МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Построение и анализ алгоритмов»

Тема: Жадный алгоритм и A*

Студент гр. 1304 Заика Т.П.

Преподаватель Шевелева А.М.

Санкт-Петербург
2023

Цель работы.

Изучить жадный алгоритм и A*, примененить их к задаче построения пути в ориентрированном графе.

Задание.

1. Разработайте программу, которая решает задачу построения пути в ориентированном графе при помощи жадного алгоритма. Жадность в данном случае понимается следующим образом: на каждом шаге выбирается последняя посещённая вершина. Переместиться необходимо в ту вершину, путь до которой является самым дешёвым из последней посещённой вершины. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес.

2. Разработайте программу, которая решает задачу построения кратчайшего пути в ориентированном графе методом A*. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес. В качестве эвристической функции следует взять близость символов, обозначающих вершины графа, в таблице ASCII.

Выполнение работы.

В ходе работы было определено, что все необходимые для решения функции можно сделать методами класса *Solution*. Рассмотрим методы и переменные класса:

1) Метод *__init__* инициализирует следующие переменные, использующиеся на протяжении всего решения каждым из методов:

graph — словарь, хранящий в качестве ключи строковое значение узла, а в качестве значения список из кортежей, в каждом из которых указан смежный узел и расстояние до него от узла, находящегося в ключе.

start_node — начальная вершина, от которой строим путь.

end_node — конечная вершина, до которой строим путь.

answer — строка ответа на задачу в требуемом формате.

way_found — булевая переменная-флаг, использующаяся для установки того, что путь был найден при решении задача жадным алгоримом.

2) Метод *__create_graph_from_input* создает представление графа в виде словаря из ввода пользователя, где ключ — узел графа, а значение — список кортежей, в каждом из которых указан смежный узел и расстояние до него от узла, находящегося в ключе.

3) Метод *__sort_graph_by_length* сортирует узлы у каждого узла графа по весу ребра.

4) Метод *__solve_by_greedy_algorythm* решает задачу методом жадного алгоритма. Для этого в параметрах метода мы передаем текущий узел и текущий построенный путь. Если мы нашли путь, то выходим из метода. Если мы установили, что текущий узел искомый, то формируем ответ и отмечаем, что путь найден, после чего выходим из метода. В остальных случаях мы для тех узлов, которые имеют свои узлы, проходимся по их узлам и берем очередной узел (узлы упорядоченны по весу ребра, поэтому вначале всегда берем самый дешевый узел), добавляя его в текущий пройденный путь, и рекурсивно продолжаем поиск искомого узла. Если мы находимся в висячем узле, который не является искомым, то выходим из метода без результата, возвращаемся к предыдущему узлы и идем к его следующему смежному узлу. Данный вариант жадного алгоритма основывается для понимании работы механизма рекурсии.

5) Метод *__compute_heuristic* считает значение функции эвристики (близость символов, обозначающих вершины графа, в таблице ASCII).

6) Метод *__solve_by_a_start_algorythm* решает задачу методом алгоритма A*. Для этого мы создаем очередь с приоритетом, а также два словаря: первый для хранения для очередного узла информации о том, из какого узла его можно достичь, а второй для хранения растояния от стартовой вершины до очередного узла. Сначала в очередь с приоритетом положим начальную вершину с минимальным приоритетом, а также установим, что в вершину нельзя прийти и расстояние до нее 0, т. к. она стартовая. Пока очередь не пуста, мы возьмем из нее очередной узел. Если это искомый узел, то прекратим цикл. В случае если узел имеет свои узлы, для каждого из его узлов подсчитаем обычное расстояние до узла. Если мы еще не записывали этот узел в словарь для хранения

расстояний или мы улучшили расстояние до этого узла, то запишем его в словарь для хранения расстояний, рассчитаем приоритет при помощи сложения начального расстояния и функции эвристики, добавим в очередь с приоритетом данный узел с заданным приоритетом, и укажем в словаре для хранения информации о том, откуда достижимы данный узел, что он достижимы из рассматриваемого во внешнем цикле узла, у которого есть свои узлы. После выхода из очереди сформируем ответ: будем проходить от искомого узла до начального по словарю для хранения информации о том, откуда достижим данный узел, и перевернем найденный путь, чтобы он вел от стартового до конечного узла.

