

**Содержание**

[Реферат 5](#__RefHeading___1)

[Введение 6](#__RefHeading___2)

[1. Постановка задачи 7](#__RefHeading___3)

[3. Описание алгоритма программы 11](#__RefHeading___4)

[4. Описание программы 16](#__RefHeading___5)

[5. Тестирование 21](#__RefHeading___6)

[6. Ручной расчет задачи 24](#__RefHeading___7)

[Приложение A 28](#__RefHeading___8)

# **Реферат**

Отчет 23 стр, 14 рисунков.

ЛАБИРИНТ, АЛГОРИТМЫ ПОИСКА ПУТЕЙ.

Цель исследования – разработка программы, реализующей алгоритм нахождения путей в лабиринте.

Включены следующие аспекты: разработка программы для создания лабиринта, его визуализация, а также алгоритм поиска кратчайшего пути. Программа реализует волновой алгоритм для поиска пути и использует случайную генерацию лабиринта.

# **Введение**

Задачи, связанные с генерацией и поиском пути в лабиринте, широко применяются в различных областях, таких как робототехника, игры, транспортная логистика и исследования искусственного интеллекта. Лабиринт как структура данных предоставляет множество возможностей для тестирования и разработки алгоритмов. Он может быть представлен в виде двумерной сетки, где некоторые ячейки проходимы, а другие являются стенами.

Разработка алгоритмов генерации лабиринтов и поиска пути в них помогает изучать основные концепции алгоритмизации, такие как работа с графами, обход в ширину и глубину, а также эффективное использование ресурсов. В данной работе рассматривается метод генерации лабиринта с использованием случайного блуждания и алгоритм волнового поиска пути, обеспечивающий нахождение кратчайшего маршрута.

Цель данной работы заключается в создании программы, которая моделирует процесс генерации лабиринта и поиска пути в нем. Программа реализована на языке Python и предоставляет функционал визуализации, сохранения и загрузки лабиринтов.

# **1. Постановка задачи**

Разработать программу для генерации лабиринта размером введенным с клавиатуры и поиска кратчайшего пути из начальной точки (1,1) до конечной точки с учетом следующих требований:

1. Лабиринт должен быть случайно сгенерирован и визуализирован в текстовом виде.
2. Пользователь должен сам ввести размер лабиринта
3. Использовать алгоритм генерации лабиринта на основе случайного блуждания с проверкой тупиков.
4. Реализовать визуализацию пути от начала до конца с помощью символов "\*".
5. Использовать волновой алгоритм для нахождения кратчайшего пути в лабиринте.
6. Сохранить и загружать лабиринт из текстового файла для последующей работы.

**2. Теоретическая часть задания**

**Реализация алгоритма поиска путей в лабиринте с помощью волнового алгоритма**

Волновой алгоритм (или метод распространения волны) — это один из алгоритмов поиска пути, который применяется для нахождения кратчайшего пути в лабиринте или сетке с заданными препятствиями. Он основан на идее распространения волны от стартовой точки по всем возможным направлениям до тех пор, пока не будет достигнута целевая точка. Принцип работы алгоритма аналогичен распространению волн в жидкости, когда от начальной точки волна распространяется во все стороны, заполняя свободное пространство и вычисляя минимальное количество шагов для достижения каждой клетки.

**1. Основные шаги волнового алгоритма**

**1.1. Инициализация**

* В лабиринте обычно используется двумерная сетка, где каждая клетка может быть либо свободной (проходимой), либо занята препятствием.
* Стартовая точка и целевая точка заранее определяются.
* Все клетки лабиринта инициализируются значениями, которые будут хранить информацию о расстоянии от стартовой точки до текущей клетки. В начальный момент стартовая клетка имеет значение 0 (расстояние до себя), а остальные клетки — бесконечность или недостижимое значение.
* Клетки, занятые препятствиями, не могут быть посещены и остаются непройденными.

**1.2. Распространение волны**

* Алгоритм начинает распространяться от стартовой точки. Вокруг стартовой клетки волна охватывает все соседние клетки (в 4-х направлениях: вверх, вниз, влево, вправо), увеличивая расстояние на 1.
* Каждая клетка, которую посещает волна, помечается значением, соответствующим количеству шагов от начальной точки. Если соседняя клетка еще не была посещена, волна продолжает распространяться от этой клетки.
* Этот процесс повторяется до тех пор, пока волна не достигнет целевой точки или не окажется невозможным ее достичь.

