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-34 Таймаров Д.А.

Проверил Морозов Н.С.

Нижний Новгород – 2023 г.

**Содержание**

[**Введение** 3](#_Toc135937272)

[**Теоретическая часть** 4](#_Toc135937273)

[Алгоритмы поиска пути 4](#_Toc135937274)

[Жадный алгоритм 6](#_Toc135937275)

[Жадный алгоритм в повседневной жизни 6](#_Toc135937276)

[Жадный алгоритм в поиске пути 6](#_Toc135937277)

[Алгоритм А\* 8](#_Toc135937278)

[**Реализация алгоритма** 10](#_Toc135937279)

[**Пример работы** 12](#_Toc135937280)

[**Заключение** 13](#_Toc135937281)

[**Список литературы** 14](#_Toc135937282)

[**Приложение** 15](#_Toc135937283)

[Листинг программы 15](#_Toc135937284)

# **Введение**

Алгоритмы обходов графов (лабиринтов) являются ключевой частью программирования, потому что многие задачи могут быть представлены в виде этих самых графов, в которых мы хотим найти кратчайший путь между вершинами. Для реализации нашей задачи требуется использовать разного рода алгоритмы, которые значительно отличаются друг от друга и используются для решения различающихся своей структурой графов. В данной работе будут рассмотрены такие алгоритмы, как «Жадный» и «А\*».

**Цель работы:** Осуществить «Жадный» и «А\*» алгоритмы для обхода графа (лабиринта)

Задачи

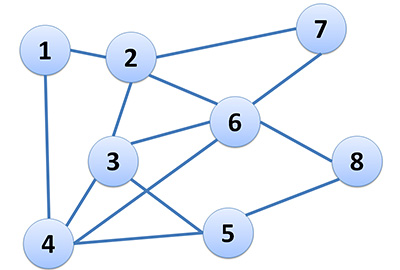
* Изучить «Жадный» и «А\*» алгоритмы
* Придумать как реализовать данные алгоритмы для нахождения маршрута в лабиринте.
* Подготовить лабиринт, а также создать координаты точек старта, ключа и выхода для проверки алгоритмов.
* Реализовать алгоритмы.
* Оформить пройденный алгоритмами путь в виде точек и запятых для наглядности.
* Создать новый файл для хранения лабиринта с реализованными алгоритмами и отмеченым путём.

# **Теоретическая часть**

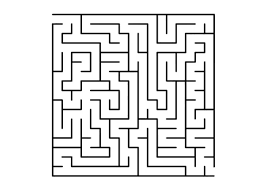
## Алгоритмы поиска пути

Поиск пути – это одна из задач, решение которой может вызвать сложности , как в выборе более подходящего для данной задачи алгоритма, так и для его реализации. Очень часто задача нахождения кратчайшего поиска пути стоит перед разработчиками игр, GPS-приложений или сервисов с картами городов или прочей местности. В случае с играми искуственный интеллект не может знать, как ему добираться до какого-либо объекта, ему нужно, чтобы был прописан чёткий алгоритм поиска пути. А в случае с GPS-приложениям ИИ необходимо не только нахождение кратчайшего пути, а также учитывание многих факторов, таких как пробки, ремонтные работы на определённой трассе и при этом, чтобы путь, созданный алгоритмом работал быстро для того, чтобы всегда необходимая информация всегда была актуальной.

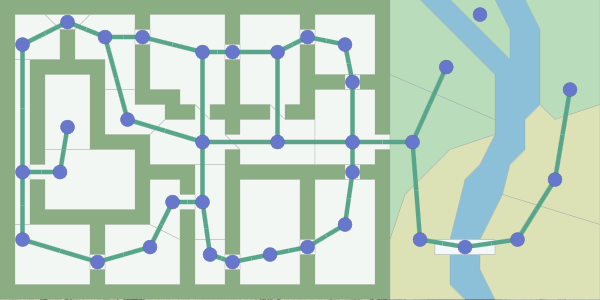
Все алгоритмы поиска пути работают с таким элементом, как графы (лабиринты).  
Граф – структура из вершин (точек), которые соединены рёбрами (линиями). В графе движение происходит по рёбрам от точек в точки.  
Так выглядит граф:



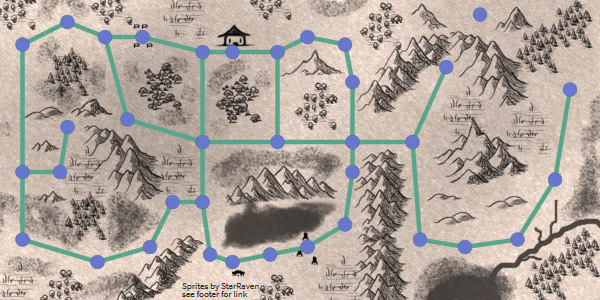
Так выглядит лабиринт:



Надо понимать, что для любого алгоритма важен только граф. Алогритм не видит то, что находится вне графа.



