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 Перевезенцев А.С.

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………………………………………………………………………..15

# 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].

**Как работает алгоритм «A\*»**

Посетив одну конкретную вершину, алгоритм «A Star» перед переходом к следующей исследует все соседние вершины. Все вершины алгоритм разделяет на 3 категории:

1. Неизвестные вершины. Это те, которые не были еще посещены и пока что даже не найдены. Получается, что и путь к ним пока остается загадкой. Таким образом, изначально все вершины, кроме стартовой, будут в этой категории.
2. Известные вершины. Это те вершины, о которых уже известно алгоритму и уже даже известен путь к ним. Такие вершины сохраняются в «списке алгоритма» и становятся в очередь для их посещения и исследования. Из этого списка исследуются те вершины, которые считаются наиболее перспективными.
3. Исследованные вершины. В эту категорию попадают те вершины, которые уже были посещены алгоритмом «A Star». К этим вершинам известен самый короткий путь, поэтому они попадают в «закрытый список» — этот список нужен для того, чтобы исключить многократное исследование одних и тех же вершин.

Когда одна из вершин становится полностью исследованной, она попадает в категорию «исследованные вершины», а все ее соседи попадают в категорию «известные вершины» и становятся годными для исследования. На каждой уже исследованной вершине устанавливается указатель до той уже исследованной вершины, к которой у нее будет кратчайший путь.

Алгоритм «A Star» завершает свою работу только в том случае, если конечная вершина переносится в категорию «исследованные вершины». В этом случае уже будет весь список исследованных вершин, а на каждой из них будет стоять указатель с кратчайшим путем. Поэтому несложно будет по указателям отследить кратчайший путь от конечной вершины до начальной.

Алгоритм «A Star» находит кратчайший путь между вершинами, основываясь на стоимости и «весе» ребер. Поэтому путь, который находит «A Star», можно по праву назвать «самым быстрым» или «самым простым». По этой причине алгоритм «A Star» очень часто применяется как раз для планирования кратчайших путей, его также часто применяют в играх.

Однако его применение ограничивается его главным недостатком — потребностью в большом количестве памяти, потому что алгоритм «A Star» хранит всю информацию об известных и исследованных вершинах [9].

# 1.Теоретическая часть

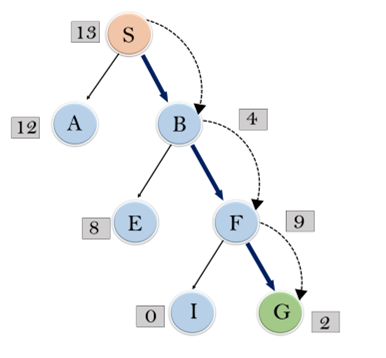
Для реализации алгоритма Greedy Best-First необходимо разобраться как он работает.

Жадный поиск работает, оценивая стоимость каждого возможного пути, а затем расширяя путь с наименьшей стоимостью. Этот процесс повторяется до тех пор, пока цель не будет достигнута.

Алгоритм использует эвристическую функцию, чтобы определить, какой путь является наиболее перспективным. Эвристическая функция учитывает стоимость текущего пути и расчетную стоимость оставшихся путей. Если стоимость текущего пути ниже оценочной стоимости остальных путей, то выбирается текущий путь.

Недостатком алгоритма Greedy Best-First является его низкая точность. Иногда он может выбрать неоптимальный путь из-за эвристической функции, которая не всегда оценивает расстояние между двумя вершинами точно. Также алгоритм может попадать в локальные минимумы и не находить оптимальный путь.

Рассмотрим приведенную ниже задачу поиска, и пройдем ее, используя жадный поиск с поиском по первому наилучшему. На каждой итерации каждый узел расширяется с использованием функции оценки. (Рис.1)



*Рисунок 1. Алгоритм поиска*

**Для чего нужен жадный алгоритм**

Вот причины использования жадного подхода:

1. У жадного подхода есть несколько компромиссов, которые могут сделать его пригодным для оптимизации.
2. Одна из важных причин состоит в том, чтобы немедленно найти наиболее выполнимое решение. В задаче выбора действий (поясняется ниже), если до завершения текущего действия можно выполнить больше действий, эти действия могут быть выполнены в одно и то же время.
3. Другая причина заключается в рекурсивном разделении проблемы на основе условия без необходимости объединения всех решений.

В задаче выбора действий шаг «рекурсивное разделение» достигается путем сканирования списка элементов только один раз и рассмотрения определенных действий.

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].

