****

**Московский авиационный институт**

**(Национальный исследовательский университет)**

**Институт № 3**

**Кафедра 311**

**Интеллектуальные системы**

**Лабораторная работа № 8**

**«Механизмы возврата и нечеткая логика.   
Элементы нечеткой логики»**

**Выполнил студент  
Веденеев Максим Кириллович**

**Группа М3З-501-БК**

**Дата 06.10.2023 г.**

**Принял преподаватель  
Кос Оксана Игоревна**

Оглавление

[Цель лабораторной работы 3](#__RefHeading___1)

[Глава 1. Реализация 4](#__RefHeading___2)

[Итоги лабораторной работы 8](#__RefHeading___3)

[Список литературы 9](#__RefHeading___4)

# Цель лабораторной работы

Изучить:

1. Механизмы возврата и нечеткую логику.
2. Реализовать его.

# Глава 1. Реализация

package eight;  
import java.awt.Point;  
import java.nio.file.Path;  
import java.util.\*;  
  
import static java.util.Objects.nonNull;  
  
public class MazePathFinder {  
 private Maze maze;  
 private Point source;  
 private Point target;  
 private boolean[][] visited;  
 private Map<Point, Point> parents;  
  
 public MazePathFinder() {  
  
 }  
 private MazePathFinder(final Maze maze,  
 final Point source,  
 final Point target) {  
 Objects.requireNonNull(maze, "Input maze are empty.");  
 Objects.requireNonNull(source, "Source point is null.");  
 Objects.requireNonNull(target, "Target point is null.");  
  
 this.maze = maze;  
 this.source = source;  
 this.target = target;  
  
 checkSourceNode();  
 checkTargetNode();  
  
 this.visited = new boolean[maze.getHeight()][maze.getWidth()];  
 this.parents = new HashMap<>();  
 this.parents.put(source, null);  
 }  
  
 public List<Point> findPath(final Maze maze,  
 final Point source,  
 final Point target) {  
 return new MazePathFinder(maze, source, target)  
 .compute();  
 }  
  
 private List<Point> compute() {  
 final Queue<Point> queue = new ArrayDeque<>();  
 final Map<Integer, List<Point>> distances = new TreeMap<>();  
  
 queue.add(source);  
  
 while (!queue.isEmpty()) {  
 final Point current = queue.remove();  
  
 if (current.equals(target)) {  
 List<Point> path = constructPath();  
 distances.put(path.size(), path);  
  
 return distances.values().stream()  
 .filter(Objects::nonNull)  
 .findAny()  
 .orElseThrow(() -> new RuntimeException("Can't find path"));  
 }  
  
 for (final Point child : generateChildren(current)) {  
 if (!parents.containsKey(child)) {  
 parents.put(child, current);  
 // Добавляет «дочерний элемент» в конец этой очереди.  
 queue.add(child);  
 }  
 }  
 }  
  
 // null означает, что Targer point недоступен из исходного узла.  
 return null;  
 }  
  
 private List<Point> constructPath() {  
 Point current = target;  
 final List<Point> path = new ArrayList<>();  
  
 while (current != null) {  
 path.add(current);  
 current = parents.get(current);  
 }  
  
 Collections.<Point>reverse(path);  
 return path;  
 }  
  
 private Iterable<Point> generateChildren(final Point current) {  
 final Point north = new Point(current.x, current.y - 1);  
 final Point south = new Point(current.x, current.y + 1);  
 final Point west = new Point(current.x - 1, current.y);  
 final Point east = new Point(current.x + 1, current.y);  
  
 final List<Point> childList = new ArrayList<>(4);  
  
 if (maze.cellIsTraversible(north)) {  
 childList.add(north);  
 }  
  
 if (maze.cellIsTraversible(south)) {  
 childList.add(south);  
 }  
  
 if (maze.cellIsTraversible(west)) {  
 childList.add(west);  
 }  
  
 if (maze.cellIsTraversible(east)) {  
 childList.add(east);  
 }  
  
 return childList;  
 }  
  
 private void checkSourceNode() {  
 checkNode(source,  
 "Source point (" + source + ") are out of maze. " +  
 "Maze width " + maze.getWidth() + " and " +  
 "maze height " + maze.getHeight() + ".");  
  
