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 Помазов С.П.

Проверил Морозов Н.С.

**Содержание**

[Введение 3](#_Toc134827179)

[Задачи 4](#_Toc134827180)

[1.Теоретическая часть 5](#_Toc134827181)

[Алгоритм A\* 5](#_Toc134827182)

[Жадный алгоритм 6](#_Toc134827183)

[2.Реализация алгоритмов 8](#_Toc134827184)

[Реализация алгоритма A\* 8](#_Toc134827185)

[Реализация жадного алгоритма 10](#_Toc134827186)

[Полученный результат 13](#_Toc134827187)

[Заключение 14](#_Toc134827188)

[Список литературы 15](#_Toc134827189)

[Приложение 1 16](#_Toc134827190)

[Листинг программы 16](#_Toc134827191)

# Введение

Алгоритмы поиска пути являются одной из ключевых областей в области компьютерных наук, и они широко применяются в различных областях, таких как компьютерные игры, робототехника, геоинформационные системы и многие другие. Эти алгоритмы позволяют находить оптимальный или приближенный путь между двумя вершинами в графе или на сетке.

Один из наиболее распространенных методов поиска пути - алгоритм A\*[1]. Он сочетает в себе эвристическую оценку и прямой поиск, позволяя находить оптимальный путь с учетом стоимости перемещения и примерной оценки расстояния до цели. Другой популярный алгоритм - жадный алгоритм, который принимает локально оптимальные решения на каждом узле, надеясь, что это приведет к глобально оптимальному результату.

Не менее популярным является жадный алгоритм (greedy algorithm) [2]. Основной принцип жадного алгоритма заключается в том, чтобы на каждой вершине выбирать наиболее выгодное решение из доступных вариантов, основываясь на заданном критерии оптимальности. Жадный алгоритм принимает решения мгновенно, без просмотра будущих шагов и без обратного взгляда на предыдущие принятые решения. Несмотря на некоторые ограничения, жадные алгоритмы широко используются в различных областях, включая оптимизацию, комбинаторику, анализ данных и многие другие, благодаря своей простоте и эффективности во многих случаях. Однако следует отметить, что жадные алгоритмы не всегда гарантируют нахождение глобально оптимального решения. Хотя они могут быть эффективны во многих задачах и давать приемлемые результаты, в некоторых случаях они могут приводить к недостаточно оптимальным или неверным решениям.

**Цель работы:** реализовать алгоритмы поиска пути: жадный алгоритм от старта до ключа, A\* от ключа до финиша.

# Задачи

* Изучить алгоритмы построения маршрута в графе;
* Выделить особенности реализации жадного алгоритма и A\*
* Подготовить исходные данные: лабиринт, координаты точек для посещения при обходе;
* Реализовать алгоритмы с заданными параметрами;
* Сохранить результаты обходов лабиринта и получившиеся маршруты в файл.

# 1.Теоретическая часть

## Алгоритм A\*

Алгоритм, известный как A\*, представляет собой широко используемый метод для нахождения наименьшего пути в графе. Основная идея заключается в том, чтобы оценить расстояние от текущей вершины до целевой вершины с помощью эвристической функции. Эта функция позволяет оценить стоимость пути от текущей вершины до цели через остальные вершины графа. Алгоритм A\* учитывает как пройденное расстояние до текущей вершины, так и эвристическую оценку до цели. В отличие от жадного алгоритма, он выбирает на каждом шаге не только наилучший вариант, но и учитывает пройденный путь до текущей вершины.

Спектр применения алгоритма A\* очень разнообразен. Его используют как в создании простых игр, так и в проектировании серьёзных алгоритмов подбора маршрутов для навигаторов и беспилотных автомобилей. Кроме того, алгоритм A\* может быть использован для решения задач оптимизации, таких как нахождение минимального количества операций, необходимых для перемещения объекта из одной точки в другую.

За пример работы алгоритма можно взять любой граф, с определенным количеством вершин и путей, которые имеют свою стоимость за проход через такой путь.

Исходя из всего вышесказанного можно сделать вывод, что A\* - очень эффективный алгоритм поиска пути, который учитывает все препятствия и ограничения, возникающие на пути, при этом оценивая количество операций, необходимых для достижения цели. Именно поэтому этот алгоритм очень востребован в множестве областей разработок.

