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 Звездочетов П. А.

Проверил Морозов Н.С.

Нижний Новгород – 2023 г.

**Содержание**

[Введение 3](#_Toc134211393)

[Цель работы 3](#_Toc134211394)

[Задачи 3](#_Toc134211395)

[Основная часть 4](#_Toc134211396)

[Структурное программирование 4](#_Toc134211397)

[Алгоритмы поиска пути 5](#_Toc134211398)

[Жадный алгоритм 5](#_Toc134211399)

[Алгоритм А\* 5](#_Toc134211400)

[Реализация алгоритма 6](#_Toc134211401)

[Пример работы алгоритма 9](#_Toc134211402)

[Заключение 10](#_Toc134211403)

[Список литературы 11](#_Toc134211404)

[Приложения 12](#_Toc134211405)

# Введение

В данной работе будут рассмотрены элементы программирования, которые имеют важное значение: алгоритмы поиска пути и структурное программирование. Алгоритмы поиска пути существенны в областях, где необходимо находить оптимальные решения, например, в логистике, играх и т.д. Структурное программирование предназначено для улучшения читаемости, тестируемости и обслуживания кода программы. Эта работа охватит такие алгоритмы поиска пути, как жадный алгоритм и алгоритм A\*. При выполнении работы будут использоваться принципы структурного программирования, такие как использование функций и разделение программы на модули. В конечном итоге, будет разработана программа, реализующая данные алгоритмы поиска пути в структурной парадигме программирования на языке Python.

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

## Задачи:

1. Изучить, что такое структурное программирование;
2. Изучить теорию алгоритмов обхода графа: жадного и А\*;
3. Написать программу на языке Python для поиска маршрута в лабиринте с помощью жадного алгоритма и алгоритма А\*.
4. Сохранить результаты обходов лабиринта и получившиеся маршруты в файл.
5. Разобрать работу написанного кода, объяснить принцип работы алгоритмов и структуру программы.

# Основная часть

## Структурное программирование

Структурное программирование — это метод проектирования и написания компьютерных программ, в котором используются только некоторые структуры языка программирования, такие как условия, циклы и подпрограммы, а также структуры данных.

Структурное программирование было разработано в 60-х годах 20 века как реакция на проблемы с производительностью и надежностью программ, написанных без использования явно выраженной структуры [5; c. 448].

Структурное программирование является основой для других методов программирования, таких как объектно-ориентированное программирование и функциональное программирование[6].

Главным принципом структурного программирования является тот факт, что любая программа может быть разложена на более мелкие и понятные взаимосвязанные модули. Это может значительно упростить процесс программирования, улучшить производительность и повысить надежность создаваемого программного продукта.

Некоторые из наиболее распространенных структур языка программирования, которые применяются в структурном программировании, включают в себя:

- Условия - позволяют контролировать, выполняется ли определенный фрагмент кода в зависимости от того, выполняются ли заданные условия.

- Циклы - позволяют выполнять фрагменты кода многократно в соответствии с заданными условиями.

- Подпрограммы - позволяют избежать дублирования кода путем выделения общих операций в отдельные функции и процедуры.

Одним из недостатков структурного программирования является ограниченность в решении некоторых задач, которые требуют более сложных алгоритмов и структур данных[1].

## Алгоритмы поиска пути

### Жадный алгоритм

Жадный алгоритм (Greedy algorithm) - это метод решения задач, который всегда выбирает локально наилучшее решение на каждом этапе, надеясь, что в итоге получится глобально оптимальное решение. Жадные алгоритмы используются для решения многих задач, например: минимальное остовное дерево, минимальный путь в лабиринте, покрытие множества, оптимизация расписания и т.д[4; c. 442-443].

Основной принцип жадного алгоритма заключается в том, что на каждом этапе алгоритма выбирается наиболее оптимальный вариант, который не противоречит ограничениям задачи. Это может привести к тому, что весь алгоритм найдет оптимальное решение для задачи [3].

