

Санкт-Петербургский политехнический университет Петра Великого  
Институт компьютерных наук и технологий  
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

## **КУРСОВОЙ ПРОЕКТ**

«Решатель лабиринта»

по дисциплине «Алгоритмы и структуры данных»

Выполнил студент гр. 3530901/00002 \_\_\_\_\_ Чешев А. Д.

Руководитель \_\_\_\_\_ Степанов Д. С.

«23» мая 2022

г. Санкт-Петербург

2022

**ЗАДАНИЕ**  
**НА ВЫПОЛНЕНИЕ КУРСОВОГО ПРОЕКТА**

студенту группы 3530901/00002 Чешеву Андрею Дмитриевичу

**1. Тема проекта:** Реализация решателя лабиринта на языке Kotlin

**2. Срок сдачи законченного проекта:** 23.05.2022

**3. Исходные данные к проекту:** IDE: IntelliJ IDEA Community Edition 2020.1, JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o, Version Java: 15

**4. Содержание пояснительной записки** (перечень подлежащих разработке вопросов): введение, основная часть (текст программы, описание программы, испытания программы), заключение, список использованных источников.

**Дата получения задания:** « 5 » апреля 2022 г.

Руководитель \_\_\_\_\_ Д. С. Степанов

(подпись)

(инициалы, фамилия)

Задание принял к исполнению \_\_\_\_\_ А. Д. Чешев

(подпись студента)

(инициалы, фамилия)

\_\_\_\_\_  
(дата)

## Исходные данные к работе

Задание - решатель Лабиринта

Реализовать генерацию лабиринта размером (N x M) и решатель к нему. Сравнить временные характеристики работы выбранных алгоритмов решения.

## Программная реализация

<https://github.com/Queenore/terra-incognita-game>

## Описание программы

### Директория core

class Maze – имеет поля height: Int, width: Int, matrix: Matrix.

fun newRandomMaze()

Генерирует лабиринт размером (N x M).

fun printMaze()

Выводит в консоль графическое представление лабиринта.

fun printTrace(path: MutableList<Pair<Int, Int>>)

Изменяет пустые клетки лабиринта на «пройденные», соответствующие элементам path.

fun deepCopy(): Maze

Возвращает «глубокую копию» сгенерированного лабиринта.

fun getTwoRandCoords()

Возвращает две случайные координаты, соответствующие пустым полям в лабиринте.

```
fun dispelMaze(prob: Double)
```

Позволяет «развеять» лабиринт, т.е. удалить каждую стенку с вероятностью prob.

```
fun graphFromMaze(edgeWeight: Double): Graph
```

Возвращает граф, соответствующий сгенерированному лабиринту.

```
class Matrix
```

```
fun get(row: Int, column: Int): Cell
```

```
fun set(row: Int, column: Int, cell: Cell)
```

```
fun twoRandCoords(): TwoCoords
```

Возвращает две случайные координаты, соответствующие пустым полям (Cell.Empty) матрицы (лабиринта).

```
override fun toString(): String
```

```
class Graph
```

```
fun addVertex()
```

Добавляет вершину в граф.

```
fun changeWeight()
```

Меняет вес ребра графа.

```
fun changeAllWeights (number: Double, oper: Operations)
```

Реализует арифметическую операцию (oper) с числом number на все ребра графа.

```
fun connect()
```

Устанавливает связь между двумя вершинами графа.

```
enum class Cell {
```

```
    Empty,
```

```
    Wall,
```

```
    Trace,
```

```
}
```

```
enum class Operations {
```

```
    Plus,
```

```
    Minus,
```

```
    Times,
```

```
    Div
```

```
}
```

```
class TwoCoords
```

Два поля – две координаты класса Pair.

## Директория **algorithms**

```
interface Shortest Path
```

Интерфейс, который должны реализовывать алгоритмы нахождения кратчайшего пути.

```
fun getPathLength(): Int
```

Должна возвращать длину кратчайшего пути.

```
fun getExecutionTime(): Long
```

Должна возвращать время работы функции solve().

