Федеральное государственное бюджетное образовательное учреждение высшего образования «Нижегородский государственный архитектурно-строительный университет» (ННГАСУ)

*Факультет инженерно-экологических систем и сооружений*

*Кафедра информационных систем и технологий*

КУРСОВАЯ РАБОТА

по дисциплине: «Язык программирования Python»

На тему: «Алгоритмы поиска пути и структурное программирование»

Выполнил студент 1 курса гр. ИС-33 Хонина Е. В.

Проверил Морозов Н.С.

Нижний Новгород – 2023 г.

Содержание

[Введение 3](#_Toc137155075)

[Задачи 3](#_Toc137155076)

[1. Теоретическая часть 5](#_Toc137155077)

[2. Реализация алгоритма 8](#_Toc137155078)

[Пример работы 11](#_Toc137155079)

[Заключение 13](#_Toc137155080)

[Список литературы 14](#_Toc137155081)

[Приложение 1 15](#_Toc137155082)

[Листинг программы 15](#_Toc137155083)

# Введение

Алгоритмы обхода графа являются одной из важнейших задач в программировании, потому что графы являются универсальным инструментом для моделирования различных задач и процессов. Графы могут быть использованы для моделирования дорожных сетей, социальных сетей, электрических схем, и многих других систем.

Обход графа позволяет находить определенную информацию о графе, такую как наличие пути между двумя вершинами, нахождение кратчайшего пути между вершинами, определение связности графа, нахождение циклов и т.д. Это позволяет решать множество задач, связанных с графами, таких как оптимизация маршрутов в логистике, поиск кратчайшего пути в GPS-навигации, анализ социальных сетей, поиск ошибок в электрических схемах и многих других.

Таким образом, алгоритмы обхода графа являются ключевым инструментом для решения многих задач в программировании и науке в целом. Они позволяют находить определенную информацию о графе и применять ее для решения различных задач. Одним из таких является жадный поиск.

**Цель работы**: реализовать алгоритмы обхода графа: в ширину и А\* для задачи поиска маршрута в лабиринте.

# Задачи

1. Изучить алгоритмы поиска пути в графах и их реализацию на языке Python.
2. Изучить основы структурного программирования на языке Python.
3. Разработать программу на языке Python, реализующую алгоритм поиска пути в графе.
4. Протестировать программу на различных графах и сравнить ее эффективность с другими алгоритмами поиска пути.
5. Реализовать структуры данных, необходимые для работы алгоритмов поиска пути.
6. Применить структурное программирование для улучшения читаемости и поддерживаемости кода.
7. Написать отчет о проделанной работе, описывающий основные результаты и выводы.

# Теоретическая часть

**Жадный алгоритм (Greedy Algorithm).**

Жадный алгоритм (Greedy Algorithm) представляет собой метод нахождения оптимального решения задачи, которая включает большую однородную структуру данных (например, массив, дерево или граф). Он основывается на переходе от оптимального решения к некоторому компоненту или небольшой части структуры данных и последующем расширении его, рассматривая дополнительные компоненты структуры данных один за другим, до достижения оптимального глобального решения. Жадный алгоритм предполагает, что локальный оптимум является частью глобального оптимума, и может быть применен к любой задаче, которая требует принятия локально оптимального выбора на каждом этапе поиска глобального оптимума. Жадные алгоритмы дают хорошие результаты для некоторых математических задач [1].

**Алгоритм А\* (А-стар).**

Алгоритм A\* был представлен в 1968 году П. Хартом, Н. Нильсоном и Б. Рафаэлем и является одним из самых известных алгоритмов для поиска субоптимального пути. Он представляет собой модификацию алгоритма Дейкстры, который может находить пути ко всем точкам, в то время как A\* находит маршрут от начальной вершины к конечной с наименьшей стоимостью. Он использует модель поиска "лучший первый" в сочетании с эвристической функцией:

𝐹(𝑥)=𝐺(𝑥)+𝐻(𝑥),

где G(x) — путь от стартовой точки до текущей, H(x) — эвристическая функция,

которая оценивает расстояние до финишной точки на каждом шаге. В качестве эвристической функции A\* использует Чебышевское расстояние, которое равно наибольшему из удалений по вертикали и горизонтали (ортогональная метрика). A\* просматривает все пути, ведущие от начальной вершины к конечной, и отдает приоритет тем, которые ведут ближе к цели. На первом шаге рассматриваются узлы, смежные с начальным, и выбирается тот, который имеет минимальное значение эвристической функции. После этого узел раскрывается и процесс повторяется до тех пор, пока не будет найден минимальный путь [1].

