**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.В. |

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.

Дополнительное задание (вариант 9): вывод графического представления графа.

**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.

Был создан абстрактный класс *PathFinder*, содержащий в себе метод *solve*, который считывает описание графа из входного потока и ищет в нём кратчайший путь при помощи абстрактного метода *find*, а также вложенный класс *Path*, представляющий из себя описание пути в графе.

Два класса наследника *PathFinder*, *Greedy* и *AStar*, переопределяют метод find для нахождения кратчайшего пути в графе жадным алгоритмом и алгоритмом A\* соответственно.

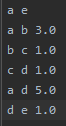
Класс *Visualizer* визуализирует граф, отмечая в нём найденный кратчайший путь.

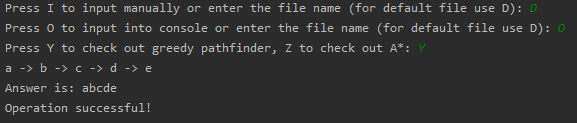
Класс *Filer* осуществляет связь с файловой системой, открывает потоки ввода и вывода в файл и записывает в файл изображение графа.

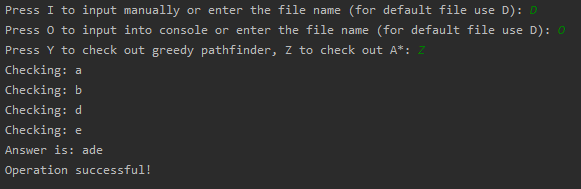
Описание классов в UML-виде приложено к отчеты в файле UML.png.

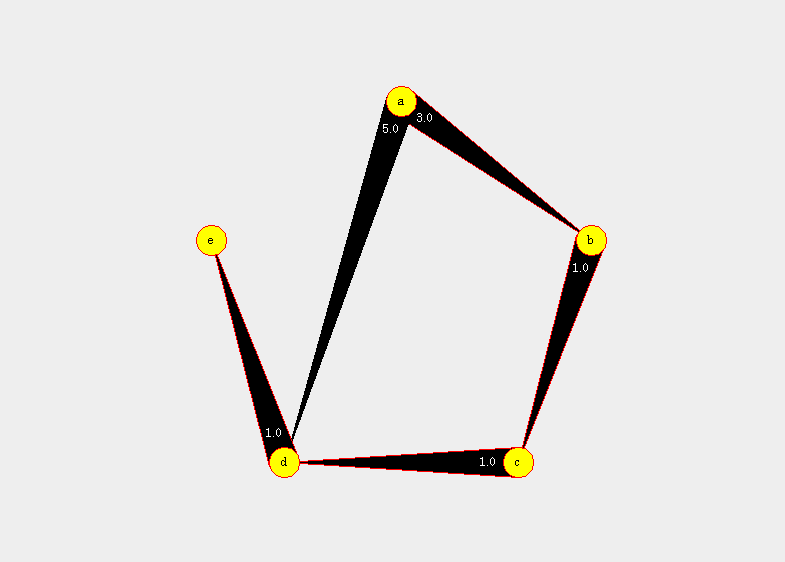
**Тестирование.**

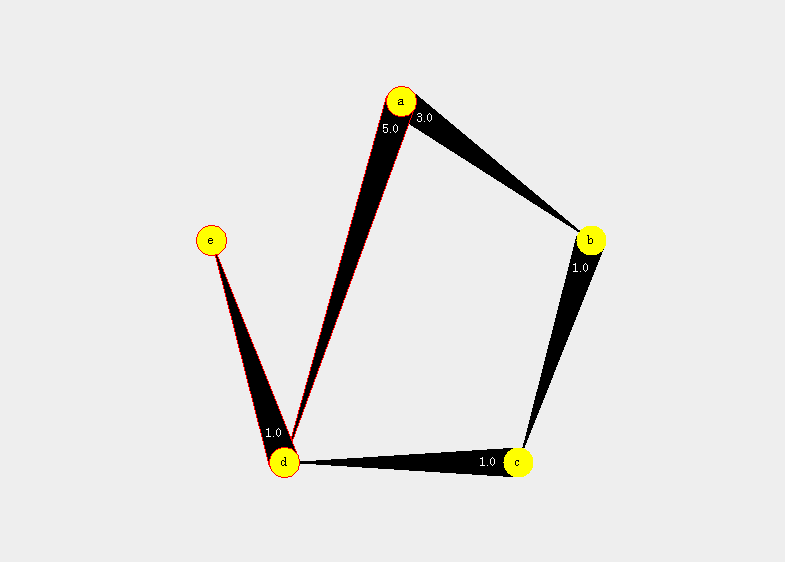
Для одного и того же набора входных данных (предоставленного на сайте stpik.org) было проведено тестирование алгоритма нахождения пути жадным алгоритмом и алгоритмом A\*.

Рисунок 1 – Входные данные.

Рисунок 2 – Вывод программы для жадного алгоритма.

