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»

Выполнил студент 1 курса гр. ИС-34 Ульдяков А.Р.

Проверил Морозов Н.С.

Нижний Новгород – 2023 г.

Содержание

[Введение 3](#_Toc135171313)

[Задачи 3](#_Toc135171314)

[1.Теоретическая часть 4](#_Toc135171315)

[2. Реализация алгоритма 7](#_Toc135171316)

[Пример работы 9](#_Toc135171317)

[Заключение 10](#_Toc135171318)

[Список литературы 11](#_Toc135171319)

[Листинг программы 12](#_Toc135171320)

# Введение

Алгоритмы обхода графа являются одной из важнейших задач в программировании. Возможность формального моделирования множества различных реальных систем с помощью графов, что позволяет программистам решать широкий спектр прикладных задач (например, построение маршрутов). Важно помнить, что все основные операции над графами (например, печать или копирование графа, преобразование графа из одного представления в другое) являются приложениями обхода графов [1].

В последнее время задача обхода ориентированных графов стала особенно актуальной в связи с тестированием конечных автоматов, или, точнее, объектов, рассматриваемых как конечные автоматы.

Примерами алгоритмов обхода графов являются поиск в глубину, поиск в ширину и жадные алгоритмы.

Использование жадных алгоритмов помогает решать проблемы путем принятия последовательности недальновидных и необратимых решений. Преимущество этих алгоритмов заключается в том, что их легко разрабатывать и они быстро работают, что является важным навыком для грамотного использования и применения.

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

# Задачи

* Изучить алгоритмы построения маршрута в графе;
* Выделить особенности реализации, необходимые в конкретной задаче поиска маршрута;
* Подготовить исходные данные: текстовый файл с лабиринтом, координаты точек входа и выхода, ключевая точка;
* Реализовать жадный алгоритм и А\* с заданными параметрами;
* Сохранить результаты обходов лабиринта и получившиеся маршруты в отдельный файл “maze-for-u-done.txt”.

# 1.Теоретическая часть

**Графы**

Граф - это абстрактная структура данных, состоящая из вершин и ребер, которые соединяют эти вершины. В программировании графы широко используются для моделирования различных задач, таких как поиск пути, сетевое планирование, анализ социальных сетей и т.д.  
  
Существует несколько типов графов:  
1. Неориентированный граф - граф, в котором ребра не имеют направления.  
2. Ориентированный граф - граф, в котором ребра имеют направление.  
3. Взвешенный граф - граф, в котором каждому ребру присвоено некоторое числовое значение (вес).  
4. Невзвешенный граф - граф, в котором ребрам не присваиваются веса.  
  
Для работы с графами в программировании используются различные алгоритмы, такие как поиск в ширину (BFS), поиск в глубину (DFS), алгоритм Дейкстры и алгоритм A\*. Каждый из этих алгоритмов имеет свои особенности и применяется в зависимости от конкретной задачи [1][2].

**Обход графа**

Обход лабиринта в виде графа является одним из примеров применения алгоритмов обхода графа в программировании. Лабиринт представляется в виде графа, где вершины представляют собой комнаты, а ребра - двери между комнатами.  
  
Для обхода лабиринта в программировании также используются алгоритмы поиска в ширину и поиска в глубину. При использовании алгоритма поиска в ширину, начинают с выбора начальной комнаты и посещения всех ее соседей. Затем происходит посещение соседей соседей, которые еще не были посещены, и так далее, пока не будет достигнута конечная комната или не  
будут посещены все комнаты.   
  
При использовании алгоритма поиска в глубину, начинают с выбора начальной комнаты и посещения ее первого соседа. Затем происходит посещение первого соседа этого соседа, и так далее, пока не будет достигнута конечная комната или не будут посещены все комнаты [3][4].

**A\***

Алгоритм A\* (A-star) - это эвристический алгоритм поиска кратчайшего пути в графе с весами на ребрах. Он использует комбинацию информированного поиска и жадного алгоритма, чтобы найти оптимальный путь между двумя вершинами графа.  
  