## Жадный алгоритм

Жадный алгоритм - это метод решения задачи, при котором на каждом шаге выбирается локально оптимальное решение в надежде на получение глобально оптимального результата. Жадный алгоритм принимает решения на основе мгновенной выгоды без учета будущих последствий.

Основная идея жадного алгоритма заключается в том, что на каждом шаге выбирается наилучшее решение согласно определенному критерию. Это может быть, например, выбор наименьшей стоимости, наибольшей выгоды или наименьшего времени.

Жадный алгоритм обладает некоторыми преимуществами, такими как простота реализации и быстрота работы. Он может быть полезен в случаях, когда необходимо быстро получить приемлемое решение, даже если оно не является оптимальным.

Однако жадный алгоритм не всегда гарантирует достижение оптимального результата. Иногда он может привести к локально оптимальному, но не глобально оптимальному решению. Это происходит из-за того, что жадный алгоритм не учитывает возможные последствия своих решений на будущих шагах.

Чтобы использовать жадный алгоритм эффективно, важно правильно определить критерий выбора на каждом шаге и убедиться, что он приводит к приемлемому результату для данной задачи. В некоторых случаях жадный алгоритм может использоваться как первоначальное приближение для более сложных оптимизационных методов.

Пример использования жадного алгоритма – выдача сдачи на кассе:  
Предположим что мы хотим купить газировку за 68 рублей, и протягивает кассиру купюру номиналом 100 рублей. Следовательно, кассир должен дать нам сдачу – 32 рубля. В кассе используются монеты номиналом 1 рубль, 2 рубля, 3 рубля, 5 рублей и 10 рублей. Сначала проверяется самая крупная монета по номиналу. 10 рублей <32 рубля, значит нам эта монета подходит, и мы можем её взять. Эта процедура повторяется ещё два раза, пока кассир не выдаст 30 рублей и получит остаток 2 рубля. Далее он проверяет более мелкую монету – 5 рублей. 5> 2 – не подходит, берем монету еще меньшей стоимости – 2 рубля. 2 = 2 – подходит. Следовательно, кассир дал нам 3 монеты по 10 рублей каждая и одну монету с номиналом 2 рубля [3].

В целом, жадный алгоритм представляет собой простой и быстрый подход к решению задач, но требует осторожного выбора критерия принятия решений, чтобы достичь желаемого результата.

# 2.Реализация алгоритмов

## Реализация алгоритма A\*

**Инициализация:**

* Установите начальную вершину и целевую вершину.
* Присвойте начальной вершине стоимость 0.
* Установите для начальной вершины эвристическую оценку расстояния до цели.
* Создайте открытый список для отслеживания вершин, которые нужно рассмотреть, и закрытый список для отслеживания уже пройденных вершин.

**Шаги алгоритма:**

* Пока открытый список не пуст:
  + Выберите вершину с наименьшей общей оценкой из открытого списка (это будет вершина с наименьшим значением суммы стоимости пути и эвристической оценки).
  + Поместите выбранную вершину в закрытый список.
  + Если выбранная вершина - целевая вершина, значит, путь найден, алгоритм завершается.
  + В противном случае:
    - Рассмотрите все соседние вершины текущей вершины.
    - Для каждой соседней вершины:
      * Вычислите стоимость пути от начальной вершины до соседней вершины, добавив стоимость пути от начальной вершины до текущей и стоимость ребра между текущей и соседней вершинами.
      * Если соседняя вершина уже находится в открытом или закрытом списке и новая оценка пути больше, чем уже имеющаяся оценка, пропустите эту вершину.
      * В противном случае:
        + Обновите стоимость пути и эвристическую оценку соседней вершины.
        + Установите текущую вершину как родительскую для соседней вершины.
        + Если соседняя вершина не находится в открытом списке, добавьте ее в открытый список.

**Завершение:**

* Если открытый список пуст и целевая вершина не достигнута, значит, путь от начальной вершины до целевой недоступен, алгоритм завершается без результата.

**Восстановление пути:**

* Если целевая вершина достигнута, нужно восстановить путь от начальной вершины до цели, переходя от целевой вершины к ее родительским вершинам, пока не достигнем начальную вершину.

**Пример использования A\*:**

Допустим, мы хотим найти кратчайший путь от вершины А до вершины J (рис. 1).

Из А есть только два пути — B и F. Вычислим стоимость: f(B) = 8 + 6 = 14 и f(F) = 3+6 =9. Следовательно, нам нужно перейти в вершину F — алгоритм продолжит работу отсюда.