**1.3. Завершение алгоритма**

* Когда волна достигает целевой точки, найден кратчайший путь.
* После этого можно восстановить путь от целевой точки к стартовой, двигаясь по клеткам, значения которых минимальны для каждой следующей клетки (то есть, выбирая клетку с минимальным значением, которая была посещена волной).

**2. Преимущества волнового алгоритма**

* **Простота реализации**: Волновой алгоритм достаточно прост в реализации и требует минимум памяти. Он не требует хранения информации о путях или дополнительных структур данных, как в случае с другими алгоритмами поиска, такими как A\* или алгоритм Дейкстры.
* **Нахождение кратчайшего пути**: Волновой алгоритм гарантированно найдет кратчайший путь в лабиринте, если такой путь существует.
* **Отсутствие эвристик**: В отличие от более сложных алгоритмов поиска, волновой алгоритм не использует эвристики, что делает его универсальным и подходящим для множества различных типов лабиринтов.

**3. Недостатки волнового алгоритма**

* **Низкая эффективность**: Алгоритм может быть медленным в случае больших лабиринтов, поскольку волна распространяется во все стороны и требует большого количества вычислений.
* **Проблемы с памятью**: Для очень больших лабиринтов требуется значительное количество памяти для хранения информации о клетках, особенно если лабиринт имеет большие размеры.

**4. Применение**

Волновой алгоритм находит широкое применение в задачах, где необходимо искать кратчайший путь в лабиринтах или сетках. Он используется в робототехнике для навигации мобильных роботов в ограниченных пространствах, в играх и задачах на программирование, где требуется нахождение пути через сетку с препятствиями. Алгоритм также используется в навигационных системах и в задачах, где необходимо оценить "доступность" всех точек пространства от начальной точки.

**Заключение**

Волновой алгоритм является простым и эффективным методом поиска кратчайшего пути в лабиринтах с препятствиями. Несмотря на его ограниченную эффективность при работе с большими лабиринтами, он все равно остается важным инструментом для решения подобных задач, благодаря своей простоте и гарантии нахождения оптимального пути.

# **3. Описание алгоритма программы**

В данной работе реализуется алгоритм генерации лабиринта с использованием метода случайного блуждания и поиска кратчайшего пути с применением волнового алгоритма.

**Генерация лабиринта**

Лабиринт представляет собой двумерную сетку, где каждая клетка может быть либо проходимой, либо стеной. Размер лабиринта задается пользователем при запуске программы и должен быть нечетным числом, большим 3.

Алгоритм генерации лабиринта включает следующие шаги:

1. **Инициализация структуры данных**: Создается двумерный массив, где все клетки изначально обозначаются как стены.
2. **Случайное блуждание**: Выбирается начальная точка, с которой начинается процесс. Путем случайного выбора направления происходит прорыв стен, пока не будут созданы коридоры лабиринта. Для предотвращения выхода за границы или создания пересечений используется проверка тупиков. Если текущая точка достигает тупика, процесс продолжается с новой случайной точки, уже находящейся в коридоре.
3. **Завершение генерации**: Процесс продолжается, пока не будут созданы все коридоры.

**Поиск пути**

Для нахождения кратчайшего пути из стартовой точки (1,1) до конечной точки (N-2, N-2) используется волновой алгоритм:

1. **Инициализация**: Стартовая точка помечается как начало волны, остальные проходимые клетки остаются незаполненными.
2. **Распространение волны**: На каждом шаге волна распространяется на соседние клетки, увеличивая значение шага. Это продолжается до тех пор, пока не будет достигнута конечная точка.
3. **Обратный проход**: Начиная с конечной точки, выполняется обратный проход по значениям шагов, что позволяет восстановить путь.

**Визуализация**

Для отображения лабиринта и пути используются следующие символы:

* # — стена.
* \* — кратчайший путь.
* Пробел — проходимая клетка, не входящая в путь.

**Сохранение и загрузка**

Программа сохраняет сгенерированный лабиринт в текстовый файл, чтобы пользователь мог продолжить работу с ним в будущем. При наличии сохраненного файла лабиринт загружается и используется вместо генерации нового.