Принцип работы алгоритма A\* заключается в том, что на каждом шаге выбирается вершина с наименьшей оценкой f(x), которая представляет собой сумму стоимости пути от начальной вершины до текущей вершины (g(x)) и эвристической оценки расстояния от текущей вершины до конечной вершины (h(x)). Эвристическая оценка может быть любой функцией, которая дает приблизительное расстояние между двумя вершинами, например, расстояние по прямой линии (эвристика Манхэттена) или расстояние по количеству переходов между вершинами (эвристика Хэмминга).  
  
Алгоритм A\* можно реализовать следующим образом:  
  
1. Инициализировать два списка: открытый список и закрытый список. В открытом списке хранятся вершины, которые еще не были обработаны, а в закрытом списке - вершины, которые уже были обработаны.  
2. Добавить начальную вершину в открытый список.  
3 .Пока открытый список не пуст:  
 a. Выбрать вершину с наименьшей оценкой f(x) из открытого списка.  
   b. Если выбранная вершина - конечная, то алгоритм завершается.  
   c. Иначе переместить выбранную вершину из открытого списка в закрытый список и рассмотреть все соседние вершины.  
   d. Для каждой соседней вершины:  
   i. Если соседняя вершина уже находится в закрытом списке, то пропустить ее.  
   ii. Если соседняя вершина еще не находится в открытом списке, то добавить ее в открытый список и вычислить ее оценку f(x).  
   iii. Если соседняя вершина уже находится в открытом списке, то обновить ее оценку f(x), если новая оценка меньше старой.  
  
Алгоритм A\* является оптимальным и гарантирует нахождение кратчайшего пути, если эвристическая оценка h(x) удовлетворяет условию admissibility (оценка не должна быть завышенной) и consistency (оценка должна быть монотонной). Однако, если эвристика не удовлетворяет этим условиям, то алгоритм может давать неправильный результат [5][6].

**Жадный алгоритм**

Жадный алгоритм (greedy algorithm) - это метод решения задач, при котором на каждом шаге выбирается лучшее решение на основе текущей информации, без учета будущих последствий. Такой подход позволяет быстро получить приближенное решение задачи, но не гарантирует оптимальность результата.

Принцип работы жадного алгоритма заключается в том, что на каждом шаге выбирается локально оптимальное решение, которое в итоге должно привести к глобально оптимальному решению. Жадный алгоритмы часто используются для оптимизации задач, связанных с вычислительной сложностью, таких как задачи на графах, задачи на динамическое программирование и другие.

Примеры задач, которые можно решить с помощью жадного алгоритма:

1. Задача о рюкзаке (knapsack problem) - требуется заполнить рюкзак предметами различной стоимости и объема таким образом, чтобы получить максимальную стоимость предметов в рюкзаке, не превышая его вместимости.

2. Задача о минимальном остовном дереве (minimum spanning tree problem) - требуется найти минимальное поддерево графа, которое содержит все вершины и имеет минимальную сумму весов ребер.

3. Задача о кратчайшем пути в графе (shortest path problem) - требуется найти кратчайший путь между двумя вершинами графа.

Жадный алгоритм можно реализовать различными способами, в зависимости от конкретной задачи. Например, для задачи о рюкзаке можно использовать следующий алгоритм:

1. Отсортировать предметы по убыванию стоимости на единицу веса.

2. Поместить в рюкзак предмет с наибольшей стоимостью на единицу веса, пока есть место.

3. Если место закончилось, разбить последний предмет на части пропорционально оставшемуся месту в рюкзаке.

Однако жадный алгоритм не всегда дает наилучший результат, так как он не учитывает будущие последствия своих решений. Например, если выбирать самый выгодный предмет на каждом шаге, то в итоге может получиться так, что в рюкзаке не окажется места для более ценных предметов, которые могли бы поместиться, если бы были выбраны на более поздних шагах [7][8].

# 2. Реализация алгоритма

Код считывает лабиринт из файла с помощью функции read\_maze. Функция возвращает массив NumPy, представляющий лабиринт. Затем переменным size\_x и size\_y устанавливаются размеры лабиринта.

Функция write\_maze записывает решенный лабиринт в новый файл. Принимает два аргумента: массив maze и путь к файлу.

Функция get\_neighborst принимает лабиринт, строку и столбец ячейки в качестве аргументов и возвращает список ее соседей. Также проверяет, является ли каждый сосед допустимой ячейкой, а не стеной.