### Алгоритм А\*

Алгоритм A\* (A-Star) является одним из наиболее популярных алгоритмов поиска пути в картах, используется в играх, приложениях GPS и в других областях, где важен нахождение оптимального пути.

Принцип работы алгоритма A\* основывается на использовании эвристики, чтобы выбрать более эффективный путь. Результатом работы алгоритма является наименьшее количество шагов, необходимых для достижения цели [8].

Порядок обхода вершин определяется эвристической функцией «расстояние + стоимость» (обычно обозначаемой как f(x)). Эта функция — сумма двух других: функции стоимости достижения рассматриваемой вершины (x) из начальной (обычно обозначается как g(x) и может быть как эвристической, так и нет), и функции эвристической оценки расстояния от рассматриваемой вершины к конечной (обозначается как h(x))[7].

Преимущество алгоритма A\* заключается в его способности находить наиболее оптимальный путь. Однако, его работу не всегда можно предсказать, особенно если карта большая, со множеством препятствий и тонкими участками [2].

Тем не менее, алгоритм A\* остается одним из наиболее популярных алгоритмов поиска пути, используется в большом количестве приложений и игр.

# Реализация алгоритма

Реализацию алгоритма начнём с проектирования определенной структуры. В программе будет создана управляющая функция. В рамках этой функции будут выполнены следующие этапы: загрузка лабиринта из файла, установка начальной точки, ключевых точек и точки выхода, последовательный вызов алгоритмов поиска пути, перенос траекторий в лабиринт и создание файла с полученными результатами. Для каждого этапа будет создана отдельная функция, что уменьшит нагрузку на главную функцию и обеспечит модульное построение программы, что в свою очередь облегчит ее разработку и отладку.

Проработав примерную структуру программы, приступим к написанию кода (Приложение 1).

Функция `readMaze` считывает лабиринт из файла `maze-for-u.txt` и преобразует его в двумерный список, заменяя символы "#" на "1" и пробелы на "0". Затем она возвращает этот список.

Функция `key` запрашивает у пользователя координаты ключевой точки и проверяет их на допустимость. Если введенные координаты находятся на стене, функция оповещает об этом и просит ввести новые координаты. Когда координаты проходят проверку, функция возвращает их в виде кортежа.

Функции `start` и `end` ищут начальную и конечную точки путей в лабиринте. Функция `start` ищет свободную клетку в верхнем ряду лабиринта, а функция `end` ищет свободную клетку в нижнем ряду.

Далее следуют две вспомогательные функции для алгоритмов поиска: `heuristic` и `get\_neighbors`.

Функция `heuristic` вычисляет эвристическую оценку расстояния от текущей точки до целевой точки. В данной реализации используется евклидово расстояние.

Функция `get\_neighbors` ищет соседние свободные клетки, которые можно посетить из текущей клетки. Клетки с координатами за пределами лабиринта или с символом "0" не добавляются в список.

Затем следуют непосредственно функции с алгоритмами поиска.

Функция "greedy\_first\_search" реализует жадный алгоритм поиска пути в лабиринте. Алгоритм начинается с добавления стартовой точки в массив "frontier". Затем выбирается точка из "frontier", которая имеет минимальное значение эвристической функции, вычисленной от текущей позиции до целевой позиции. Если выбранная точка является конечной точкой, то функция возвращает список из пути, который включает текущую позицию. В противном случае, выбранная точка удаляется из массива "frontier", помещается в массив "visited", и для нее находятся соседние точки, которые не были посещены. Если соседняя точка еще не была посещена и не находится в массиве "frontier", то она добавляется в "frontier". Алгоритм повторяется до тех пор, пока не будет найден путь до целевой точки или "frontier" не окажется пустым.