Данный подход позволяет изучить ключевые аспекты алгоритмизации, такие как:

* Генерация случайных структур.
* Применение волнового алгоритма для решения задач поиска пути.
* Визуализация данных и их сохранение в файлы.

Ниже представлен псевдокод части программы, которая отвечает за поиск кротчайшего пути в лабиринте.

Константы:

WAY = -3 // Путь

WALL = -2 // Стена

PASS = -1 // Проход

Списки смещений для движения:

dx = [0, 1, 0, -1]

dy = [-1, 0, 1, 0]

Функция deadend(x, y, maze, height, width):

Подсчитать количество проходов вокруг текущей клетки (x, y)

Если все направления заблокированы, вернуть True

Иначе вернуть False

Функция ended(maze, height, width):

Для каждой клетки лабиринта проверить, остались ли непройденные клетки

Если нет, вернуть True

Иначе вернуть False

Функция mazemake(maze, height, width):

Инициализировать лабиринт как заполненный стенами

Установить начальные координаты (x, y)

Пока лабиринт не завершён:

Пометить текущую клетку как проход

Пока возможно движение:

Выбрать случайное направление

Если движение в этом направлении возможно и приводит к стене:

Пометить промежуточную клетку и следующую клетку как проход

Переместиться в новую клетку

Если текущая клетка — тупик:

Выбрать случайную пройденную клетку и продолжить

Завершить генерацию лабиринта

Функция drawing(map):

Очистить экран

Для каждой строки карты:

Вывести символы для стен, пути и прохода

Функция main():

Установить случайный генератор

Запросить размер лабиринта от пользователя

Проверить корректность введённого значения

Создать пустую карту

Если файл лабиринта существует:

Загрузить лабиринт из файла

Иначе:

Сгенерировать лабиринт и сохранить в файл

Отобразить лабиринт

Установить стартовую точку (1, 1)

Использовать волновой алгоритм для поиска кратчайшего пути:

Распространять волну из стартовой точки

Если достигнута конечная точка, завершить

Построить путь обратно из конечной точки в стартовую

Обозначить путь на карте

Отобразить карту с найденным путём

Если программа запускается напрямую:

Вызвать функцию main()

# **4. Описание программы**

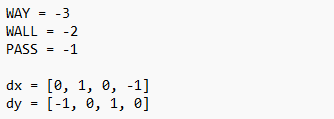


**Рис 1 – Импорт модулей**

random - модуль для генерации случайных чисел, используется для создания случайных графов.

Модуль time предоставляет функции для работы с временем. Он используется для получения текущего времени, задержки выполнения программы, работы с метками времени и многого другого.

Модуль os предоставляет функции для взаимодействия с операционной системой. Он используется для работы с файловой системой, процессами, путями и другими операционными системами.

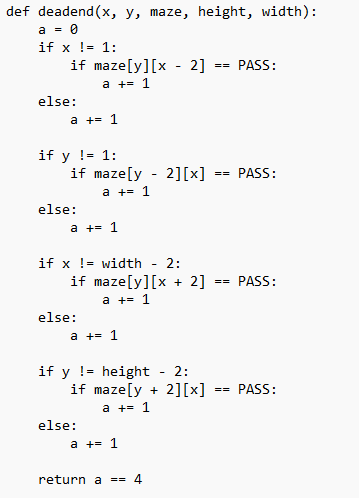


***Рис 2 – Константы и переменные***

Эти переменные задают коды для различных типов клеток в лабиринте:

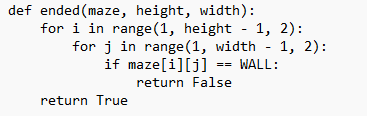
* WAY (значение -3) — путь, который будет отмечен после решения лабиринта.
* WALL (значение -2) — стена лабиринта.
* PASS (значение -1) — проходимая область лабиринта.

dx, dy - Это массивы сдвигов для перемещения по четырём направлениям: вверх (0, -1), вправо (1, 0), вниз (0, 1), влево (-1, 0).