Функция Greedy\_best\_first\_search принимает лабиринт, начало и цель в качестве аргументов и возвращает решенный лабиринт. Использует жадный алгоритм поиска "лучший первым", чтобы найти кратчайший путь от начала до сокровища. Функция инициализирует приоритетную очередь с начальной ячейкой и стоимостью, равной 0. Далее переходит в цикл, который продолжается до тех пор, пока приоритетная очередь не опустеет или цель не будет найдена. На каждой итерации цикла удаляет ячейку с наименьшей стоимостью из очереди приоритетов и проверяет, является ли она целью. Если это цель, функция выходит из цикла. Если это не является целью - получает соседей ячейки и вычисляет стоимость для каждого соседа. Затем добавляет соседа в очередь приоритетов с приоритетом, равным его стоимости. Функция также отслеживает путь от начала до каждой ячейки, используя словарь came\_from, и стоимость доступа к каждой ячейке, используя словарь cost\_so\_far. Потом использует словарь came\_from для построения кратчайшего пути от начала до цели.

Функция a\_star\_search принимает лабиринт, начало и цель в качестве аргументов и возвращает решенный лабиринт. Применяет алгоритм поиска A\*, чтобы найти кратчайший путь от сокровища к выходу. Функция похожа на функцию greedy\_best\_first\_search, но она использует эвристику для оценки затрат от каждой ячейки до цели. Используемая эвристика - это евклидово расстояние между ячейкой и целью.

