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

ОТЧЕТ

по учебной практике

Тема: АЛГОРИТМ БОРУВКИ

Студент гр. 8304 _____ Порывай П.А

Студент гр. 8304 _____ Карабанов Р.Е.

Студент гр. 8304 _____ Ивченко А.А.

Руководитель _____ Ефремов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Порывай П.А группы 8304
Студент Карабанов Р.Е группы 8304
Студент Ивченко А.А. группы 8304

Тема практики: Алгоритм Борувки

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.

Алгоритм: <визуализируемый алгоритм>.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 00.07.2020

Дата защиты отчета: 00.07.2020

Студент _____ Порывай П.А

Студент _____ Карабанов Р.Е

Студент _____ Ивченко А.А

Руководитель _____ Ефремов М.А.

АННОТАЦИЯ

Целью текущей учебной практики является разработка GUI приложения для нахождения минимального остовного дерева для заданного графа с помощью алгоритма Борувки.

Программа разрабатывается на языке Java командой из трех человек, каждый из которых имеет определенную специализацию. Сдача и показ проекта на определенном этапе выполнения осуществляется согласно плану разработки.

SUMMARY

The purpose of this paper is the development of a graphical application interface for a minimum spanning tree for a given graph using the Boruwka algorithm.

The program is developed in Java by a team of three students. The project is carried out in accordance with the development plan.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данного совместного проекта является реализация программы, выполняющей визуализацию алгоритма Борувки нахождения минимального остовного дерева с пояснениями.

Программа предоставляет пользователю структурированный интерфейс для создания и изменения графа, а также возможность отслеживать состояние графа. Во время визуализации работы алгоритма Борувки пользователь может вернуться к предыдущему шагу, либо перейти к следующему. Каждый шаг алгоритма имеет графическое отображение.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

Программа представляет собой визуализацию алгоритма Борувки, нахождения минимального остовного дерева для взвешенного неориентированного графа.

1.1 Требования к вводу исходных данных

Программа предоставляет пользователю следующий интерфейс для создания графа:

- Чтения графа из файла
- Изменения текущего графа
- Генерация графа по входным параметрам

1.2 Требования к визуализации

Графический интерфейс предоставляет пользователю следующие методы для изменения графа:

- Добавить вершину
- Соединить вершины
- Применить алгоритм
- Удалить вершину

Пользователь имеет возможность применить алгоритм Борувки для построенного графа, если он удовлетворяет некоторым условиям. Алгоритм Борувки нахождения минимального остовного дерева для заданного графа имеет два способа реализации:

- Вывод минимального остовного дерева немедленно.
- Пошаговая визуализация всех итераций алгоритма.

Если выбран вариант пошаговой визуализации алгоритма, пользователю предоставляется возможность самому управлять последовательность

выполнения алгоритма: он может либо «откатиться» к предыдущей итерации, либо пройти к следующей. Таким образом, программа предлагает следующий интерфейс для прохода по алгоритму:

- Вперед (не доступен на последнем шаге)
- Назад (не доступен на первом шаге)

Во время работы алгоритма окрашивается минимальное по весу ребро, пока граф не примет вид леса, каждый элемент которого имеет оригинальный цвет. При объединении компонент связности оба поддерева окрашиваются в одинаковый цвет.

Результатом работы алгоритма будет удаление всех ребер, не входящих в минимальное остовное дерево.