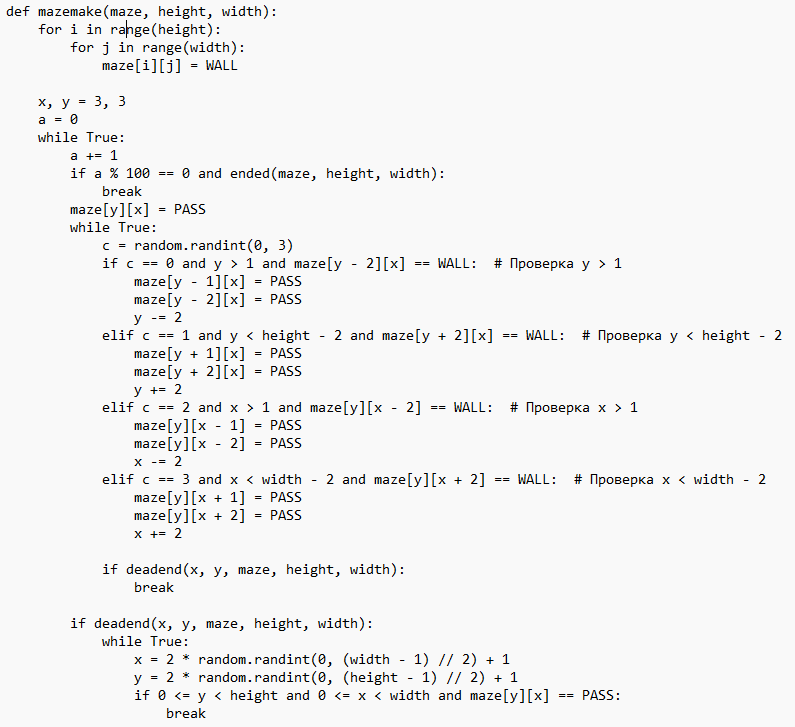
**Рис 3 – Функция deadend**

Функция deadend проверяет, является ли текущая клетка тупиком. Она проверяет соседние клетки (вверх, вниз, влево, вправо) на наличие проходимых путей. Если все четыре соседние клетки — стены или выходят за пределы лабиринта, то клетка считается тупиком.

****

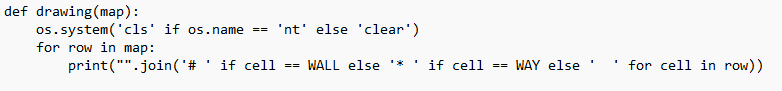
**Рис 4 – Функция ended**

Эта функция проверяет, завершена ли генерация лабиринта. Если в лабиринте ещё есть стены на позициях, которые могли бы быть путём (нечетные индексы), то генерация не завершена. Если все такие клетки очищены, лабиринт считается готовым.

****

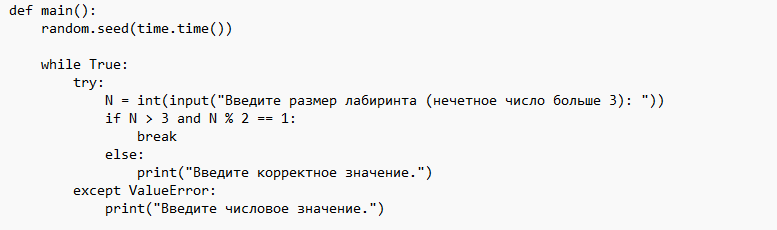
**Рис 5 - Функция mazemake**

ыЭта функция создаёт лабиринт с помощью алгоритма случайного карвинга (карвинг — вырезание пути в лабиринте). Она случайным образом выбирает направления и прорезает стены, создавая пути. Если обнаруживается тупик, программа пытается выбрать новый стартовый пункт.

****

**Рис 6 – Функция drawing**

Функция drawing отвечает за отображение лабиринта в терминале. Она очищает экран и выводит текущий лабиринт. Стены отображаются как #, путь — как \*, а пустые клетки (проходимые участки) остаются пустыми.