**Структурное программирование**

Структурное программирование — это методология программирования, которая была разработана в 1960-х годах. Она предполагает, что программа должна быть разбита на небольшие блоки, называемые подпрограммами, каждая из которых должна выполнять определенную функцию. Эти подпрограммы могут быть связаны друг с другом для создания более сложных программ.

Неклюдов С. А. и Балса А. Р. в своей работе «Парадигмы программирования как инструменты разработчика программных систем» рассматривают структурное программирование как одну из парадигм программирования. Они утверждают, что структурное программирование позволяет создавать программы, которые легко читать, понимать и поддерживать.

Основные принципы структурного программирования, описанные Неклюдовым и Балсой, включают в себя:

1. Разделение программы на подпрограммы. Каждая подпрограмма должна выполнять одну конкретную функцию.
2. Использование структурных управляющих конструкций, таких как последовательность, ветвление и циклы. Это позволяет создавать программы, которые легко понимать и отлаживать.
3. Избегание использования операторов безусловного перехода, таких как goto. Вместо этого следует использовать структурные управляющие конструкции.
4. Использование однозначных имен для переменных и подпрограмм. Это позволяет создавать программы, которые легко понимать и поддерживать.
5. Избегание дублирования кода. Вместо этого следует использовать подпрограммы для выполнения повторяющихся задач.

В целом, структурное программирование является эффективным подходом к разработке программных систем. Оно позволяет создавать программы, которые легко понимать, поддерживать и модифицировать [2].

# Реализация алгоритма

Моя программа представляет собой реализацию алгоритмов поиска кратчайшего пути в лабиринте для нахождения ключа и выхода.

Первоначально программа считывает лабиринт из файла с помощью функции **read\_maze()**.

Затем производится выбор произвольного места для ключа. Сначала определяются высота и ширина лабиринта с помощью функций **len()**, которые определяют длину списка вдоль оси y и длину первого элемента в списке вдоль оси x соответственно.

Затем создается пустой список **passages**, который будет содержать координаты всех проходов в лабиринте.

Далее происходит перебор всех клеток лабиринта с помощью двух циклов **for** в диапазонах от 0 до высоты и от 0 до ширины лабиринта соответственно. Если текущая клетка является проходом (условие **if maze[i][j] == " "**), то ее координаты добавляются в список **passages** с помощью метода **append()**.

Затем с помощью функции **random.choice()** выбирается случайная клетка из списка **passages**, которая будет использоваться в качестве места расположения ключа. Координаты выбранной клетки затем используются для обновления лабиринта, где символ "\*" заменяет проход, обозначая место расположения ключа.

Далее функция **find\_path\_to\_key()** находит кратчайший путь от начальной точки до ключа с помощью жадного алгоритма и принимает два аргумента: **maze** - двумерный массив, представляющий лабиринт, и **key** - координаты клетки, где находится ключ.

Алгоритм начинает поиск с клетки (0, 1), которая считается начальной точкой. Затем создается стек **stack**, в котором хранятся кортежи, содержащие текущую позицию и путь до нее. Стек инициализируется кортежем, содержащим начальную позицию и ее путь.

Далее, пока стек не пуст, извлекается последний элемент из стека и проверяется, является ли он ключом. Если это так, то возвращается путь до ключа. В противном случае, текущая позиция помечается как посещенная, и для нее находятся соседи. Соседи сортируются по эвристической функции, которая вычисляет расстояние от соседа до ключа. Затем для каждого соседа проверяется, был ли он уже посещен, и если нет, то он добавляется в стек вместе с путем до него.