`fun printSolution()`

Вывод в консоль графического представления кратчайшего пути.

`fun solve()`

Реализация нахождения кратчайшего пути.

`class BFS`

Нахождение кратчайшего пути между двумя вершинами графа алгоритмом BFS.  
Реализует интерфейс ShortestPath.

`class A*`

Нахождение кратчайшего пути между двумя вершинами графа алгоритмом A\*.  
Реализует интерфейс ShortestPath.

`class AntAlg`

Нахождение кратчайшего пути между двумя вершинами графа муравьиным алгоритмом. Реализует интерфейс ShortestPath.

## Испытания работы программы

```
maze size: 25 25
start: (21, 1)
finish: (6, 19)
```

Рис. 1. Исходные данные (размер лабиринта и точки, между которыми нужно найти кратчайший путь).

На рис. 2–6 в графическом виде представлены результаты работы BFS, A\* и муравьиного алгоритмов.

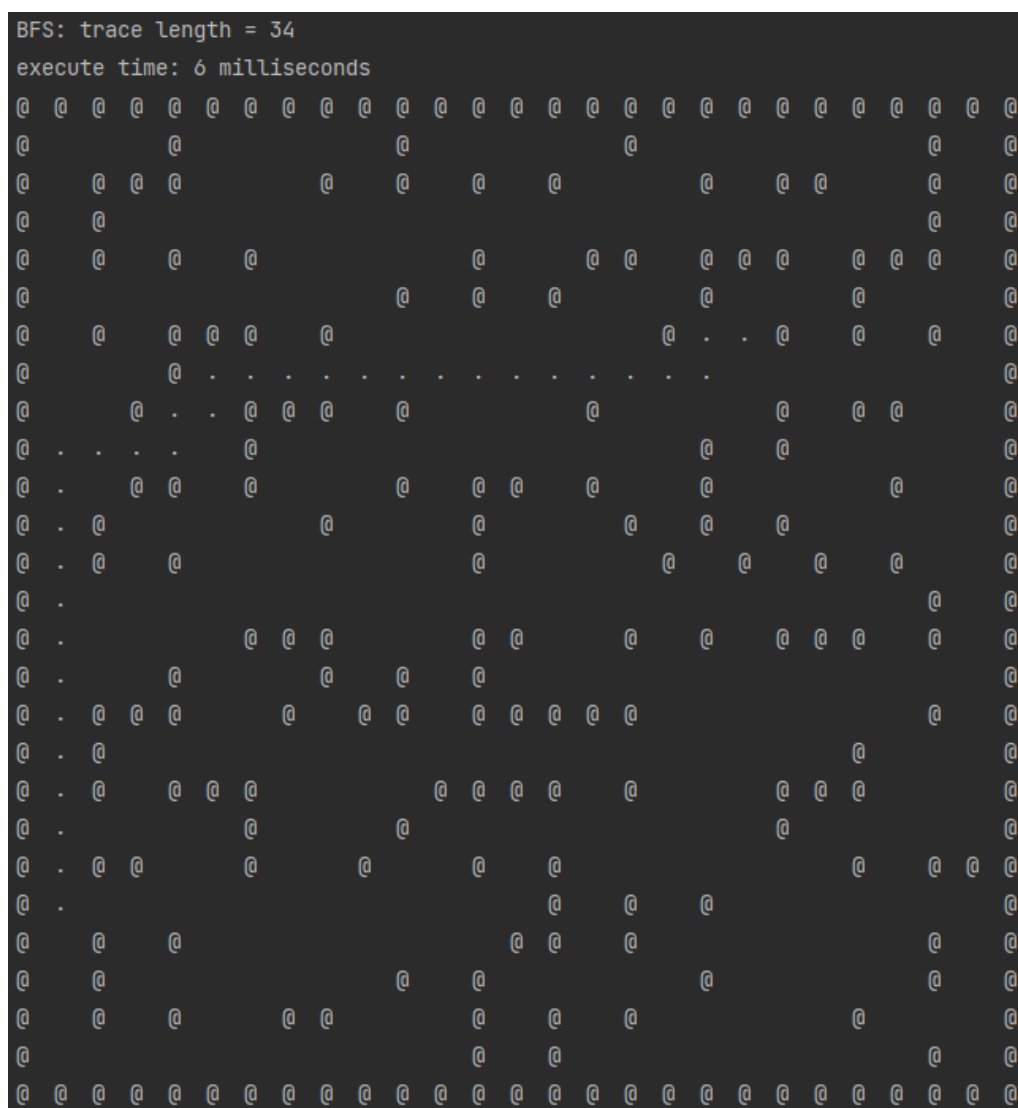


Рис. 2. Результаты работы алгоритма BFS (кратчайший путь).

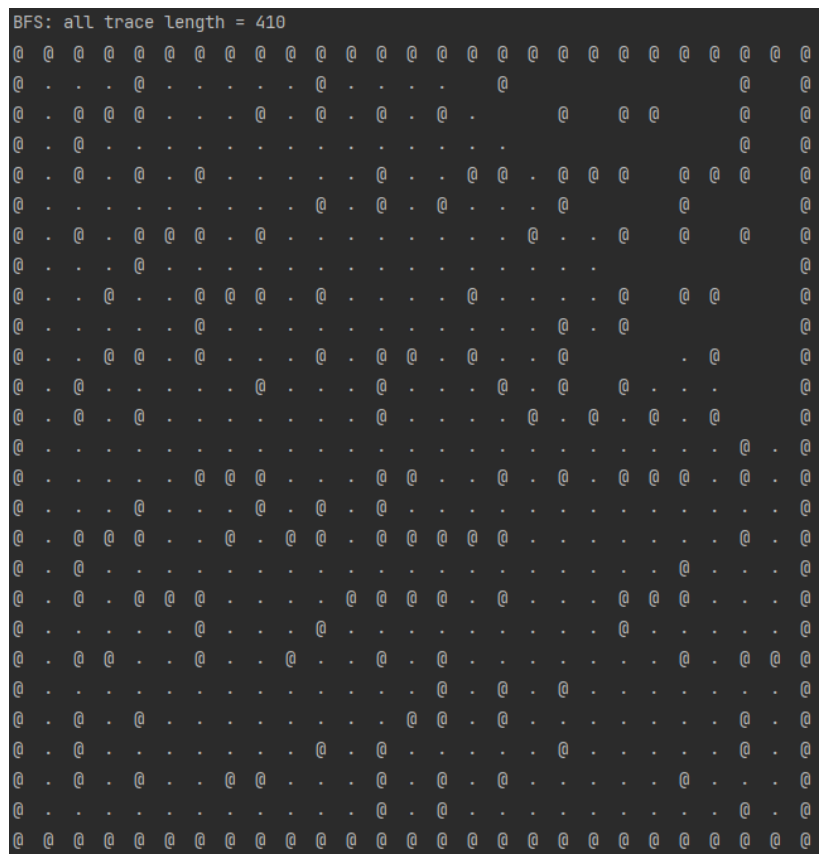


Рис. 3. Результаты работы алгоритма BFS (весь пройденный путь).