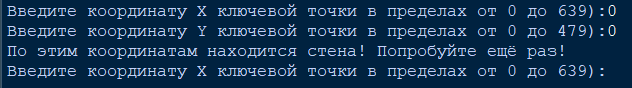
Чтобы уменьшить строковую нагрузку, для функции "greedy\_first\_search" была написана функция "get\_path", котораявосстанавливает и возвращает путь от стартовой точки до конечной точки. Функция принимает аргумент "current", который является списком всех посещенных точек в лабиринте. Функция использует обратный список "current" и удаляет все точки, которые не принадлежат пути. Остальные точки добавляются в новый путь. Это делается через проверку расстояния между первой точкой и последней точкой на пути. Если расстояние между двумя точками равно 1, то они являются соседними и первая точка добавляется в путь. В противном случае первая точка удаляется из списка "current", и алгоритм продолжает проверку следующей точки.

Функция "astar" реализует алгоритм A\*. Алгоритм начинается с пустого списка "path", словаря "distances" и словаря "previous". Текущая точка добавляется в "distances" со значением 0, а затем добавляется в "to\_visit". Пока есть элементы в "to\_visit", текущая точка "current" удаляется из него. Если "current" - это конечная точка, то путь до этой точки был найден. В противном случае находятся соседние точки, и для каждой из них вычисляется расстояние до "current" и обновляются значения в словаре "distances" и "previous". Если соседняя точка еще не находится в "distances", то она добавляется в "to\_visit". В конце, если конечная точка находится в словаре "previous", алгоритм проходит по списку "previous", начиная с конечной точки и до начальной точки, добавляя каждую точку в "path". В конце алгоритм возвращает путь "path".

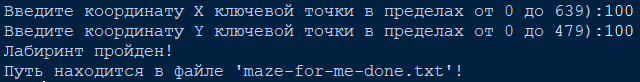
В функции `main` вызываются все вышеперечисленные функции. Жадный поиск выполняется от начальной точки до ключевой точки, A\* выполняется от ключевой точки до конечной точки. Затем путь, найденный жадным поиском, помечается символом ".", а путь, найденный A\*, помечается символом ",". Ключевая точка помечается символом "\*". Новый лабиринт записывается в файл `maze-for-me-done.txt`. В конце функция выводит уведомление о завершении поиска пути и местоположении нового лабиринта.

# Пример работы алгоритма

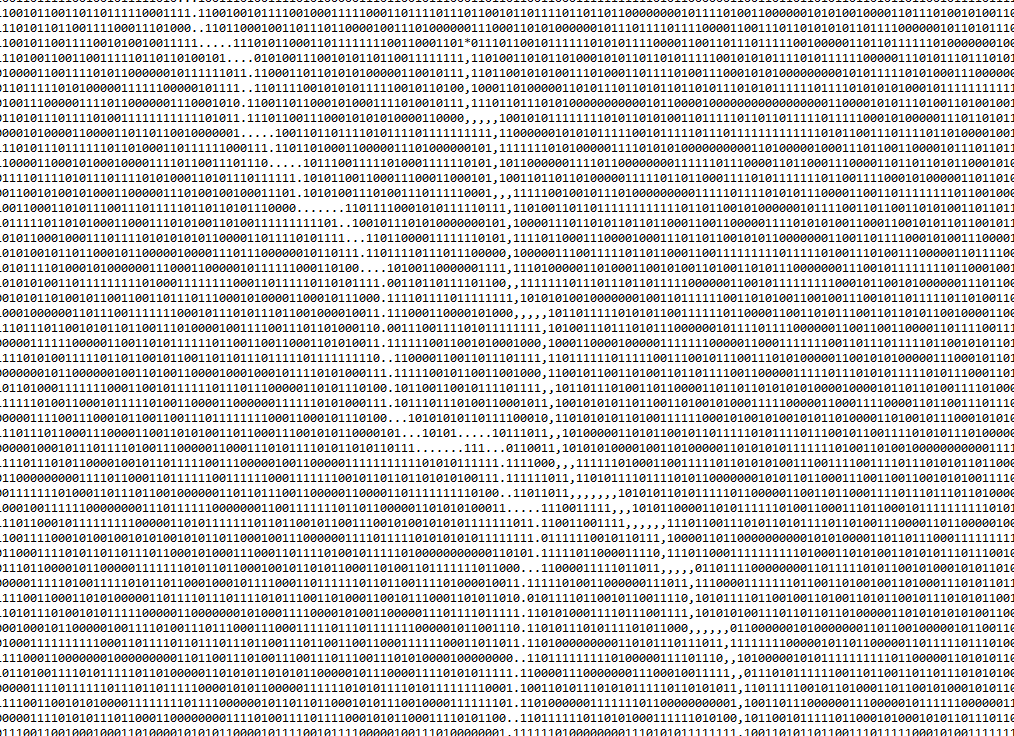
При запуске программы она просит ввести координаты ключа. Если введённые координаты соответствуют стене, то выводится ошибка с предложением повторить ввод.



