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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы поиска пути»

Тема: Жадный алгоритм и A*.

Студент гр. 8304 _____ Сергеев А.Д.

Преподаватель _____ Размочаева Н.А.

Санкт-Петербург

2019

Цель работы.

Научиться использовать жадный алгоритм и алгоритм A* для поиска пути в ориентированном графе.

Задание.

Для жадного алгоритма:

Жадность в данном случае понимается следующим образом: на каждом шаге выбирается последняя посещённая вершина. Переместиться необходимо в ту вершину, путь до которой является самым дешёвым из последней посещённой вершины. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес.

Для алгоритма A*:

Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес. В качестве эвристической функции следует взять близость символов, обозначающих вершины графа, в таблице ASCII.

Порядок выполнения работы.

Написание работы производилось на базе операционной системы Windows 10 на языке программирования java в среде программирования IntelliJ IDEA.

Было решено представить ориентированный граф в виде словаря, ключами которого были бы буквы, обозначающие вершины графа, а значениями — словари, ключами которых были бы буквенные обозначения вершин, в которые существуют рёбра из данной, а значениями — вес пути до этих вершин.

В соответствии с требованиями, изложенными в задании, также было принято решение хранить путь в виде строки, представляющей из себя последовательность вершин и целого числа, обозначающего общий вес пути.

Описание классов в UML-виде приложено к отчеты в файле UML.png.

Вывод.

В результате лабораторной работы были получены знания о жадном алгоритме и об алгоритме A*.

ПРИЛОЖЕНИЯ

Приложение А

Исходный код программы, файл Main.java

```
package com.company;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        System.out.print("Press Y to check out greedy
pathfinder, Z to check out A*: ");
        Scanner sc = new Scanner(System.in);

        char alg = sc.next().charAt(0);
        Pathfinder PF;

        if (alg == 'Y') PF = new Greedy();
        else if (alg == 'Z')PF = new AStar();
        else {
            System.out.println("Wrong letter, sorry :/");
            return;
        }

        String ans = PF.solve();
        if (ans != null) System.out.println(ans);
        else System.out.println("There's no path available!");
    }
}
```

Приложение Б

Исходный код программы, файл PathFinder.java

```
package com.company;

import java.util.HashMap;
import java.util.Scanner;

public abstract class Pathfinder {
     HashMap<Character, HashMap<Character, Double>> nodes = new
HashMap<>();

    public String solve() {
        Scanner sc = new Scanner(System.in);
        char first = sc.next().charAt(0);
        char last = sc.next().charAt(0);

        int len = (int) last - (int) first;
        nodes = new HashMap<>(len);
        for (char i = 'a'; i <= 'z'; i++) {
            nodes.put(i, new HashMap<>());
        }

        char source;
        char target;
        double weight;
        while (sc.hasNextLine()) {
            source = sc.next().charAt(0);
            target = sc.next().charAt(0);
            weight = Double.parseDouble(sc.next());
            nodes.get(source).put(target, weight);
        }
```

```
            sc.close();

            Path shortest = find(first, last);
               return (shortest != null) ? shortest.getLiteral() :
null;
        }

        protected abstract Path find(char first, char last);



        public static class Path {
            private String literal;
            private double length;

            public Path(String literal, double length) {
                this.literal = literal;
                this.length = length;
            }

            public Path(char literal, int length) {
                this.literal = "";
                this.literal += literal;
                this.length = length;
            }

            public Path addFront(char node, double length) {
                StringBuilder sb = new StringBuilder(this.literal);
                sb.insert(0, node);
                this.literal = sb.toString();
                this.length += length;
                return this;
```

```
        }

        public Path addBack(char node, double length) {
            this.literal += node;
            this.length += length;
            return this;
        }

        public String getLiteral() {
            return literal;
        }

        public double getLength() {
            return length;
        }

        public char getEnd() {
            return literal.charAt(literal.length() - 1);
        }
    }
}
```

Приложение В

Исходный код программы, файл Greedy.java

```
package com.company;

import java.util.*;

public class Greedy extends Pathfinder {
    @Override
    protected Path find(char start, char end) {
        if (start == end) return new Path(start, 0);
        if (nodes.get(start).isEmpty()) return null;

                                        double    shortestLength    =
Collections.min(nodes.get(start).values());
        LinkedList<Path> paths = new LinkedList<>();
                    for   (Map.Entry<Character,   Double>   map   :
nodes.get(start).entrySet()) {
            if (map.getValue() == shortestLength) {
                Path path = find(map.getKey(), end);
                if (path != null) {
                                    paths.add(path.addFront(start,
shortestLength));
                }
            }
        }

        if (paths.isEmpty()) return null;

        Path SP = paths.peek();
        double shortestPath = SP.getLength();
```

```
                for (Path path : paths) if (path.getLength() <
shortestPath) {
                    shortestPath = path.getLength();
                    SP = path;
                }

                return SP;
            }
        }
```

Приложение Г

Исходный код программы, файл Square.java

```
package com.company;

import java.util.LinkedList;
import java.util.Map;
import java.util.TreeMap;

public class AStar extends Pathfinder {
    private char first, last;

    private double g(Path path) {
        return path.getLength();
    }

    private double h(Path path) {
        return Math.abs((int) last - (int) path.getEnd());
    }

    private double f(Path path) {
        return g(path) + h(path);
    }

    @Override
    public Path find(char first, char last) {
        this.first = first;
        this.last = last;

        LinkedList<Path> closed = new LinkedList<>();
        TreeMap<Double, Path> opened = new TreeMap<>();
```

```
            Path beginning = new Path(first, 0);
            opened.put(f(beginning), beginning);

            while (!opened.isEmpty()) {
                                Map.Entry<Double,  Path>  current  =
opened.firstEntry();

                    if (current.getValue().getEnd() == last) return
current.getValue();

                opened.remove(current.getKey());
                closed.push(current.getValue());

                            for (Map.Entry<Character, Double> near:
nodes.get(current.getValue().getEnd()).entrySet()) {
                                                Path  vertex  =  new
Path(current.getValue().getLiteral(), current.getValue().getLength())
                                                .addBack(near.getKey(),
near.getValue());

                    if (!contains(closed, vertex))
                        if (contains(opened, vertex)) {
                               double prevDist = getDistance(opened,
vertex);
                            if (prevDist > f(vertex)) {
                                opened.remove(prevDist);
                                opened.put(f(vertex), vertex);
                            }
                        } else {
                            opened.put(f(vertex), vertex);
                        }
                }
```

```
            }

            return null;
        }

        private boolean contains(LinkedList<Path> paths, Path path)
{
            for (Path p : paths) {
                if (p.getEnd() == path.getEnd()) return true;
            }
            return false;
        }

        private boolean contains(TreeMap<Double, Path> paths, Path
path) {
            for (Path p : paths.values()) {
                if (p.getEnd() == path.getEnd()) return true;
            }
            return false;
        }

        private double getDistance(TreeMap<Double, Path> paths, Path
path) {
            for (Map.Entry<Double, Path> p : paths.entrySet()) {
                  if (p.getValue().getEnd() == path.getEnd()) return
p.getKey();
            }
            return -1;
        }
    }
```