****

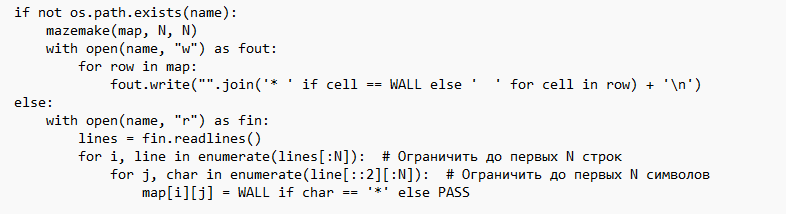
**Рис 7 – Основная функция main**

В основной функции запрашивается у пользователя размер лабиринта (нечетное число больше 3). Это значение сохраняется в переменную N.

****

**Рис 8 – Сохранение лабиринта в файл**

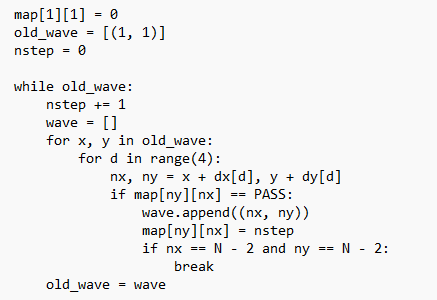
Создаётся пустой лабиринт (массив map), который будет заполнен стенами. Имя файла для сохранения лабиринта определяется как Nlab.txt.

****

**Рис 9 – создание лабиринта**

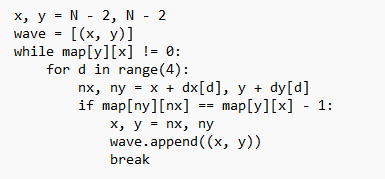
Если файл с лабиринтом ещё не существует, генерируется новый лабиринт и сохраняется в файл. Если файл уже существует, лабиринт загружается из файла.

После генерации или загрузки лабиринта его отображение происходит с помощью функции drawing.

****

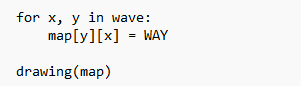
**Рис 10 – Алгоритм волнового распространения**

Этот блок реализует алгоритм волнового распространения для решения лабиринта. Он начинает с верхнего левого угла и расширяется на соседние клетки, отмечая шаги. Когда достигнута нижняя правая клетка, решение найдено.

****

**Рис 11 – Трассировка пути**

Этот блок выполняет обратный процесс — трассирует путь от конца к началу, восстанавливая наилучший путь.

****

**Рис 12 – Отображение пройденного лабиринта**

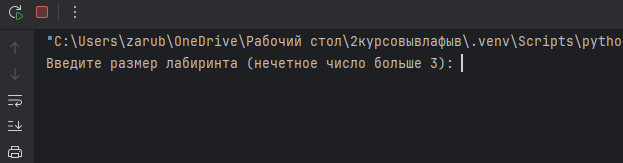
Здесь путь, найденный алгоритмом, отмечается в лабиринте как WAY. В конце отображается финальный лабиринт с решённым путём.

# **5. Тестирование**

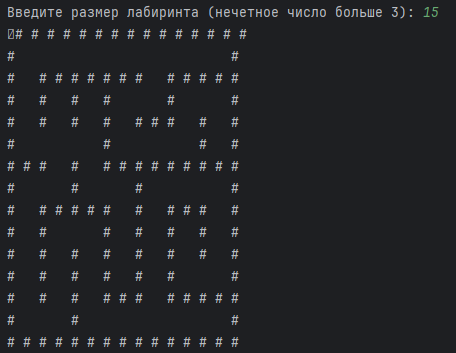
Среда разработки PyCharm предоставляет все средства, необходимые при разработке и отладке многомодульной программы. Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, алгоритмом программы, взаимодействием функций. Ниже продемонстрирован результаты тестирования программы при вводе пользователем различных размеров лабиринта и использовании различного функционала программы.

Таблица 1 – Описание поведения программы при тестировании

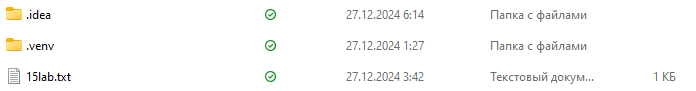
|  |  |  |  |
| --- | --- | --- | --- |
| Описание теста | Предусловие | Тестирование | Ожидаемый результат |
| Запуск программы | - | Запускаем программу с помощью PyCharm | Открытия начального экрана |
| Автоматическая генерация лабиринта | - | Ввод размера лабиринта | При вводе вразмера лабиринта на экране должен появится случайно сгенерированный лабиринт |
| Вывод лабиринта | Программа запущена | Смотрим лабиринт | Лабиринт сохраняется в текстовый файл |
| Вывод лабиринта с кратчайшим путем | Программа запущена | Видим путь к выходу в лабиринте | На экран выводится лабиринт и путь к выходу в нем |