Из точки F есть два пути — G и H. Снова вычислим стоимость:  f(G) = 4 +5 = 9 and f(H) = 10 + 3 = 13. Значит, мы переходим в точку G.

Следуя пути I—J, получаем следующее:  f(I) = 7 + 1 = 8 и f(J) = 10. Так как все значения, следующие за вершиной F, меньше f(B), возвращаться к вершине B не имеет смысла.

Но допустим другой вариант. Предположим, что f(I) больше f(B) при прохождении через F и G (f(I) > 14). В этом случае алгоритм A\* прекратит дальнейшую работу и переместится в вершину В. Но, так как f(C) > f(I), работа алгоритма продолжается именно в вершине I [4].

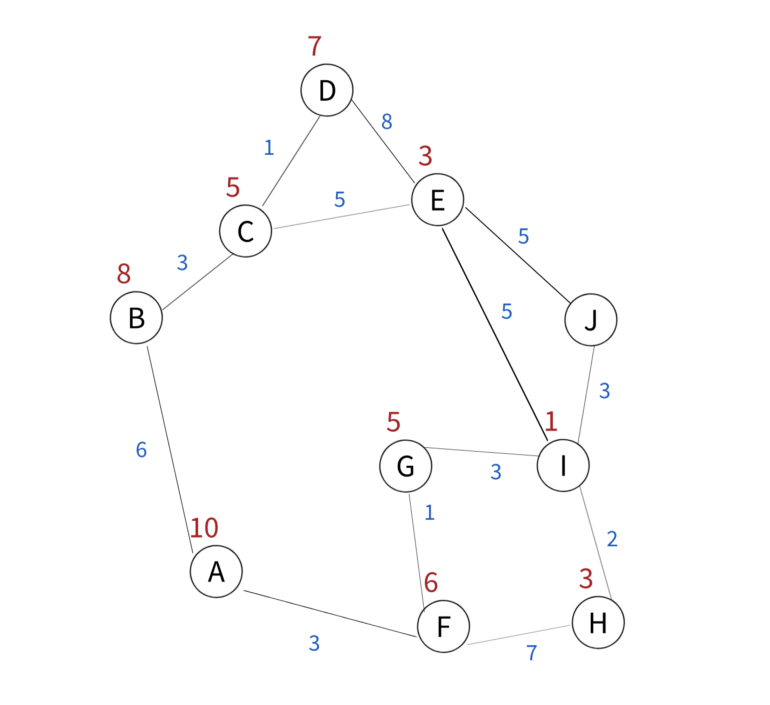


Рисунок 1

## Реализация жадного алгоритма

**Инициализация:**

* Куча (очередь с приоритетом – priority\_queue), используемая для хранения вершин с их приоритетами. Изначально пуста.
* Множество, используемое для отслеживания посещенных вершин (visited). Изначально пусто.
* В очередь priority\_queue добавляется стартовая вершина с приоритетом 0.

**Основной цикл алгоритма:**

* Пока priority\_queue не пуста:
  + Извлекается вершина с наименьшим приоритетом из priority\_queue в переменные с текущей ценой и текущей стоимостью.
  + Если текущая позицияравна целевой позиции, алгоритм прерывается, так как путь найден.
  + Если текущая позиция уже была посещена (находится в visited), цикл продолжается со следующей итерации.
  + Текущая позиция добавляется в visited для отметки посещения.
  + Создается список, содержащий возможные направления движения (в данном случае, вверх, вниз, влево, вправо).
  + Для каждого направления (dx, dy) в directions:
    - Вычисляются новые координаты x и y путем суммирования текущих координат с соответствующими значениями (dx, dy).
    - Проверяется, что новые координаты x и y находятся в пределах лабиринта и не содержат препятствия
    - Вычисляется приоритет для новой позиции на основе эвристической функции, которая оценивает оставшееся расстояние до целевой позиции.
    - Новая позиция (x, y) с приоритетом добавляется в приоритетную очередь.

**Завершение и результат:**

* + Если цикл прервался, значит, целевая позиция достигнута, и алгоритм завершается.
  + Возвращается множество visited, которое содержит все посещенные вершины в процессе алгоритма.

**Пример использования с приоритетной очередью:**

Даны заявки на проведение занятий в некоторой аудитории. В каждой заявке указаны начало и конец занятия (si и fi для i-ой заявки). Нужно из всех заявок оставить как можно больше так, чтобы они не пересекались. При этом если одна заявка закончилась во время t, а следующая началась во время t, то их можно ставить подряд.