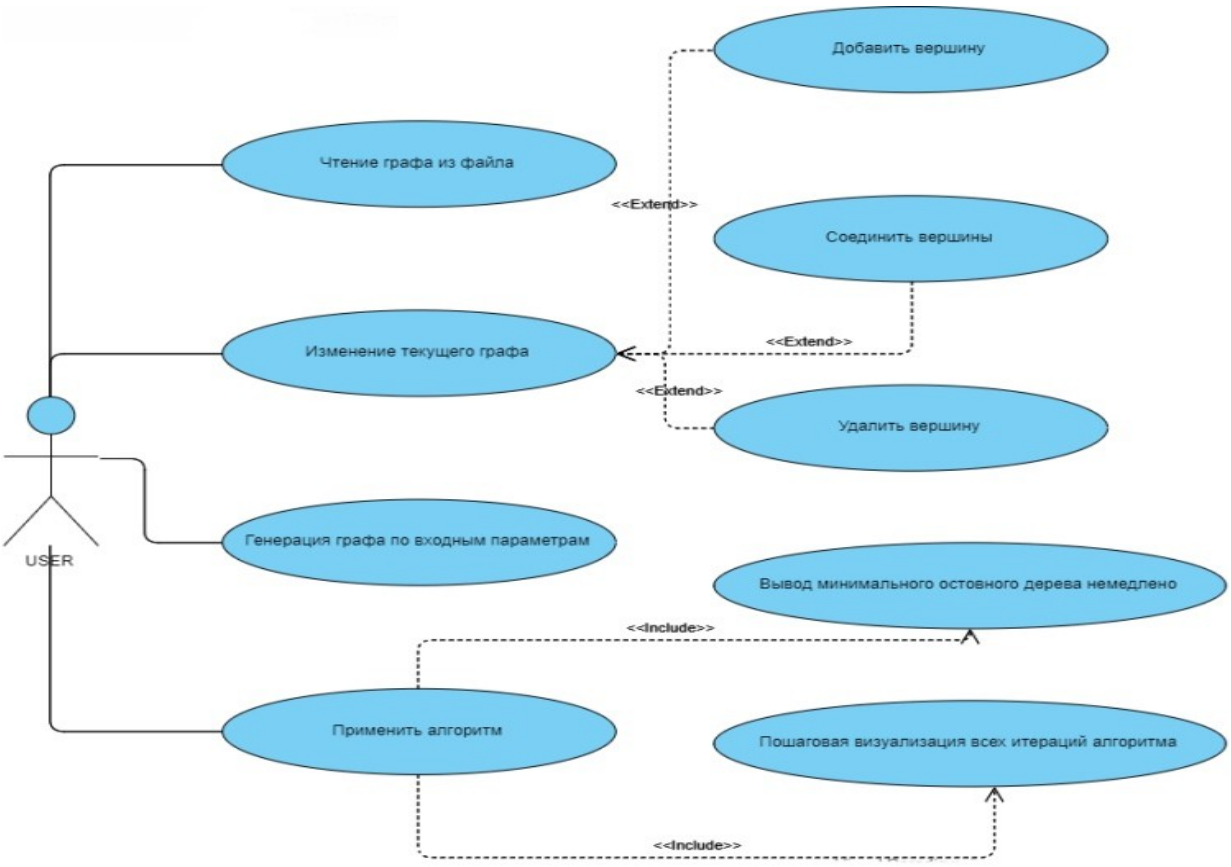

Рисунок 1 – use-case диаграмма.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1План разработки

02 июля: Создание директории, которая должна содержать файл с исходным кодом и скрипт для запуска.

04 июля: Создание интерфейс с заглушками: все элементы интерфейса размещены, но могут не исполнять функционал.

06 июля: Решение алгоритма при нажатии на кнопку графического интерфейса и отображение конечного результата работы алгоритма.

08 июля: Сделан прототип программы в котором визуализируется как получение и отображение результата сразу, так и пошаговое выполнение алгоритма.

10 июля: Программа полностью корректно собирается из исходников в один исполняемый jar архив.

2.2Распределение ролей в бригаде

Ивченко А.А. - визуализация программы

Порывай П.А. - логика алгоритма

Карабанов Р.Е. - тестирование и сборка программы

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Пакет simple

public class Edge – класс, определяющий ребро, с соответствующими полями, имеет методы присвоения полям значений, а также методы возврата этих значений

public class Graph implements ILoadable, IChangable – класс, определяющий граф и методы интерфейсов ILoadable – для обобщенной загрузки графа из файла, IChangable – для изменения графа

public class Node – класс узел для GUI, (через него задаеются координаты вершины в графе)

public class Subset – класс, хранящий родителя и ранг(вспомогательный для алгоритма Борувки)

Пакет GUI

public class GUI extends JFrame – GUI, выводящийся на экран

public class GraphPainter extends JPanel – класс, реализующий рисование графа

3.2. Основные методы

public ArrayList<Edge> boruvkaMST(Graph graph, GraphPainter GraphPanel, boolean vizualization) – основной алгоритм Боруски

В public class GUI extends JFrame:

public GUI создает интерфейс используя пакет swing, и наследуется от «главного” окна содержит объекты «кнопки”, “ждущие” объекты, которые создаются при нажатии кнопок, объекты «подокон”, которые в свою очередь добавляются стандартным методом add

тонструктор GUI, создает графический интерфейс.