Если координаты введены корректно, то программа находит путь и оповещает об этом.



Файл «maze-for-me-done.txt», получившийся в результате работы алгоритма, представляет собой лабиринт, где «1» - стена, «0» - коридор, «.» - траектория пути жадного алгоритма, «,» - траектория пути алгоритма А\*, «\*» - ключ. Так как содержимое файла имеет 640 строк и 480 столбцов, в качестве примера работы будет продемонстрирован участок у ключевой точки при введённых координатах ключа (100, 100).



# Заключение

В рамках выполнения данной работы были изучены теоретические основы структурного программирования, жадного алгоритма и алгоритма А\*.

Написана программа в структурной парадигме на языке Python, реализующая данные алгоритмы для поиска маршрута в лабиринте. Результаты обходов лабиринта и получившиеся маршруты были сохранены в файл.

Таким образом, цель курсовой работы достигнута. Результатом является программа, способная находить маршрут в лабиринте с помощью алгоритмов обхода графа с сохранением полученных маршрутов в файле. Работа программы была проверена на нескольких тестовых лабиринтах и показала хорошие результаты.

# Список литературы

1. Авачева Т. Г., Пруцков А. В. Современный взгляд на концепцию структурного программирования // Cloud of Science. — 2019. — Т. 6. — № 4. — С. 646–665.
2. Басараб М.А., Домрачева А.Б., Купляков В.М. Алгоритмы решения задачи быстрого поиска пути на географических картах. / Басараб М.А., Домрачева А.Б., Купляков В.М. - Текст: электронный // Инженерный журнал: наука и инновации. - 2013. - № 11. – с. 8 - URL: <https://cyberleninka.ru/article/n/algoritmy-resheniya-zadachi-bystrogo-poiska-puti-na-geograficheskih-kartah/viewer>
3. Жадные алгоритмы// Habr/[Электронный ресурс]. URL: <https://habr.com/en/articles/120343/> (Дата обращения: 03.05.2023)
4. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е. — М.: Вильямс, 2005. — 1296 с.
5. Макконнелл С. Совершенный код. Мастер класс / Пер. с англ - М. : Издательство «Русская редакция», 2010. — 896 стр.
6. Структурное , процедурное программирование, Переход к ООП и сравнение// Центр исследования искусственного интеллекта "ЕЦИИИ"/ [Электронный ресурс]. URL: <https://intellect.icu/strukturnoe-protsedurnoe-programmirovanie-perekhod-k-oop-i-sravnenie-185> (Дата обращения: 03.05.2023)
7. A\* // Википедиа / [Электронный ресурс]URL: [https://ru.wikipedia.org/wiki/A\*](https://ru.wikipedia.org/wiki/A*) (Дата обращения: 05.05.2023)
8. Easy A \* (звезда) Поиск пути // digitrain.ru / [Электронный ресурс]. URL:<https://digitrain.ru/articles/337034/> (Дата обращения: 05.05.2023)