Этот алгоритм использует эвристическую функцию для выбора следующей клетки, что делает его жадным. Такой подход может не дать оптимального решения, но в большинстве случаев он работает достаточно хорошо и быстро.

Затем ячейки на этом пути помечаются символом ".".

Затем функция **find\_path\_to\_exit()** находит кратчайший путь от ключа до выхода с помощью алгоритма A\* и помечает ячейки на этом пути символом ",".

На вход функции передаются три аргумента: **maze** - лабиринт в виде двумерного массива, **start** - стартовая точка в лабиринте, **end** - конечная точка (выход) из лабиринта.

Внутри функции создается объект **PriorityQueue**, который используется для хранения вершин графа, которые еще не были обработаны. В начале алгоритма в очередь добавляется стартовая вершина с приоритетом 0 и путем, содержащим только стартовую вершину.

Далее происходит цикл, в котором из очереди извлекается вершина с наименьшим приоритетом. Если эта вершина является конечной точкой, то возвращается путь до нее. В противном случае, вершина помечается как посещенная и добавляется в множество **visited**.

Затем происходит перебор всех соседних вершин (соседей текущей вершины), которые еще не были посещены. Для каждого соседа создается новый путь, который состоит из текущего пути и соседней вершины. Для этого нового пути вычисляется приоритет, который равен сумме длины пути и эвристической функции, которая оценивает расстояние от соседней вершины до конечной вершины (выхода). Затем новый путь и соседняя вершина добавляются в очередь с приоритетом, вычисленным на предыдущем шаге.

Алгоритм продолжает работу до тех пор, пока очередь не опустеет или не будет найден путь до конечной вершины. Если путь до выхода не найден, функция вернет **None**.

В конце результат записывается в файл "maze-for-me-done.txt" с помощью функции **open()** и **write()**.

Структуры данных, используемые в программе, включают в себя двумерный массив **maze** для хранения лабиринта, стек **stack** для реализации жадного алгоритма, приоритетную очередь **queue** для реализации алгоритма A\*, множество **visited** для хранения посещенных ячеек и список **path** для хранения пути.

# Пример работы

При запуске кода (Рис. 1) пишется «Process finished with exit code 0», что означает, что процесс завершился успешно. Код выхода 0 — это стандартный код выхода, который указывает на успешное завершение программы без ошибок или проблем. Если бы процесс завершился с ошибкой, код выхода был бы отличен от нуля и сообщил бы о причине ошибки.

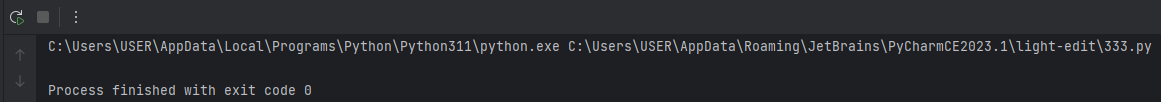


Рисунок 1. Завершение процесса

Ниже показаны начальный (Рис. 2) и конечный (Рис. 3) участки файла maze-for-me-done.txt, на которых показаны примеры работы жадного и А\* алгоритмов.