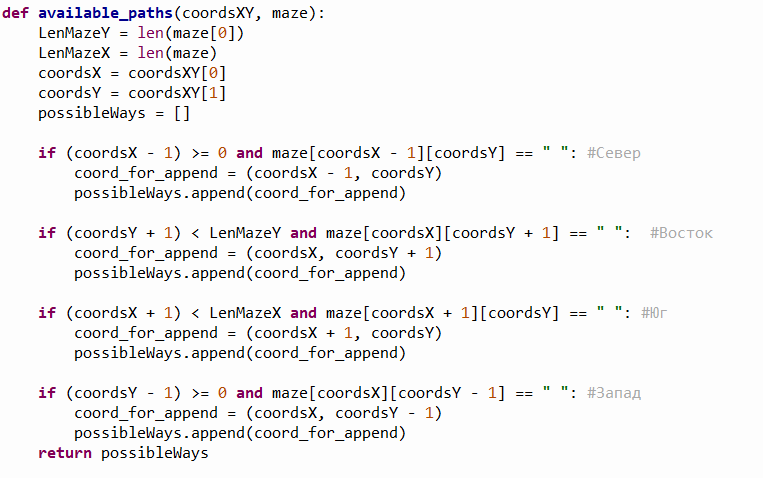
Порядок обхода вершин определяется эвристической функцией «расстояние + стоимость» (обычно обозначаемой как f(x)). Эта функция — сумма двух других: функции стоимости достижения рассматриваемой вершины (x) из начальной (обычно обозначается как g(x) и может быть как эвристической, так и нет), и функции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

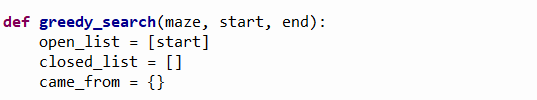
# Реализация алгоритма

**Жадный алгоритм**

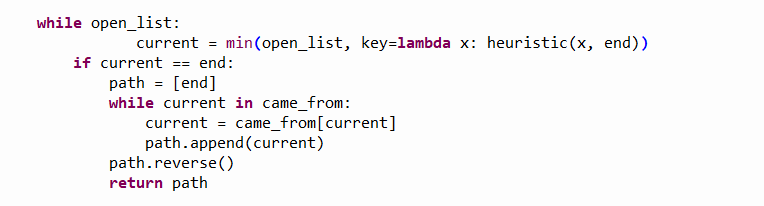


Инициализируем список открытых и закрытых вершин

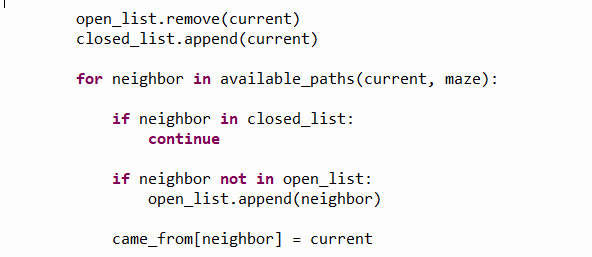
Инициализируем словарь, в котором будем хранить "родителей" вершин



Ищем путь от начала до конца

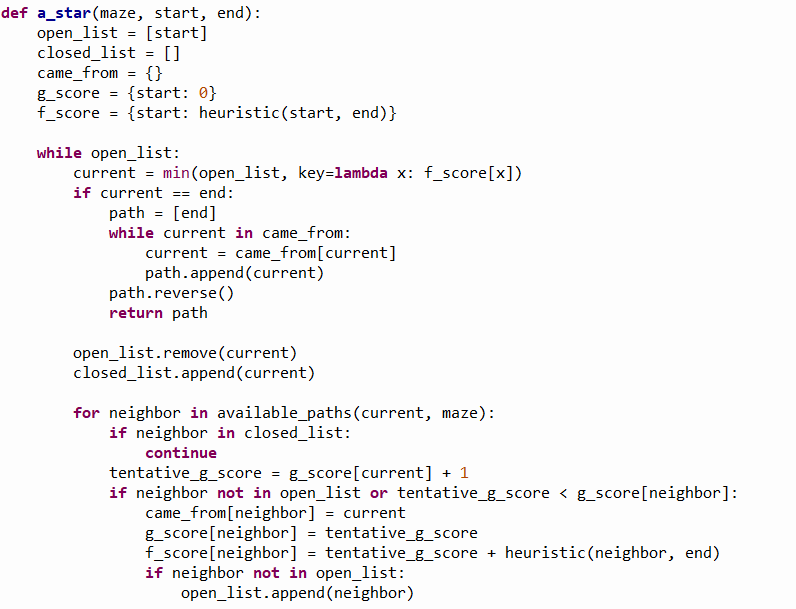
Выбираем из списка открытых вершин вершину с наименьшей эвристической оценкой. Если мы нашли конечную точку, составляем путь и возвращаем его

Перемещаем текущую вершину из списка открытых в список закрытых вершин. Ищем все соседние вершины текущей вершины. Если соседняя вершина уже в списке закрытых, пропускаем ее. Если соседняя вершина еще не в списке открытых, добавляем ее туда. Добавляем текущую вершину как "родителя" для соседней вершины.