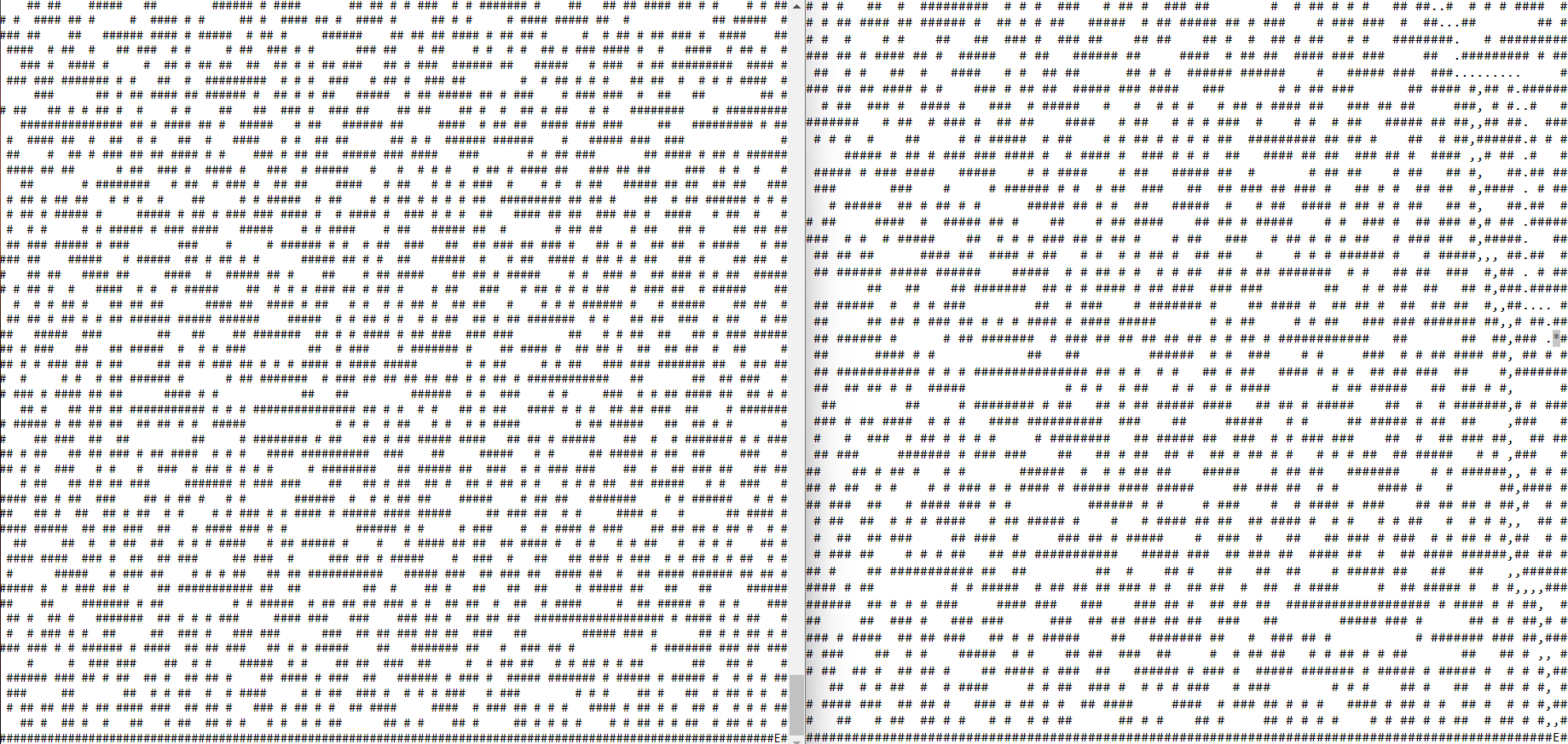
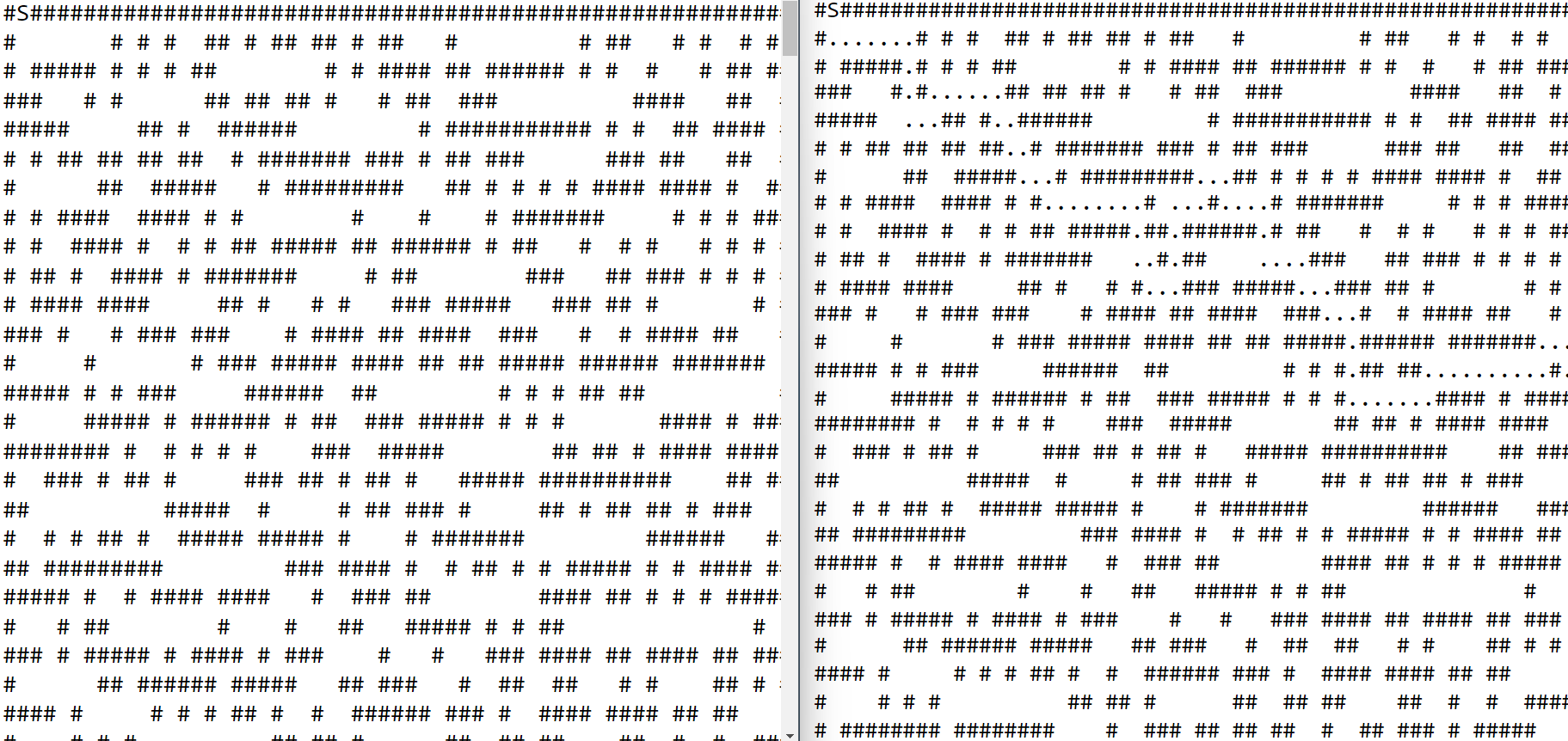
Функция remove\_walls принимает решенный лабиринт в качестве аргумента и удаляет стены, в случае, если путей из лабиринта нет, и аватар зациклился. для построения пути от начала к выходу. Функция сначала инициализирует словарь wall\_count, который подсчитывает, сколько раз каждая стена примыкала к ячейке пути. Далее функция выполняет итерацию по лабиринту и обновляет словарь wall\_count для каждой ячейки пути. Затем функция удаляет стены, содержащие более трех смежных ячеек контура.