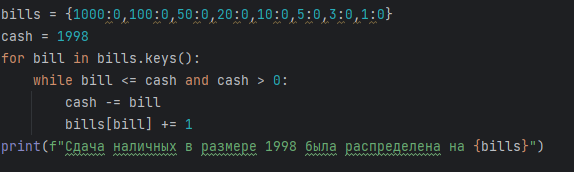
Соответственно обе данные картинки для алгоритма будут абсолютно идентичны несмотря на отличительную внешность карт.



## 

## Жадный алгоритм

### Жадный алгоритм в повседневной жизни

Жадный алгоритм можно часто встретить в повседневной жизни, когда нам нужно первоначально делать первостепенные задачи, а второстепенные постоянно ставить по приоритету ниже, чем первостепенные. Например, когда человек, занимающий фрилансом получает в день несколько заказов, то он в первую очередь делает самые прибыльные, а к менее прибыльным приходит только после того, как самые прибыльные закончились.  
Также можно привести ещё один пример с иллюстрацией:  
  
Данный код выдаёт наличные купюрами и монетами по приниципу жадного алгоритма: мы разбиваем число 1998 на самые большие купюры, сначала мы выдаём 1 купюру в виде 1000 рублей, затем 9 купюр в виде 100 рублей и так далее по убывающей.

Результат работы:



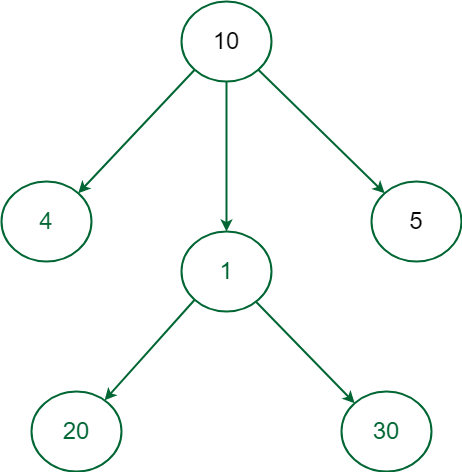
### Жадный алгоритм в поиске пути

Принцип работы жадного алгоритма в поисках пути лежит в том, что на каждом своём шагу он делает выбор в сторону ячейки (позиции), которая имеет наименьшую стоимость до того момента, пока не был достигнут пункт назначения. В жадном алгоритме используется эвристическое расстояние, которое оценивается как наименьшее расстояние от соседних клеток до конечной, за счёт этого расстояния выбирается самая ближайшая клетка именно с точки зрения расстояния, не учитывая заграждения.

Минус жадного алгоритма лежит в том, что в случае довольно сложной структуры графа с большим количеством препятствий жадный алгоритм будет неэффективен, и на проход препятствий он будет тратить крайне много времени.

В большинстве своём перед выбором жадного алгоритма нужно исследовать структуру задачи, устройство пути, иначе выбор такого алгоритма может привести к неудачным последствиям.

Пример на графе.  
Как будет работать жадный алгоритм здесь?  
Будем считать, что значения на вершинах это монеты, соответственно, где значение больше, та вершина лучше на первый взгляд.  
Если взять вершину с значением 10, как нашу стартовую точку, то следующую вершину для перемещения алгоритм выбирет с наибольшим значением, то есть 5. И путь на этом закончится, а мы в итоге получим 10 + 5 15 монет. Но чтобы найти наиболее прибыльный путь, нужно вычислить всю возможную сумму путей, и тогда мы получим 10 + 1 + 30 = 41 монет.