Если жадный алгоритм не находит путь, функция возвращает None

**Алгоритм astar:**

****

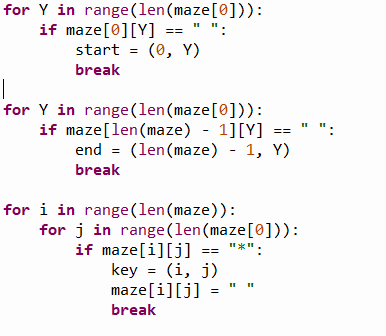
Если A\* не находит путь, функция возвращает None



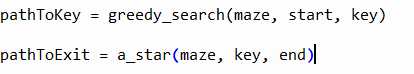
Читаем лабиринт из файла



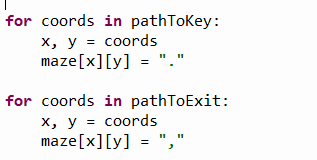
Ищем начальную, конечную точки и точку с ключом в лабиринте



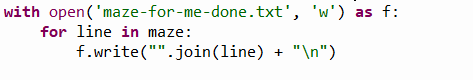
Ищем путь от начала до точки с ключом с помощью жадного алгоритма. Ищем путь от точки с ключом до конца с помощью алгоритма A\*



От точки-ключа до выхода "." , от точки-ключа до выхода ","



Записываем измененный лабиринт в файл



Здесь мы можем увидеть, что из начала лабиринта идут точки, до места, где находится ключ, а после нахождения ключа уже идут запятые, до выхода из лабиринта. (Рис.2)

# 

# 

*Рисунок 2. Участок итогового лабиринта*

# 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 и загружены для проверки наставником этой работы.

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**

***Мой программный код:***

def available\_paths(coordsXY, maze):

LenMazeY = len(maze[0])

LenMazeX = len(maze)

coordsX = coordsXY[0]

coordsY = coordsXY[1]

possibleWays = []

if (coordsX - 1) >= 0 and maze[coordsX - 1][coordsY] == " ": #Север

coord\_for\_append = (coordsX - 1, coordsY)

possibleWays.append(coord\_for\_append)

if (coordsY + 1) < LenMazeY and maze[coordsX][coordsY + 1] == " ": #Восток

coord\_for\_append = (coordsX, coordsY + 1)

possibleWays.append(coord\_for\_append)

if (coordsX + 1) < LenMazeX and maze[coordsX + 1][coordsY] == " ": #Юг

coord\_for\_append = (coordsX + 1, coordsY)

possibleWays.append(coord\_for\_append)

if (coordsY - 1) >= 0 and maze[coordsX][coordsY - 1] == " ": #Запад

coord\_for\_append = (coordsX, coordsY - 1)

possibleWays.append(coord\_for\_append)

return possibleWays

def greedy\_search(maze, start, end):

open\_list = [start]

closed\_list = []

came\_from = {}

while open\_list:

current = min(open\_list, key=lambda x: heuristic(x, end))

if current == end:

path = [end]

while current in came\_from:

current = came\_from[current]

path.append(current)

path.reverse()

return path

open\_list.remove(current)

closed\_list.append(current)

for neighbor in available\_paths(current, maze):

if neighbor in closed\_list:

continue

if neighbor not in open\_list:

open\_list.append(neighbor)

came\_from[neighbor] = current

return None

def a\_star(maze, start, end):

open\_list = [start]

closed\_list = []

came\_from = {}

g\_score = {start: 0}

f\_score = {start: heuristic(start, end)}

while open\_list:

current = min(open\_list, key=lambda x: f\_score[x])

if current == end:

path = [end]

while current in came\_from:

current = came\_from[current]

path.append(current)

path.reverse()

return path

open\_list.remove(current)

closed\_list.append(current)

for neighbor in available\_paths(current, maze):

if neighbor in closed\_list:

continue

tentative\_g\_score = g\_score[current] + 1

if neighbor not in open\_list or tentative\_g\_score < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = tentative\_g\_score

f\_score[neighbor] = tentative\_g\_score + heuristic(neighbor, end)

if neighbor not in open\_list:

open\_list.append(neighbor)

return None

def heuristic(current, end):

return abs(current[0] - end[0]) + abs(current[1] - end[1])

with open('maze-for-u.txt', 'r') as f:

maze = [list(line.strip()) for line in f.readlines()]

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

if maze[0][Y] == " ":

start = (0, Y)

break

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

if maze[len(maze) - 1][Y] == " ":

end = (len(maze) - 1, Y)

break

for i in range(len(maze)):

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

if maze[i][j] == "\*":

key = (i, j)

maze[i][j] = " "

break

pathToKey = greedy\_search(maze, start, key)

pathToExit = a\_star(maze, key, end)

for coords in pathToKey:

x, y = coords

maze[x][y] = "."

for coords in pathToExit:

x, y = coords

maze[x][y] = ","

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

for line in maze:

# f.write("".join(line) + "\n")