# Приложения

**Приложение 1**

**Листинг программы**

def readMaze(file="maze-for-u.txt"):

maze = [list(line.replace("#", "1").replace(" ", "0")) for line in open(file).read().split("\n")[:-1]]

return maze

def key(maze):

while 1:

try:

x = int(input(f"Введите координату X ключевой точки в пределах от 0 до {len(maze)-1}):"))

if x >= 0 and x < len(maze):

y = int(input(f"Введите координату Y ключевой точки в пределах от 0 до {len(maze[0]) - 1}):"))

if y >= 0 and y < len(maze[0]):

if str(maze[x][y]) == "1":

print("По этим координатам находится стена! Попробуйте ещё раз!")

else:

return (x, y)

else:

print("Введённое значение выходит за пределы! Попробуйте ещё раз!")

continue

else:

print("Введённое значение выходит за пределы! Попробуйтеещёраз!")

continue

except Exception:

print("Вы ввели неверное значение! Попробуйте ещё раз.")

def end(maze):

for y in range(len(maze[0])):

if maze[len(maze)-1][y] == '0':

return (len(maze)- 1,y)

def start(maze):

for y in range(len(maze[0])):

if maze[0][y] == '0':

return (0,y)

def heuristic(a, b):

return ((a[0] - b[0]) \*\* 2 + (a[1] - b[1]) \*\* 2) \*\* 0.5

def get\_neighbors(maze, pos):

row, col = pos

candidates = [

(row - 1, col),

(row + 1, col),

(row, col - 1),

(row, col + 1)

]

result = []

for r, c in candidates:

if 0 <= r < len(maze) and 0 <= c < len(maze[0]) and maze[r][c] == "0":

result.append((r, c))

return result

def greedy\_first\_search(maze, start, goal):

frontier = [start]

visited = []

currentList = [start]

while len(frontier) != 0:

current = min(frontier, key=lambda n: heuristic(n, goal))

currentList.append(current)

if current == goal:

return get\_path(currentList)

frontier.remove(current)

visited.append(current)

neighbors = get\_neighbors(maze, current)

for neighbor in neighbors:

if neighbor not in visited and neighbor not in frontier:

frontier.append(neighbor)

def get\_path(current):

path = [current[-1]]

current = list(reversed(current))

i = len(current)

for j in range(i-1):

if ((current[0][0]-path[-1][0])\*\*2 + (current[0][1]-path[-1][1])\*\*2)\*\*(1/2) == 1:

path.append(current[0])

del current[0]

else:

del current[0]

return list(reversed(path))

def astar(maze, start, end):

path = []

distances = {start: 0}

previous = {}

to\_visit = [start]

while to\_visit:

current = to\_visit.pop(0)

if current == end:

break

neighbors = get\_neighbors(maze, current)

for neighbor in neighbors:

distance = distances[current] + 1

if neighbor not in distances or distance < distances[neighbor]:

distances[neighbor] = distance

previous[neighbor] = current

to\_visit.append(neighbor)

to\_visit.sort(key=lambda x: distances[x] + heuristic(x, end))

if end in previous:

current = end

while current != start:

path.insert(0, current)

current = previous[current]

path.insert(0, start)

return path

def main():

maze = readMaze()

key\_coords = key(maze)

greedyPath = greedy\_first\_search(maze,start(maze),key\_coords)

astarPath = astar(maze, key\_coords, end(maze))

for point in greedyPath:

maze[point[0]][point[1]] = "."

for point in astarPath:

maze[point[0]][point[1]] = ","

maze[key\_coords[0]][key\_coords[1]] = "\*"

with open("maze-for-me-done.txt", "w") as f:

f.write('\n'.join([''.join(map(str, line)) for line in maze]))

print("Лабиринтпройден!\nПутьнаходитсявфайле 'maze-for-me-done.txt'!")

main()