 if (!maze.cellIsFree(source.x, source.y)) {  
 throw new IllegalArgumentException(  
 "Source point (" + source + ") is not in free cell.");  
 }  
 }  
  
 private void checkTargetNode() {  
 checkNode(target,  
 "Targer point (" + target + ") are out of maze. " +  
 "Maze width " + maze.getWidth() + " and " +  
 "maze height " + maze.getHeight() + ".");  
  
 if (!maze.cellIsFree(target.x, target.y)) {  
 throw new IllegalArgumentException(  
 "Target point (" + target + ") is not in free cell.");  
 }  
 }  
  
 private void checkNode(final Point node, final String errorMessage) {  
 if (node.x < 0  
 || node.x >= maze.getWidth()  
 || node.y < 0  
 || node.y >= maze.getHeight()) {  
 throw new IllegalArgumentException(errorMessage);  
 }  
 }  
  
 public static void main(String[] args) {  
 int[][] mazePlan = {  
 {0,1,0,0,0,0,0,0},  
 {0,0,0,1,0,0,0,0},  
 {1,0,0,0,0,0,0,0},  
 {1,1,0,0,0,0,0,0},  
 {0,0,0,1,0,1,0,0},  
 {0,0,0,1,0,1,0,0}  
 };  
  
 boolean[][] maze2 = new boolean[mazePlan.length][mazePlan[0].length];  
  
 for (int i = 0; i < maze2.length; ++i) {  
 for (int j = 0; j < maze2[i].length; ++j) {  
 maze2[i][j] = mazePlan[i][j] > 0;  
 }  
 }  
  
 final Maze maze = new Maze(maze2);  
 final Point source = new Point(2,0); // Same as new Point(0, 0):  
 final Point target = new Point(7, 5);  
 final Point secondTarget = new Point(0, 5);  
  
  
 long startTime = System.nanoTime();  
 final List<Point> path = new MazePathFinder().findPath(maze,  
 source,  
 target);  
 final List<Point> secondPath = new MazePathFinder().findPath(maze,  
 source,  
 secondTarget);  
 long endTime = System.nanoTime();  
  
 List<Point> shortestPath = path.size() < secondPath.size()  
 ? path  
 : secondPath;  
  
 System.out.printf("Search maze BFS in %d milliseconds.\n",  
 (endTime - startTime) / 1\_000\_000L);  
  
 System.out.printf("Path to point [x='%d', y='%d']: %d steps%n", target.x, target.y, path.size() - 1);  
 System.out.println(maze.withPath(path));  
  
 System.out.printf("Path to point [x='%d', y='%d']: %d steps%n", secondTarget.x, secondTarget.y, secondPath.size() - 1);  
 System.out.println(maze.withPath(secondPath));  
  
 System.out.println();  
 System.out.println("Shortest path: " + (shortestPath.size() - 1));  
 System.out.println(maze.withPath(shortestPath));  
 }  
  
}

Результат компиляции данного кода представлен на Рисунке 2.

Лабиринт содержит два выхода, кратчайший путь находится путём перебора всех возможных.

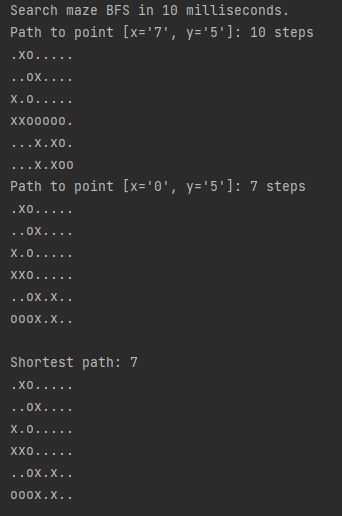


Рисунок 2. Результат выполнения программы

# Итоги лабораторной работы

Мы научились писать код для поиска кратчайшего пути в лабиринте.

# Список литературы

1. ГОСТ Р 7.0.11-2011. Система стандартов по информации, библиотечному и издательскому делу. Диссертация и автореферат диссертации. Структура и правила оформления. - 19 c.
2. https://lisiynos.github.io/s1/graph\_alg.html