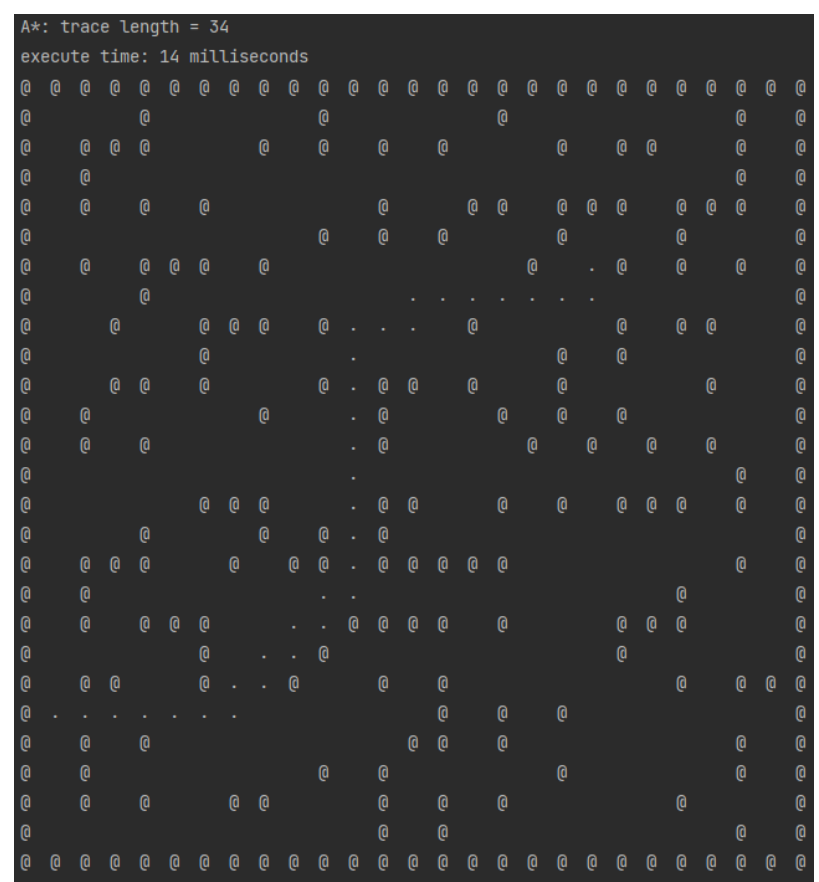


Рис. 4. Результаты работы алгоритма A\* (кратчайший путь).



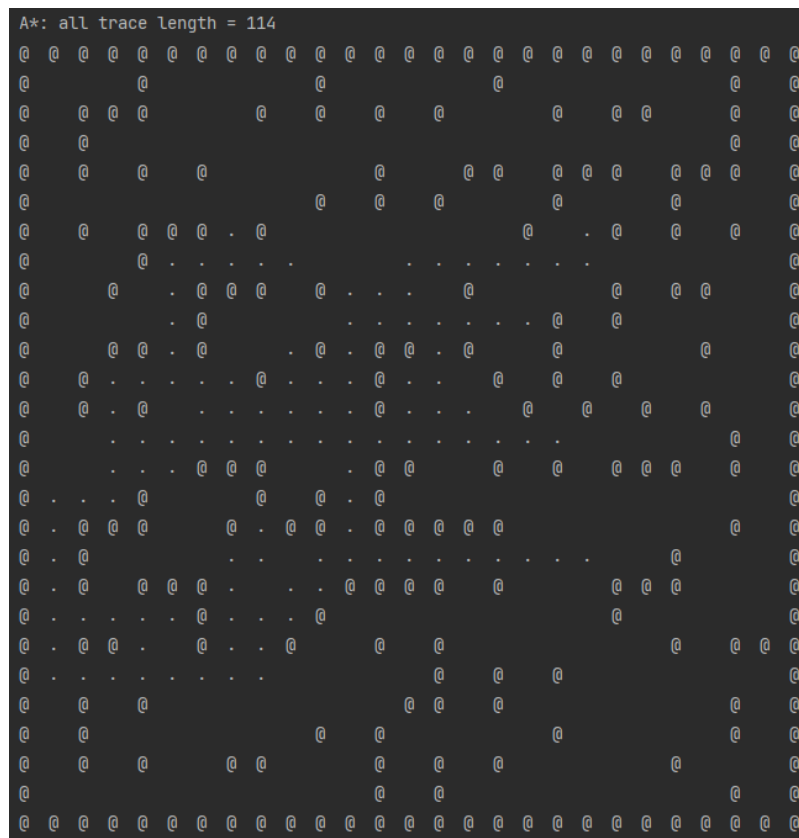


Рис. 5. Результаты работы алгоритма A\* (весь пройденный путь).