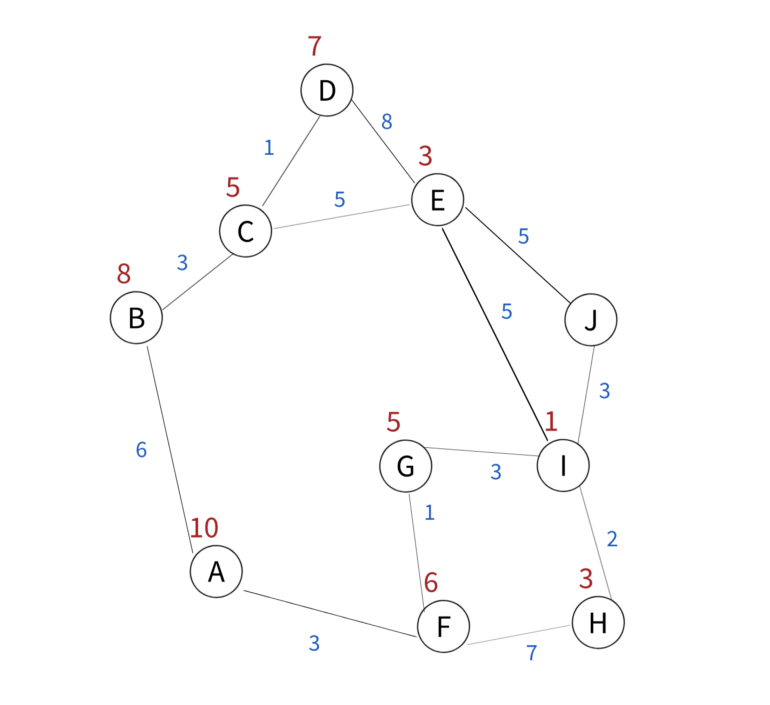
## Алгоритм А\*

Алгоритм А\* оптимизирован для нахождения пути от начальной до единственной целевой точки, следовательно для А\* будут предпочтительны точки, через которые пути для конечной цели будут преимущественно лучше.

Алгоритм А\* не всегда будет работать быстрее остальных алгоритмов, но в отличии от них А\* всегда будет находить кратчайший путь, а главное, что он универсален для разных структур графов, так как в нём применяется целая эвристическая функция.  
Эвристическая функция: F(x) = G(x) + H(x)  
G(x) – стоимость соседних клеток для возможного посещения.  
H(x) – эвристическое расстояние.  
F(x) – общая стоимость соседней клетки, складывающихся из 2 показателей выше.

Алгоритм А\*, в отличие от других алгоритмов обхода графов, является «умным» алгоритмом, и многие игры и веб-карты используют этот алгоритм для крайне эффективного поиска кратчайшего пути. Для примера взять жанр Tower Defense. В этом типе стратегической игры игрок должен защищаться от противника путём выстраивания сооружений, также в подобного вида играх используются многочилсенные типы местности для разнообразия процесса игры, но для того, чтобы искусственный интеллект врагов не сломался, имея такое большое количество факторов на своём пути, и используется алогритм А\*, ведь у всех врагов одна точка назначения, а на пути стоят «вершины с разными весами». А для таких условий А\* подходит лучше всего.

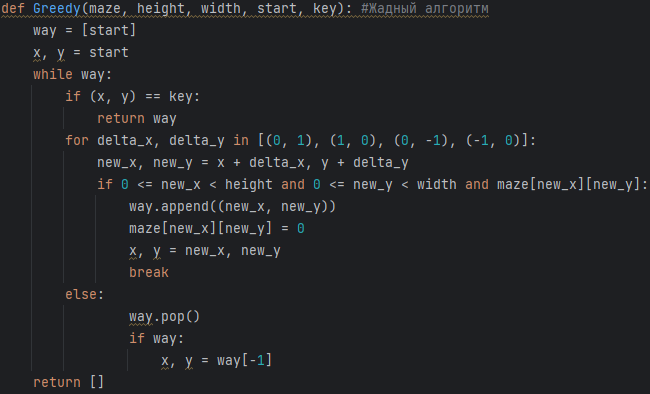
Разберём пример:



Наша задача: найти кратчайший путь из вершины А в вершину J. Сначала вычисляются 2 соседние с A вершины: B и F. f(B) = G(B) + H(B),   
где g(B) = 6, как стоимость самой клетки, а h(B) = 8, как эвристическое расстояние до точки J. Получим, что f(B) = 14, тоже самое сделаем с точкой F. f(F) = 3 + 6 = 9. Так как f(F) < f(B), то переходим в вершину F(F) и видим 3 ближайшие вершины: A, G, H. A – у нас должна быть записана как посещённая, то тогда остаются только G и H. Производим те же самые операции, получаем, что f(G) = 9, f(H) = 13, f(G) < f(H), значит следуюшая точка, в которую мы перемещаемся – это f(G). Рядом с ней есть только I. дорога от f(I) = 7 + 1 = 8. Так как все значения которые, нам попадались по пути A-F-G-I-J будут меньше значения f(B), то в точку B возвращаться нет смысла, но если бы было наоборот, тогда мы продолжили строить путь и от точки B.

