются в остовное дерево. Увеличивается счетчик ребер totalEdgesInOstovTree. Получаем ребро edge с помощью метода getEdge() и его вес weight с помощью метода getEdgeWeight(). Добавляем вес в общий вес минимального остовного дерева cost. Добавляем общий вес в список costList. Формируем строку str с информацией о добавленном ребре. Устанавливаем эту строку на метке label. Также устанавливаем на метке costLabel общий вес. Добавляем строку в список комментариев commentList. Увеличиваем шаг на 1. Затем вызывается метод drawGraph(), который отрисовывает граф с добавленным ребром.

Метод main() является точкой входа в программу. Он вызывает метод launch(), который запускает приложение JavaFX.

3.2. Основные методы

В файле Controller.java определен класс Controller, который является контроллером для пользовательского интерфейса. Он содержит различные элементы управления (кнопки, метки и т. д.), которые объявлены с использованием аннотации @FXML для связывания с соответствующими элементами в файле разметки FXML.

Функция initialize() выполняет некоторые действия при инициализации контроллера. Она устанавливает начальные значения некоторых элементов управления и связывает их с соответствующими обработчиками событий.

Функции handleDeleteVertexAction(), handleAddVertexAction() и handleDeleteEdgeAction() являются обработчиками событий для удаления вершин, добавления вершин и удаления ребер соответственно. Они отображают всплывающие окна для взаимодействия с пользователем и вызывают соответствующие методы класса KruscalApplication для выполнения соответствующих действий над графом.

Функции handleImportGraphAction(), handleStepForwardAction(), handleStartAgainAction() и handleStepBackAction() также являются обработчиками событий для импорта графа, выполнения шага вперед, начала заново и выполнения шага назад соответственно. Они вызывают соответствующие методы класса KruscalApplication для выполнения соответствующих действий.

В классе Controller также определена переменная application типа KruscalApplication, которая используется для вызова методов из класса KruscalApplication.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

Во время проверки ошибок выявлено не было, все кнопки выполняют свои функции и обрабатывают исключения. Кнопки позволяют пользователю самостоятельно рисовать граф, а также выбирать его из файла, к каждому действию алгоритма Краскала идет пояснение, нельзя добавить ребро или вершину, если они уже существуют, вес должен являться положительным числом. Файл с входными данными обрабатывается на положительность веса и правильную запись файла. Нельзя удалить ребро или вершину, которые не существуют

4.2. Тестирование алгоритма Краскала

Алгоритм работает корректно, проверяется граф на связность, правильно составляет остовное дерево: в нем содержатся все вершины и нет циклов и петель.

ЗАКЛЮЧЕНИЕ

Цель данного проекта заключалась в визуализации алгоритма Крускала для построения минимального остовного дерева графа. Давайте проанализируем соответствие поставленной цели и полученного результата.

1. Визуализация алгоритма: Цель была достигнута. Проект предоставляет визуализацию каждого шага алгоритма Крускала, позволяя пользователям наглядно наблюдать процесс построения минимального остовного дерева графа.

2. Построение минимального остовного дерева: Цель также была достигнута. Проект генерирует минимальное остовное дерево графа с использованием алгоритма Крускала. Каждый добавленный шаг отображается на графе, позволяя пользователям видеть, какие ребра были добавлены и какая структура остовного дерева накоплена.

3. Понятность и удобство использования: Проект достигает этой цели, предоставляя простой и интуитивно понятный интерфейс визуализации. Пользователи могут легко взаимодействовать с графом, отображаемым на экране, и видеть информацию о каждом добавленном ребре.

4. Техническая реализация: Проект реализован с использованием языка программирования Java и JavaFX для создания пользовательского интерфейса. Алгоритм Крускала реализован в методе doAlgorithm(). Весь проект организован вокруг объекта Graph, который содержит информацию о вершинах и ребрах графа.

В итоге, проект по визуализации алгоритма Крускала соответствует поставленной цели и успешно визуализирует каждый шаг алгоритма, строит минимальное остовное дерево графа и обеспечивает понятность и удобство использования для пользователей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Kathy Sierra, Bert Bates "Head First Java", 2005

2. Herbert Schildt, "Java: A Beginner's Guide", 2019

3. Hendrik Ebbers, "Mastering JavaFX 8 Controls", 2018

4. Oracle, "Oracle JavaFX Documentation", 2012

5. Oracle, "Getting Started with JavaFX", 2008

7. Web: https://www.youtube.com/watch?v=FLkOX4Eez6o

8. Web: https://www.youtube.com/watch?v=tyeqqz7o2pI

9. Web: https://youtu.be/FLkOX4Eez6o

ПРИЛОЖЕНИЕ А

НАЗВАНИЕ ПРИЛОЖЕНИЯ

1.  Controller.java
2.  KruscalApplication.java
3.  hello-view.fxml