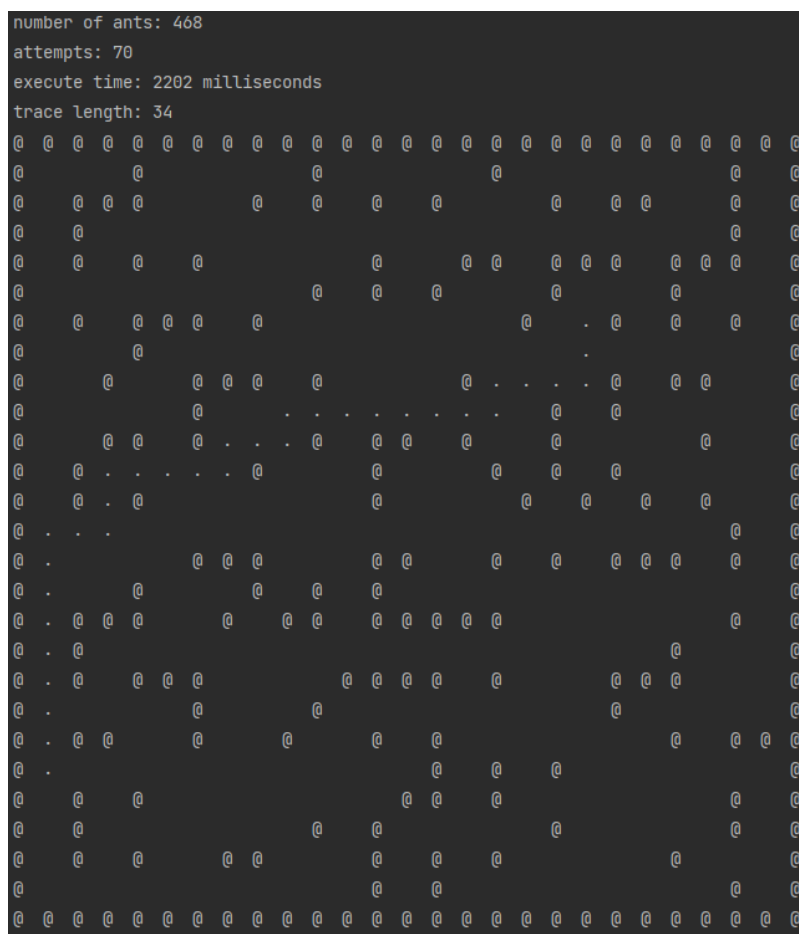


Рис. 6. Результаты работы муравьиного алгоритма (кратчайший путь).

```
Information about the execution time
(50 iterations,
maze size for bfs and A* in range 11..151,
maze size for ant algorithm in range 7..41)
#####
average bfs execution time = 11.52 milliseconds

average A* execution time = 4.72 milliseconds

average ant algorithm execution time: 2230.32 milliseconds
  min ant algorithm execution time: 20 milliseconds
  max ant algorithm execution time: 8672 milliseconds
#####
```

Рис. 7. Результаты временного тестирования трех алгоритмов.

## Заключение

В ходе данной курсовой работе были исследованы три алгоритма (BFS, A\*, муравьиный алгоритм) нахождения кратчайшего пути в лабиринте, представленным в виде графа. Был сделан вывод о том, что наиболее эффективным алгоритмом по времени выполнения оказался A\*.

## **Список использованных источников**

1. [https://ru.wikipedia.org/wiki/Алгоритм\\_Ли](https://ru.wikipedia.org/wiki/Алгоритм_Ли)
2. [https://ru.wikipedia.org/wiki/A\\*](https://ru.wikipedia.org/wiki/A*)
3. <https://betacode.net/13613/java-priorityqueue>
4. [https://ru.wikipedia.org/wiki/Роевой\\_интеллект](https://ru.wikipedia.org/wiki/Роевой_интеллект)
5. <https://elib.spbstu.ru/dl/2/v17-1903.pdf/view>