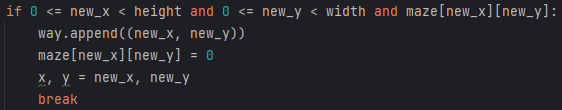
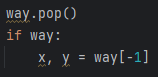
# **Реализация алгоритма**

Общий вид функции жадного алгоритма:

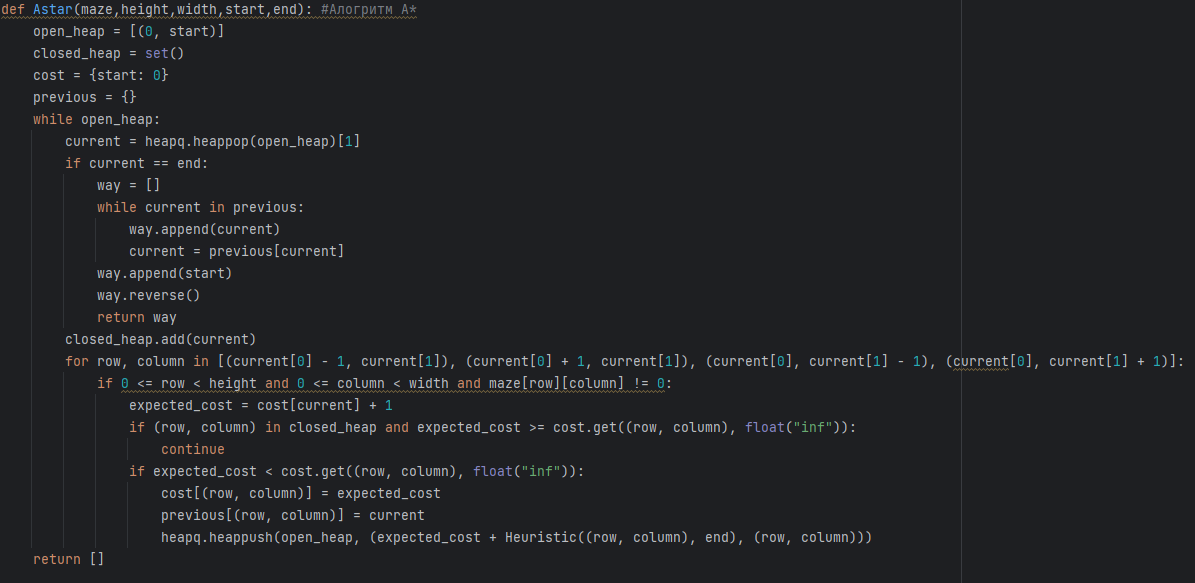


Функция содержит 5 параметров: сам лабиринт, его высота, длина, координаты начала, координаты ключа (обе точки хранятся в виде кортежа, а лабиринт в виде списка списков).  


Создаём, как список с кортежом, в него будем класть все последующие точки нашего пути. Создаём 2 переменные x, y для хранения координат последующих точек.  
В цикле создаём проверку на сравнение текущих координат с координатами ключа, чтобы когда алгоритм дошёл до ключа вернул нам весь пройденный путь.

Создаём соседние точки:  
  
Выставляем условие, чтобы алгоритм не лез в стены и не заходил за границы лабиринта. Допустимые точки записываем в наш путь.  
  
Если пришли в тупик, то удаляем последнюю точку и возвращаемся в предыдущую.  


Общий вид функции A\*

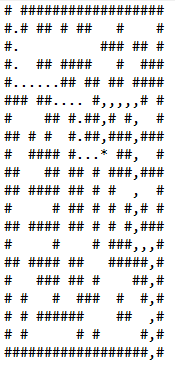


A\* принимает такие же параметры, только start на этот раз имеет координаты ключа, а end - координаты выхода.

Для A\* была импортирована дополнительная библиотека heapq для более удобный преобразований структур данных.  
  
Также мы использовали Эвристическую стоимость для алгоритма A\*.  


# **Пример работы**

Точки – путь до ключа с помощью жадного алгоритма  
Звёздочка – ключ  
Запятые – путь от ключа до выхода с помощью A\*



# **Заключение**

В ходе выполнения лабораторной работы были изучены различные алгоритмы поиска пути, а также их применение и реализация в разных сферах жизни. А также, получив новые знания, мы применили их на практике, написав код, использующий сразу 2 алгоритма (жадный и А\*) для обхода лабиринта и нахождения в нём кратчайшего пути.