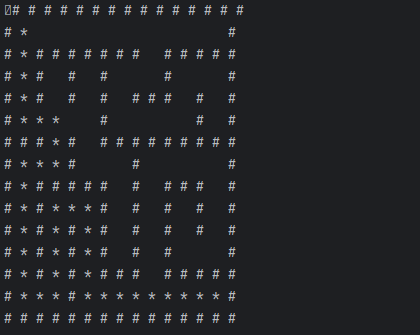
**Рис 13 – ввод размера лабиринта**

****

**Рис 14 – вывод лабиринта**

****

**Рис 15 – сохранение лабиринта в файл**

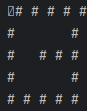
****

**Рис 16 – вывод лабиринта с путем к выходу**

Таблица 2 – Результаты тестирования

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | открытие окна | Верно |
| Автоматическая генерация лабиринта | Должна появиться строка, запрашивающая размер лабиринта | Верно |
| Вывод лабиринта | При вводе размера должен вывестись случайно сгенерированный лабиринт определенного размера, а так же сохраниться в файлы | Верно |
| Вывод лабиринта с кратчайшим путем | Вывод лабиринта с путем к выходу | Верно |

# **6. Ручной расчет задачи**

****

**Рис 17 – пример лабиринта**

**Инициализация**

1. Размер лабиринта N=5N = 5N=5.
2. Стартовая точка (1,1) — верхний левый угол.
3. Конечная точка (3,3) — нижний правый угол, доступная ячейка перед стеной.

**Шаг 1: Применение волнового алгоритма**

Алгоритм начинается с точки (1,1), которая устанавливается в 0. На каждой итерации клетки, соседствующие с текущей волной, получают значение, увеличенное на 1, если они являются проходами (−1).

**Итерация 1:**

* + Текущая волна: (1,1).
  + Соседи: (2,1) и (1,2).

**Итерация 2:**

* + Текущая волна: (2,1), (1,2).
  + Соседи:
    - Для (2,1): (3,1).
    - Для (1,2): (1,3).

**Итерация 3:**

* + Текущая волна: (3,1), (1,3).
  + Соседи:
    - Для (3,1): (3,2).
    - Для (1,3): (2,3).

**Итерация 4:**

* + Текущая волна: (3,2), (2,3).
  + Соседи:
    - Для (3,2): (3,3) (конечная точка достигнута).
    - Для (2,3): нет новых соседей.

**Шаг 2: Восстановление пути**

Начинаем с конечной точки (3,3)(3, 3)(3,3), значение которой 444, и двигаемся в соседние клетки, где значение меньше на 111, пока не достигнем стартовой точки (1,1)

1. (3,3)→(3,2)(3→3)
2. (3,2)→(3,1)(3→2)
3. (3,1)→(2,1)(2→1)
4. (2,1)→(1,1)(1→0)

****

**Рис 18 – вывод лабиринта с путем.**

**Заключение**

Таким образом, в процессе создания данного проекта разработана программа, реализующая поиск путей в лабиринте на языке Python. При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания лабиринтов. Приобретены навыки по реализации волнового алгоритма. Углублены знания языка программирования Python.

**Список литературы**

1. **Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.** "Алгоритмы: построение и анализ" – М.: "Вильямс", 2020.
2. **Седжвик Р.** "Алгоритмы на графах" – СПб.: Питер, 2019.
3. **Грабарь Ю.В., Кайнов А.С.** "Алгоритмы поиска пути: теория и практика" – М.: МГТУ им. Баумана, 2021.
4. **Сухоплюев Н.В.** "Математическое моделирование и алгоритмы поиска пути" – Казань: Казанский университет, 2018.
5. **Хирш Д.** "Основы теории графов" – М.: "Бином", 2017.
6. **Столяров А.В.** "Практическое программирование. Волновой алгоритм в задачах" – М.: Диалог-МИФИ, 2019.
7. **Шеньков В.В.** "Искусственный интеллект и алгоритмы поиска" – Новосибирск: НГУ, 2022.