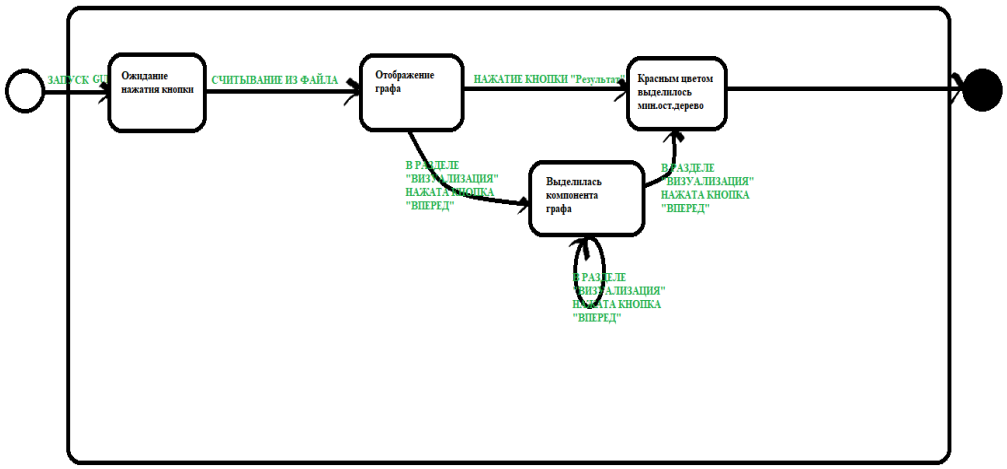

Рисунок 2 – автомат работы визуализации.

4. ТЕСТИРОВАНИЕ

4.1. План тестирования программы.

Проведение тестирования программы планируется с использованием библиотеки для модульного тестирования Junit. Объектами тестирования станут ключевые методы, используемые в алгоритме Борувки.

Таблица 1 – План тестирования.

Объект тестирования	**Данные**	**Результаты**
Метод generateGraph()	Количество рёбер и вершин графа	После выполнения метода должен быть создан граф с определёнными значениями.
Метод readFromFile()	Имя файла, в котором записан граф.	Должен быть считан граф, записанный в файле.
Метод saveGraph()	Генерируемый граф из определённого числа вершин и ребер	Запись графа в файл и проверка файла на пустоту.
Метод clear()	Генерируемый граф из определённого числа вершин и ребер	Очистка данный графа.
Метод addNode ()	Считанный из файла	Проверка добавления

	граф	вершины в граф.
Метод addEdge()	Считанный из файла граф	Проверка добавления ребра в граф.

4.2 Результаты тестирования.

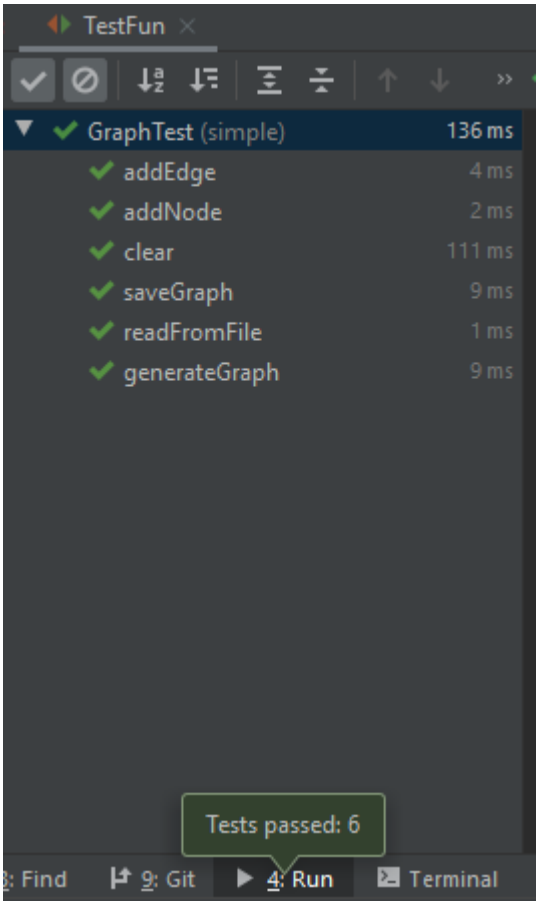

Рисунок 3 - результаты тестирования

ЗАКЛЮЧЕНИЕ

Таким образом, командой была реализована работа и визуализация алгоритма Борувки. Пользователь имеет возможность задать входные данные двумя способами – вручную, с помощью файла и сгенерировать случайный граф. Есть возможность пошагового выполнения алгоритма.

В ходе совместной работы были получены практические знания о ЯП Java, системе тестирования Junit и навыках командной разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Борувка : https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Борувки