Рисунок 3 – Вывод программы для алгоритма A\*.

Рисунок 4 – Визуализация графа для жадного алгоритма.

Рисунок 5 – Визуализация графа для алгоритма A\*.

**Вывод.**

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

**Приложение А**

**Исходный код программы, файл Main.java**

package com.company;

import java.io.PrintStream;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

System.out.print("Press I to input manually or enter the file name (for default file use D): ");

Scanner sc = new Scanner(System.in);

String inp = sc.next();

Scanner src;

if (inp.charAt(0) == 'D') {

src = new Scanner(Filer.readFromFile(Filer.inputFile));

} else if (inp.charAt(0) == 'I') {

src = sc;

} else {

src = new Scanner(Filer.readFromFile(inp));

}

System.out.print("Press O to input into console or enter the file name (for default file use D): ");

inp = sc.next();

PrintStream ps;

if (inp.charAt(0) == 'D') {

ps = Filer.writeToFile(Filer.outputFile);

} else if (inp.charAt(0) == 'O') {

ps = System.out;

} else {

ps = Filer.writeToFile(inp);

}

System.out.print("Press Y to check out greedy pathfinder, Z to check out A\*: ");

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(src, ps);

ps.println();

if (ans != null) ps.println("Answer is: " + ans);

else ps.println("There's no path available!");

ps.flush();

System.out.println("Operation successful!");

}

}

**Приложение Б**

**Исходный код программы, файл PathFinder.java**

package com.company;

import java.io.PrintStream;

import java.util.HashMap;

import java.util.Scanner;

public abstract class Pathfinder {

HashMap<Character, HashMap<Character, Double>> nodes = new HashMap<>();

PrintStream out;

public String solve(Scanner sc, PrintStream ps) {

out = ps;

char first = sc.next().charAt(0);

char last = sc.next().charAt(0);

int len = (int) last - (int) first;

nodes = new HashMap<>(len);

char source;

char target;

double weight;

while (sc.hasNextLine()) {

source = sc.next().charAt(0);

target = sc.next().charAt(0);

if (!nodes.containsKey(source)) nodes.put(source, new HashMap<>());

if (!nodes.containsKey(target)) nodes.put(target, new HashMap<>());

weight = Double.parseDouble(sc.next());

nodes.get(source).put(target, weight);

}

sc.close();

Path shortest = find(first, last);

Visualizer vis = new Visualizer();

vis.draw(nodes, shortest.getLiteral());

vis.print();

return shortest.getLiteral();

}

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) {

out.print(start);

if (start == end) return new Path(start, 0);

else out.print(" -> ");

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 AStar.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);

out.print("Checking: " + beginning.getEnd());

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)) {

out.print("\nChecking: " + vertex.getEnd());

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;

}

}

**Приложение Д**

**Исходный код программы, файл Visualizer.java**

package com.company;

import javax.swing.\*;

import java.awt.\*;

import java.awt.geom.Ellipse2D;

import java.awt.geom.Rectangle2D;

import java.awt.image.BufferedImage;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

public class Visualizer {

GraphicPanel gp;

void draw(HashMap<Character, HashMap<Character, Double>> nodes, String path) {

JFrame root = new JFrame("Plot");

root.setMinimumSize(new Dimension(801, 601));

root.getContentPane().setLayout(new BorderLayout());

root.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

gp = new GraphicPanel(nodes);

gp.solvation = path;

root.getContentPane().add(gp, "Center");

root.setVisible(true);

root.pack();

}

public void print() {

Filer.imageOut(gp.print());

}

static class GraphicPanel extends JPanel {

HashMap<Character, HashMap<Character, Double>> nodes;

ArrayList<Character> literals;

String solvation;

public GraphicPanel(HashMap<Character, HashMap<Character, Double>> nodes) {

this.nodes = nodes;

this.literals = new ArrayList<>(nodes.keySet());

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setFont(new Font("Century Schoolbook", Font.PLAIN, 12));

if ((nodes != null) && (nodes.size() > 0)) {

for (char letter : literals) {

for (Map.Entry<Character, Double> line : nodes.get(letter).entrySet()) {

putLine(letter, line.getKey(), line.getValue(), (Graphics2D) g);

}

}

for (char letter : literals) putNode(letter, (Graphics2D) g);

}

}

private void putNode(char identifier, Graphics2D graphics) {

Point pos = findPos(literals.indexOf(identifier), literals.size());

graphics.setPaint(Color.yellow);

graphics.fill(new Ellipse2D.Double(pos.x - 15, pos.y - 15, 30, 30));

if (solvation.contains(String.valueOf(identifier))) {

graphics.setPaint(Color.red);

graphics.drawOval(pos.x - 15, pos.y - 15, 30, 30);

}

graphics.setPaint(Color.black);

drawCenteredString(graphics, String.valueOf(identifier), pos.x - 15, pos.y - 15, 30, 30);

}

private void putLine(char second, char first, double weight, Graphics2D graphics) {