**Решение:**

Здесь жадность становится не такой уже очевидной, потому что неясно в каком порядке рассматривать заявки, те непонятно как “жадно” их набирать.

Давайте посмотрим на **самую первую по времени конца заявку**. Заметьте, что нам всегда выгодно включить её в оптимальный ответ - она заканчивается раньше всех остальных, а поэтому если в оптимальном ответе самая первая заявка - другая, мы можем безболезненно заменить её на самую первую по времени конца, и новых пересечений не появится, так как мы просто сдвинули самую первую заявку еще левее.

Раз всегда есть оптимальный ответ, в котором выбрана эта самая левая по времени конца заявка, давайте её возьмем, и выберем **самую первую по времени конца заявку из оставшихся, не пересекающихся с той**.

Из такого рассуждения про одну самую левую по времени конца заявку следует сразу и общее жадное решение задачи - нужно идти слева направо по заявкам, которые отсортированы по времени конца, и брать новую, если можем, то есть если её начало не раньше, чем конец самой последней уже выбранной [5].

## Полученный результат

Результат работы алгоритма можно увидеть на рис. 2.

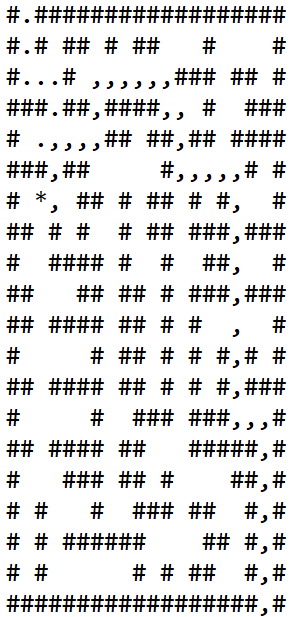


Рисунок 2

# Заключение

В ходе работы были достигнуты поставленные цели и выполнены задачи, включающие изучение алгоритмов построения маршрута в графе, выделение особенностей реализации жадного алгоритма и алгоритма A\*, подготовку исходных данных, реализацию алгоритмов с заданными параметрами и сохранение результатов в файл.

Конечно, у обоих, доставшихся мне алгоритмов есть много сходств, но также есть и очень много различий. Лично я выделил для себя следующие особенности:

Алгоритм A\* комбинирует эвристическую оценку и прямой поиск, что позволяет находить оптимальный путь с учетом стоимости перемещения и приближенной оценки расстояния до цели. Реализация алгоритма A\* требует использования структур данных, таких как множество и словарь, и регулярных обновлений текущих значений g- и f-оценок для каждой вершины.

Жадный алгоритм, в свою очередь, не учитывает будущие шаги и не просматривает предыдущие решения. Жадные алгоритмы принимают мгновенные решения, опираясь на заданный критерий оптимальности. Они обладают простотой и эффективностью во многих случаях, но не гарантируют нахождение глобально оптимального решения. Поэтому важно оценивать их применимость к конкретной задаче и учитывать возможность получения недостаточно оптимальных результатов.

# Список литературы

1. Простое объяснение алгоритмов поиска пути и A\* / Хабр [Электронный ресурс]. URL: (<https://habr.com/ru/articles/444828/>) (Дата обращения: 05.05.2023)
2. Жадные алгоритмы - Алгоритмика [Электронный ресурс]. URL: (https://ru.algorithmica.org/cs/combinatorial-optimization/greedy/) (Дата обращения: 05.05.2023)
3. Алгоритмы – часть 1. Жадные алгоритмы, алгоритм Дейкстры [Электронный ресурс]. URL: (<https://webdevblog.ru/zhadnye-algoritmy-chast-1-algoritm-dejkstry/>) (Дата обращения: 06.05.2023)
4. Алгоритм A\* и его реализация на Python [Электронный ресурс]. URL: (https://pythonist.ru/algoritm-a-star-i-ego-realizacziya-na-python/) (Дата обращения: 06.05.2023)
5. Жадный алгоритм – Tinkoff Generation [Электронный ресурс]. URL: (<https://algorithmica.org/tg/greedy>) (Дата обращения: 06.05.2023)

# Приложение 1

## Листинг программы

import heapq

def read\_maze(file):   
 with open(file, 'r') as f:  
 maze = [[char for char in line.strip()] for line in f]  
 return maze