2. Junit : https://junit.org/junit5/docs/current/user-guide/#writing-tests-classes-and-methods

3. Maven: https://maven.apache.org

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД АЛГОРИТМА.

```
package simple;
import java.awt.Color;
import java.awt.List;
import java.util.ArrayList;

import javax.swing.JButton;
import javax.swing.JTextField;

import gui.GraphPainter;

public class Boruvka{

	private int iter;
	private ArrayList<Edge> sortEdges;
	public int MSTweight;

	public ArrayList<Edge> boruvkaMST(Graph graph, JTextField textField) {


	     iter = 0;
	       int vertNum = graph.factVertNum;
	       int edgeNum = graph.factEdgeNum;
	       this.sortEdges = new ArrayList<Edge>();
	       System.out.println(graph.factVertNum + " " + graph.factEdgeNum + "
" + graph.getVertNum() + " " + graph.getEdgeNum());
	       if (graph.isolated) {System.out.println (true); return sortEdges;}
	       else {System.out.println(false);};
	       String[] vertNames = graph.getVertNames();
	       Edge[] edges = graph.getEdges();

	       Subset[] subsets = new Subset[vertNum];
	       int[] cheapest = new int[vertNum];
	       for(int i = 0; i < edgeNum; i++) {
	                 if(edges[i]!=null)
	                 System.out.println(edges[i].getDest() + " " +
edges[i].getSrc());
	                 else
	                 System.out.println("null");
	       }
```

```
System.out.println(vertNum + edgeNum + "fdsgdg");
int k = 0;
for (int i = 0; i < vertNum; i++) {
    if ( vertNames[i] != "0") {
          subsets[i] = new Subset();
          subsets[i].setParent(i);
          subsets[i].setRank(0);
          cheapest[i] = -1;
    }else {subsets[i] = null;cheapest[i] = -2;k++;}
}

int numTree = vertNum - k;
k = 0;
MSTweight = 0;

System.out.println("Initializing Boruvka's MST" + numTree);

while (numTree > 1) {
    System.out.println("Number of Trees:" + numTree);

    //Reset the cheapest values every iteration
    for (int i = 0; i < vertNum; i++) {
           if(cheapest[i] != -2) cheapest[i] = -1;

    }

    //Iterate over all edges to find the cheapest
    //edge of every subtree
    for (int i = 0; i < edgeNum; i++) {
          if (edges[i]!= null) {
          if ( edges[i].getSrc() != -1) {
        //Find the subsets of the corners of the edge
        int set1 = find(subsets, edges[i].getSrc());
        int set2 = find(subsets, edges[i].getDest());

        //If the two corners belong to the same subset,
        //ignore the current edge
        if (set1 != set2) {

            //If they belong to different subsets, check which
            //one is the cheapest
```

```
                        if (cheapest[set1] == -1 ||
edges[cheapest[set1]].getWeight() > edges[i].getWeight()) {

                            cheapest[set1] = i;
                        }

                        if (cheapest[set2] == -1 ||
edges[cheapest[set2]].getWeight() > edges[i].getWeight()) {

                            cheapest[set2] = i;
                        }
                    }
                    }
                }
                }
                //Add the cheapest edges obtained above to the MST
                for (int j = 0; j < vertNum; j++) {
                      if (vertNames[j] != "0") {

                    //Check if the cheapest for current set exists

                    if ((cheapest[j] != -1) && (cheapest[j] != -2)) {

                      int set1 = find(subsets, edges[cheapest[j]].getSrc());
                      int set2 = find(subsets, edges[cheapest[j]].getDest());



                      if (edges[cheapest[j]].getSrc() != -1) {
                        if(set1 != set2){
                            MSTweight += edges[cheapest[j]].getWeight();
                            System.out.println("Edge ("+
vertNames[edges[cheapest[j]].getSrc()] + ", " +
vertNames[edges[cheapest[j]].getDest()]+") added to the MST");

                            Edge e = new Edge(edges[cheapest[j]].getSrc(),
edges[cheapest[j]].getDest(), edges[cheapest[j]].getWeight());
                            sortEdges.add(e);
                            iter++;

                            uniteSubsets(subsets, set1, set2);
                              numTree--;
```

```
                        }
                    }
                    }
                }
                }
            }
            textField.setText("Final weight of MST :" + MSTweight);
            return sortEdges;
}
        //Method to find the set of a vert
        //It utilizes path compression technique
        private int find(Subset[] subsets, int vert) {
            if (subsets[vert].getParent() != vert) {
                subsets[vert].setParent(find(subsets,
subsets[vert].getParent()));
            }
            return subsets[vert].getParent();
        }

        //Method to unite subsets, it uses the rank to select the parent
        private void uniteSubsets(Subset[] subsets, int v1, int v2){

            int rootv1 = find(subsets, v1);
            int rootv2 = find(subsets, v2);

            if(subsets[v1].getRank() < subsets[v2].getRank()){
                subsets[v1].setParent(subsets[v2].getParent());
            }else if(subsets[v1].getRank() > subsets[v2].getRank()){
                subsets[v2].setParent(subsets[v1].getParent());
            }else{
                subsets[v2].setParent(subsets[v1].getParent());
                subsets[v1].setRank(subsets[v1].getRank() + 1);
            }

        }
}
```