****

Рисунок 2. Начальная часть

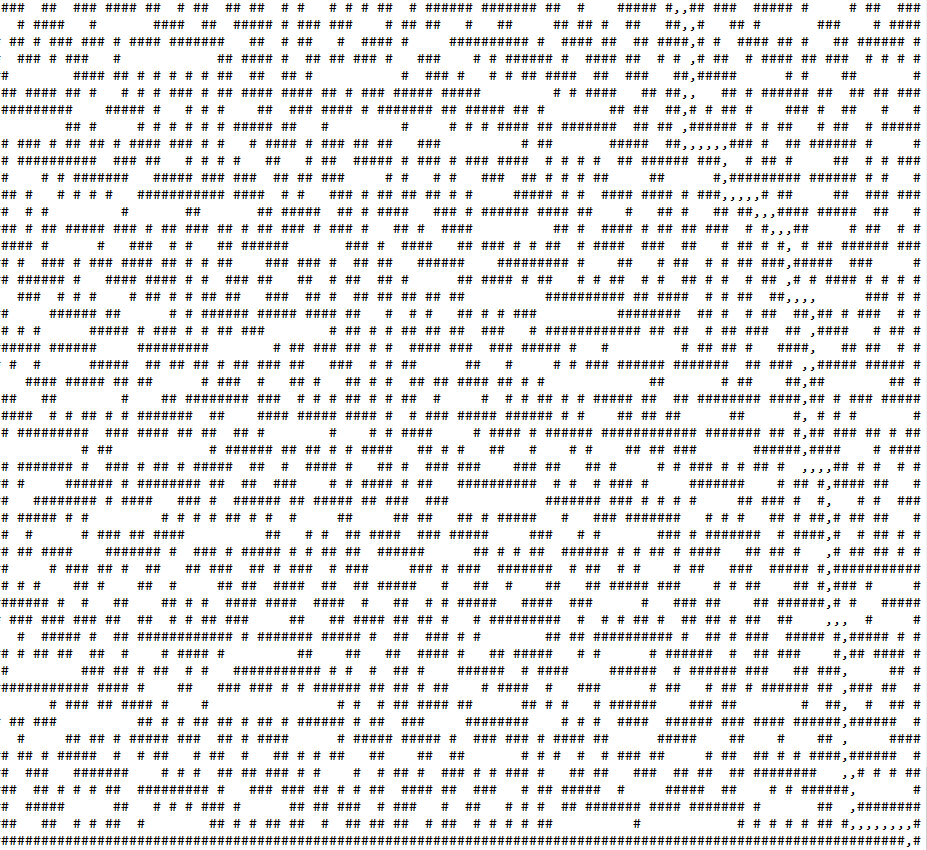


Рисунок 3. Конечная часть

# Заключение

В процессе выполнения курсовой работы были изучены основы алгоритмов обхода графа, таких как жадный алгоритм и А\*, а также структурного программирования.

Структурное программирование является важным инструментом для создания программ, которые легко понимать, поддерживать и модифицировать. Жадный алгоритм поиска пути может быть полезным в различных приложениях, в то время как A\* алгоритм может быть более эффективным для поиска пути в больших лабиринтах.

Таким образом, для достижения наилучших результатов в разработке программных систем необходимо учитывать различные методологии и алгоритмы, а также выбирать наиболее подходящие для решения конкретных задач. Использование языка программирования Python позволяет реализовывать эффективные алгоритмы поиска пути и применять структурное программирование для создания надежных и удобных в использовании программных систем.

# Список литературы

1. Научная электронная библиотека Elibrary.ru [Электронный ресурс] // Султанова Ахира Бахман «Сравнительный анализ алгоритмов поиска оптимального пути» URL: https://www.elibrary.ru/item.asp?id=44398121
2. Научная электронная библиотека Elibrary.ru [Электронный ресурс] // Неклюдова С. А., Балса А. Р. «Парадигмы программирования как инструменты разработчика программных систем» URL: https://www.elibrary.ru/item.asp?id=21687237
3. Google Scholar [Электронный ресурс] // «Greedy Algorithms» URL: <https://www.cs.cmu.edu/~avrim/451f09/lectures/lect1001.pdf>
4. Habr [Электронный ресурс] // «Введение в алгоритм A\*» URL: https://habr.com/ru/articles/331192/
5. Tproger [Электронный ресурс] // "Python для начинающих: самоучитель с примерами кода" URL: https://tproger.ru/translations/python-3-4-quick-start/
6. Proglib [Электронный ресурс] // «Структурное программирование: история, основные принципы, плюсы и минусы» URL: <https://proglib.io/p/strukturalnoe-programmirovanie-2020-01-09>
7. Tproger [Электронный ресурс] // «Python для начинающих: самоучитель с примерами кода» URL: https://tproger.ru/translations/python-3-4-quick-start/

# Приложение 1

## Листинг программы