# Поиск координат стартовой точки, ключа и выхода

def find\_coordinates(maze):  
 start = (0, maze[0].index(' '))  
 key = None  
 exit = (len(maze)-1, maze[len(maze)-1].index(' '))  
 for row in range(len(maze)):  
 if '\*' in maze[row]:  
 key = (row, maze[row].index('\*'))  
 break  
 return start, key, exit

#Эвристическая стоимость

def heuristic\_cost(start, end):  
 return abs(end[0] - start[0]) + abs(end[1] - start[1])

# Жадный алгоритм поиска пути до ключа  
# он отображает посещения всех точек, в которых побывал аватар в поисках ключа

def greedy\_algorithm(maze, start, end):  
 pq = []   
 visited = set()  
 heapq.heappush(pq, (0, start))  
 while pq:  
 current\_cost, current\_pos = heapq.heappop(pq)  
 if current\_pos == end:  
 break  
 if current\_pos in visited:  
 continue  
 visited.add(current\_pos)  
 directions = [(0, -1), (0, 1), (-1, 0), (1, 0)]  
 for dx, dy in directions:  
 x, y = current\_pos[0] + dx, current\_pos[1] + dy  
 if 0 <= x < len(maze) and 0 <= y < len(maze[0]) and maze[x][y] != '#':  
 priority = heuristic\_cost(end, (x, y))  
 heapq.heappush(pq, (priority, (x, y)))  
 return visited  
  
#А\_star алгоритм поиска кратчайшего пути  
#оцениваем стоимость вариантов путей

def a\_star(maze, start, end):  
 open\_set = set([start])  
 came\_from = {} #предыдущая точка  
 g\_score = {start: 0}  
 f\_score = {start: heuristic\_cost(start, end)}

while open\_set:  
 current = min(open\_set, key=lambda x: f\_score[x])  
 if current == end:  
 path = [current]  
 while current in came\_from:  
 current = came\_from[current]  
 path.append(current)  
 path.reverse()  
 return path  
 open\_set.remove(current)  
 for neighbor in get\_neighbors(current, maze):  
 tentative\_g\_score = g\_score[current] + 1  
 if neighbor not in g\_score or tentative\_g\_score < g\_score[neighbor]:  
 came\_from[neighbor] = current  
 g\_score[neighbor] = tentative\_g\_score  
 f\_score[neighbor] = tentative\_g\_score + heuristic\_cost(neighbor, end)  
 if neighbor not in open\_set:  
 open\_set.add(neighbor)  
 return None

#Функция для определения соседних точек, относительно текущей

def get\_neighbors(coord, maze):  
 directions = [(0, -1), (0, 1), (-1, 0), (1, 0)]  
 neighbors = []  
 for dx, dy in directions: #dx,dy - дельта направления  
 x, y = coord[0] + dx, coord[1] + dy  
 if 0 <= x < len(maze) and 0 <= y < len(maze[0]):  
 if maze[x][y] != "#":  
 neighbors.append((x, y))  
 return neighbors

#Функции для перевода путей в символы и записи в новый файл

def write\_maze\_greedy(maze, path, key):  
 for pos in path:  
 maze[pos[0]][pos[1]] = '.'  
 maze[key[0]][key[1]] = '\*'  
 with open('maze-for-me-done.txt', 'w') as f:  
 for row in maze:  
 f.write(''.join(row) + '\n')

def write\_maze\_astar(maze, path, key):  
 for pos in path:  
 maze[pos[0]][pos[1]] = ','  
 maze[key[0]][key[1]] = '\*'  
 with open('maze-for-me-done.txt', 'w') as f:  
 for row in maze:  
 f.write(''.join(row) + '\n')

#Основной код:

#Считываем лабиринт в массив

maze = read\_maze('maze-for-u.txt')

#Находим координаты (x,y) для Точки старта, Точки финиша и Точки ключа

start, key, exit = find\_coordinates(maze)

print(start,key,exit)

#Находим путь до ключа с помощью Жадного алгоритма

path1 = greedy\_algorithm(maze,start,key)

#Находим путь от ключа до выхода с помощью A\*

path2 = a\_star(maze,key,exit)

#Записываем пути в новый текстовый документ "maze-for-me-done.txt" (создается автоматически)

write\_maze\_greedy(maze,path1,key)  
write\_maze\_astar(maze,path2,key)