# **Список литературы**

1. Веб-сайт с новостями, статьями, связынные с информационными технологиями: электронный сайт.  
Введение в алгоритм A\*.  
URL: <https://habr.com/ru/articles/331192> (дата обращения: 23.04.2023)

2. Платформа для статей в областях науки и компьютерных технологий: электронный сайт.  
Характеристика и применение жадного алгоритма.  
URL: <https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials> (дата обращения: 20.05.2023)

3. Платформа для публикации текстов, статей и блогов: электронный сайт.  
Пошаговый разбор алгоритма A\*.  
URL: <https://medium.com/nuances-of-programming/алгоритм-поиска-a-3bb59be05a79> (дата обращения: 14.05.2023)

4. Графовая система управлениями базами данных:  
Алгоритмы поиска пути.  
URL: <https://neo4j.com/developer/graph-data-science/path-finding-graph-algorithms/> (дата обращения: 23.05.2023)

5. Swaroop C.H. A Byte of Python / электронная книга // версия 2.02 – 2020 -164 с.

# **Приложение 1**

## Листинг программы

import heapq  
def Greedy(maze, height, width, start, key): #Жадный алгоритм  
 way = [start]  
 x, y = start  
 while way:  
 if (x, y) == key:  
 return way  
 for delta\_x, delta\_y in [(0, 1), (1, 0), (0, -1), (-1, 0)]:  
 new\_x, new\_y = x + delta\_x, y + delta\_y  
 if 0 <= new\_x < height and 0 <= new\_y < width and maze[new\_x][new\_y]:  
 way.append((new\_x, new\_y))  
 maze[new\_x][new\_y] = 0  
 x, y = new\_x, new\_y  
 break  
 else:  
 way.pop()  
 if way:  
 x, y = way[-1]  
 return []  
  
def Heuristic(row, column): #Эвристическая стоимость  
 return abs(column[0] - row[0]) + abs(column[1] - row[1])  
  
def Astar(maze,height,width,start,end): #Алогритм A\*  
 open\_heap = [(0, start)]  
 closed\_heap = set()  
 cost = {start: 0}  
 previous = {}  
 while open\_heap:  
 current = heapq.heappop(open\_heap)[1]  
 if current == end:  
 way = []  
 while current in previous:  
 way.append(current)  
 current = previous[current]  
 way.append(start)  
 way.reverse()  
 return way  
 closed\_heap.add(current)  
 for row, column in [(current[0] - 1, current[1]), (current[0] + 1, current[1]), (current[0], current[1] - 1), (current[0], current[1] + 1)]:  
 if 0 <= row < height and 0 <= column < width and maze[row][column] != 0:  
 expected\_cost = cost[current] + 1  
 if (row, column) in closed\_heap and expected\_cost >= cost.get((row, column), float("inf")):  
 continue  
 if expected\_cost < cost.get((row, column), float("inf")):  
 cost[(row, column)] = expected\_cost  
 previous[(row, column)] = current  
 heapq.heappush(open\_heap, (expected\_cost + Heuristic((row, column), end), (row, column)))  
 return []  
  
mazefile = "maze-for-u.txt"  
maze = []  
greedymaze = []  
astarmaze = []  
  
with open(mazefile, "r", encoding="utf-8") as readfile:  
 file = readfile.readlines()  
 for line in file:  
 shear = [0 if sym == "#" else 1 for sym in line[:-1]]  
 maze.append(shear[:])  
 greedymaze.append(shear[:])  
 astarmaze.append(shear[:])  
  
height = len(file)  
width = len(file[0]) - 1  
start = (58, 25)  
print("Координаты аватара: ", start,"\n")  
key = (542, 510)  
continuator = key  
print("\nКоординаты ключа: ", key,"\n")  
escape = (355,421)  
print("\nКоординаты выхода: ", escape,"\n")  
  
greedyway = Greedy(greedymaze, height, width, start, key) #Поиск пути до ключа через жадный алгоритм  
astarway = Astar(astarmaze, height, width, continuator, escape) #Поиск пути до выхода от ключа через алгоритм А\*  
  
print(greedyway,'\n',astarway)  
  
with open("maze-for-me-done.txt", "w", encoding="utf-8") as writefile:  
 for x in range(height):  
 for y in range(width):  
 if (x, y) == key:  
 writefile.write("\*")  
 elif ((x, y) in astarway and (x, y) in greedyway):  
 writefile.write(";")  
 elif (x, y) in greedyway:  
 writefile.write(".")  
 elif (x, y) in astarway:  
 writefile.write(",")  
 elif maze[x][y] == 0:  
 writefile.write("#")  
 else:  
 writefile.write(" ")  
 writefile.write("\n")  
print("Лабиринт пройден! Был создан файл: maze-for-me-done.txt")