Функция solve\_maze принимает путь к файлу лабиринта в качестве аргумента и возвращает решенный лабиринт. Сначала считывает лабиринт из файла, используя функцию read\_maze, затем находит начальную ячейку и запрашивает координаты клада. Функция проверяет, является ли ячейка с сокровищами стеной, и предлагает пользователю ввести новые координаты, если это так. Затем помечает ячейку с сокровищами символом «\*» и находит кратчайший путь от начала до сокровища, используя функцию greedy\_best\_first\_search. После находит кратчайший путь от сокровища к выходу, используя функцию a\_star\_search. Наконец удаляет стены с помощью функции remove\_walls и возвращает решенный лабиринт.

В конечном итоге переменной maze присваивается значение решенного лабиринта, возвращаемого функцией solve\_maze.

# Пример работы

Изначальный лабиринт После работы кода

****

# Заключение

В результате проделанной работы был создан и реализован алгоритм генерации путей в графе с использованием жадного алгоритма и алгоритма A\*. Для разработки программы потребовались исследования в области обработки графов, алгоритмов обхода графов и ранжирования приоритетов.

В конечном итоге была разработана программа на языке программирования Python для поиска пути от начальной до конечной точки в лабиринте. Используя жадный алгоритм и A\*, была сгенерирована карта лабиринта от начальной точки до сокровища (.), и от сокровища до выхода (,).

# Список литературы

1. "Graphs and Graph Algorithms" [Электронный ресурс] URL: <https://web.stanford.edu/class/archive/cs/cs161/cs161.1168/lecture13.pdf> (Дата обращения: 29.04.2023)
2. "Graphs", GeeksforGeeks [Электронный ресурс] URL: <https://web.stanford.edu/class/archive/cs/cs161/cs161.1168/lecture13.pdf> <https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/> (Дата обращения: 29.04.2023)
3. "Graph Traversal - Breadth First and Depth First Search", Khan Academy, [Электронный ресурс] URL: <https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs> (Дата обращения: 29.04.2023)
4. "Graph Traversal: Depth-First Search and Breadth-First Search", Coursera, [Электронный ресурс] URL: <https://www.coursera.org/lecture/algorithms-part2/graph-traversal-depth-first-search-and-breadth-first-search-5z7YD> (Дата обращения: 29.04.2023)
5. "A\* Search Algorithm", GeeksforGeeks, [Электронный ресурс] URL: <https://www.geeksforgeeks.org/a-search-algorithm/> (Дата обращения: 29.04.2023)
6. "Artificial Intelligence - A\* Search Algorithm", Tutorialspoint, [Электронный ресурс] URL: <https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_a_star_search_algorithm.htm> (Дата обращения: 29.04.2023)
7. "Greedy Algorithm", GeeksforGeeks, [Электронный ресурс] URL: <https://www.geeksforgeeks.org/greedy-algorithms/> (Дата обращения: 02.05.2023)
8. "Greedy Algorithms", Brilliant, [Электронный ресурс] URL: <https://brilliant.org/wiki/greedy-algorithm/> (Дата обращения: 02.05.2023)

# Листинг программы

**Приложение 1**