# **Приложение A**

**Листинг программы**

import random

import time

import os

WAY = -3

WALL = -2

PASS = -1

dx = [0, 1, 0, -1]

dy = [-1, 0, 1, 0]

def deadend(x, y, maze, height, width):

a = 0

if x != 1:

if maze[y][x - 2] == PASS:

a += 1

else:

a += 1

if y != 1:

if maze[y - 2][x] == PASS:

a += 1

else:

a += 1

if x != width - 2:

if maze[y][x + 2] == PASS:

a += 1

else:

a += 1

if y != height - 2:

if maze[y + 2][x] == PASS:

a += 1

else:

a += 1

return a == 4

def ended(maze, height, width):

for i in range(1, height - 1, 2):

for j in range(1, width - 1, 2):

if maze[i][j] == WALL:

return False

return True

def mazemake(maze, height, width):

for i in range(height):

for j in range(width):

maze[i][j] = WALL

x, y = 3, 3

a = 0

while True:

a += 1

if a % 100 == 0 and ended(maze, height, width):

break

maze[y][x] = PASS

while True:

c = random.randint(0, 3)

if c == 0 and y > 1 and maze[y - 2][x] == WALL: # Проверка y > 1

maze[y - 1][x] = PASS

maze[y - 2][x] = PASS

y -= 2

elif c == 1 and y < height - 2 and maze[y + 2][x] == WALL: # Проверка y < height - 2

maze[y + 1][x] = PASS

maze[y + 2][x] = PASS

y += 2

elif c == 2 and x > 1 and maze[y][x - 2] == WALL: # Проверка x > 1

maze[y][x - 1] = PASS

maze[y][x - 2] = PASS

x -= 2

elif c == 3 and x < width - 2 and maze[y][x + 2] == WALL: # Проверка x < width - 2

maze[y][x + 1] = PASS

maze[y][x + 2] = PASS

x += 2

if deadend(x, y, maze, height, width):

break

if deadend(x, y, maze, height, width):

while True:

x = 2 \* random.randint(0, (width - 1) // 2) + 1

y = 2 \* random.randint(0, (height - 1) // 2) + 1

if 0 <= y < height and 0 <= x < width and maze[y][x] == PASS:

break

def drawing(map):

os.system('cls' if os.name == 'nt' else 'clear')

for row in map:

print("".join('# ' if cell == WALL else '\* ' if cell == WAY else ' ' for cell in row))

def main():

random.seed(time.time())

while True:

try:

N = int(input("Введите размер лабиринта (нечетное число больше 3): "))

if N > 3 and N % 2 == 1:

break

else:

print("Введите корректное значение.")

except ValueError:

print("Введите числовое значение.")

map = [[WALL for \_ in range(N)] for \_ in range(N)]

name = f"{N}lab.txt"

if not os.path.exists(name):

mazemake(map, N, N)

with open(name, "w") as fout:

for row in map:

fout.write("".join('\* ' if cell == WALL else ' ' for cell in row) + '\n')

else:

with open(name, "r") as fin:

lines = fin.readlines()

for i, line in enumerate(lines[:N]): # Ограничить до первых N строк

for j, char in enumerate(line[::2][:N]): # Ограничить до первых N символов

map[i][j] = WALL if char == '\*' else PASS

drawing(map)

map[1][1] = 0

old\_wave = [(1, 1)]

nstep = 0

while old\_wave:

nstep += 1

wave = []

for x, y in old\_wave:

for d in range(4):

nx, ny = x + dx[d], y + dy[d]

if map[ny][nx] == PASS:

wave.append((nx, ny))

map[ny][nx] = nstep

if nx == N - 2 and ny == N - 2:

break

old\_wave = wave

x, y = N - 2, N - 2

wave = [(x, y)]

while map[y][x] != 0:

for d in range(4):

nx, ny = x + dx[d], y + dy[d]

if map[ny][nx] == map[y][x] - 1:

x, y = nx, ny

wave.append((x, y))

break

for x, y in wave:

map[y][x] = WAY

drawing(map)

if \_\_name\_\_ == "\_\_main\_\_":

main()