Point posFirst = findPos(literals.indexOf(first), literals.size());

Point posSecond = findPos(literals.indexOf(second), literals.size());

Point perpFirst;

Point perpSecond;

Point center;

if ((posSecond.x != posFirst.x) && (posSecond.y != posFirst.y)) {

double a = 1.0 / (posSecond.x - posFirst.x);

double b = -1.0 / (posSecond.y - posFirst.y);

double mult = Math.sqrt(30 \* 30 / (a \* a + b \* b));

Point sameFirst = new Point((int) (posSecond.x + b \* mult), (int) (posSecond.y - a \* mult));

Point sameSecond = new Point((int) (posSecond.x - b \* mult), (int) (posSecond.y + a \* mult));

Rectangle2D rect = new Rectangle();

rect.setFrameFromDiagonal(posFirst, posSecond);

if (rect.contains(sameFirst)) center = sameFirst;

else center = sameSecond;

double aP = -b;

double bP = a;

double multP = Math.sqrt(15 \* 15 / (aP \* aP + bP \* bP));

perpFirst = new Point((int) (posSecond.x + bP \* multP), (int) (posSecond.y - aP \* multP));

perpSecond = new Point((int) (posSecond.x - bP \* multP), (int) (posSecond.y + aP \* multP));

} else if (posSecond.x == posFirst.x) {

perpFirst = new Point(posSecond.x + 15, posSecond.y);

perpSecond = new Point(posSecond.x - 15, posSecond.y);

center = new Point(posSecond.x, posSecond.y + 30);

if (Math.abs(posFirst.y - center.y) > Math.abs(posFirst.y - posSecond.y)) center = new Point(posSecond.x, posSecond.y - 30);

} else {

perpFirst = new Point(posSecond.x, posSecond.y + 15);

perpSecond = new Point(posSecond.x, posSecond.y - 15);

center = new Point(posSecond.x + 30, posSecond.y);

if (Math.abs(posFirst.x - center.x) > Math.abs(posFirst.x - posSecond.x)) center = new Point(posSecond.x - 30, posSecond.y);

}

graphics.setPaint(Color.black);

graphics.fillPolygon(new int[] {posFirst.x, perpFirst.x, perpSecond.x}, new int[] {posFirst.y, perpFirst.y, perpSecond.y}, 3);

if (solvation.contains(String.valueOf(new char[] {second, first}))) {

graphics.setPaint(Color.red);

graphics.drawPolygon(new int[] {posFirst.x, perpFirst.x, perpSecond.x}, new int[] {posFirst.y, perpFirst.y, perpSecond.y}, 3);

}

graphics.setPaint(Color.white);

drawCenteredString(graphics, String.valueOf(weight), center.x - 15, center.y - 15, 30, 30);

}

public void drawCenteredString(Graphics2D g, String string, int x, int y, int width, int height) {

FontMetrics metrics = g.getFontMetrics(g.getFont());

x += (width - metrics.stringWidth(string)) / 2;

y += ((height - metrics.getHeight()) / 2) + metrics.getAscent();

g.drawString(string, x, y);

}

private Point findPos(int elementNumber, int total) {

double ang = 360.0 / total \* elementNumber;

double trueAng = Math.toRadians(90) - Math.toRadians(ang);

int x = 401 + (int) (Math.cos(trueAng) \* 200);

int y = 301 - (int) (Math.sin(trueAng) \* 200);

return new Point(x, y);

}

private BufferedImage print() {

BufferedImage bi = new BufferedImage(this.getSize().width, this.getSize().height, BufferedImage.TYPE\_INT\_ARGB);

Graphics g = bi.createGraphics();

this.paint(g);

return bi;

}

}

}

**Приложение Е**

**Исходный код программы, файл Filer.java**

package com.company;

import javax.imageio.ImageIO;

import java.awt.image.BufferedImage;

import java.io.\*;

import java.nio.file.Paths;

public class Filer {

private static final String afterPath = "\\src\\com\\";

public static final String inputFile = "input.txt";

public static final String outputFile = "output.txt";

public static final String imageFile = "graph.png";

public static InputStream readFromFile(String fileName) {

String absolute = Paths.get("").toAbsolutePath().toString() + afterPath + fileName;

FileInputStream fist;

try {

fist = new FileInputStream(absolute);

} catch (FileNotFoundException ffe) {

fist = null;

}

return fist;

}

public static PrintStream writeToFile(String fileName) {

String absolute = Paths.get("").toAbsolutePath().toString() + afterPath + fileName;

PrintStream fist;

try {

fist = new PrintStream(absolute);

} catch (FileNotFoundException ffe) {

fist = null;

}

return fist;

}

public static void imageOut(BufferedImage bi) {

try{

ImageIO.write(bi, "png", new File(Paths.get("").toAbsolutePath().toString() + afterPath + imageFile));

}catch (Exception e) {

System.out.println("Can not save image...");

}

}

}