7) Метод *get_solution_by_greedy* задает последовательность действий (создание графа из пользовательского ввода, его сортировка, само решение) для решения задачи методом жадного алгоритма.

8) Метод *get_solution_by_a_start* задает последовательность действий (создаение графа из пользовательского ввода, само решение) для решения задачи методом алгоритма A*.

9) Метод *print_answer* печатает ответ на задачу в консоль.

Разработанный программный код см. в приложении А.

Выводы.

Исследован, изучен жадный алгоритм и A*, применины к задаче построения пути в ориентрированном графе. Оба алгоритма заключены в одном классе. Каждый из алгоритмом успешно проходит все тесты на платформе Stepik. Граф хранится в виде словаря. Жадный алгоритм реализован при помощи рекурсии, что позволяет получить легко читаемый код, основанный на понимании работы механизма рекурсии. Алгоритм A* использует структуру данных «очередь с приоритетом», а также словари, что позволяет быстро и эффективно решать поставленную задачу. Функцией эвристики для алгоритма A* является близость символов, обозначающих вершины графа, в таблице ASCII.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
from queue import PriorityQueue
# Вариант очереди, который извлекает открытые записи в порядке приоритета (сначала самый низкий).
from sys import stdin
# Стандартный ввод используется для всего интерактивного ввода


# Класс решения
class Solution:
    def __init__(self):
        """
        Инициализирует поля класса, необходимые для решения
        """
        self.graph = {}
        self.start_node = None
        self.end_node = None
        self.answer = ""
        self.way_found = False

    def __create_graph_from_input(self) -> None:
        """
        Создает граф в виде словаря из ввода пользователя
        """
        self.start_node, self.end_node = input().split()
        process_graph = {}

        for line in stdin:
            cur_node, next_node, distance = line.split()
            if not process_graph.get(cur_node):
                process_graph[cur_node] = [(next_node, float(distance))]
            else:
                process_graph[cur_node] += [(next_node, float(distance))]
        self.graph = process_graph

    def __sort_graph_by_length(self) -> None:
        """
        Сортирует узлы у каждого узла в графе по весу ребра
        """
        for node in self.graph:
            self.graph.update({node: sorted(self.graph[node], key=lambda n: n[1])})

    def __solve_by_greedy_algorythm(self, cur_node: str, cur_way: str) -> None:
        """
        Решает задачу методом жадного алгоритма
        :param cur_node: текущий узел
        :param cur_way: текущий построенный путь
        """
        if self.way_found:
```

```
                return
            if cur_node == self.end_node:
                self.answer = cur_way
                self.way_found = True
                return
            if self.graph.get(cur_node):
                for nodes in self.graph[cur_node]:
                    next_node = nodes[0]
                    next_way = cur_way + f"{next_node}"
                    self.__solve_by_greedy_algorythm(next_node,
next_way)

        def __compute_heuristic(self, cur_node: str) -> int:
            """
            Считает значении функции эвристики
            :param cur_node: текуший узел
            :return: значение функции эвристики
            """
            return abs(ord(self.end_node) - ord(cur_node))

        def __solve_by_a_star_algorythm(self) -> None:
            """
            Решает задачу методом алгоритма A*
            """
            frontier = PriorityQueue()
            frontier.put((0, self.start_node))
            came_from = {}
            cost_so_far = {}

            came_from[self.start_node] = None
            cost_so_far[self.start_node] = 0
            while not frontier.empty():
                cur_node = frontier.get()[1]
                if cur_node == self.end_node:
                    break
                if self.graph.get(cur_node):
                    for        next_node,        next_distance        in
self.graph.get(cur_node):
                        new_cost     =     cost_so_far[cur_node]     +
next_distance
                        if next_node not in cost_so_far or new_cost <
cost_so_far[next_node]:
                            cost_so_far[next_node] = new_cost
                            priority        =        new_cost        +
self.__compute_heuristic(next_node)
                            frontier.put((priority, next_node))
                            came_from[next_node] = cur_node

            cur_node = self.end_node
            self.answer = f"{self.end_node}"
            while cur_node != self.start_node:
                cur_node = came_from[cur_node]
                self.answer += cur_node
            self.answer = self.answer[::-1]

        def get_solution_by_greedy(self) -> None:
            """
            Задает последовательность действий для
```

```
            решения задачи методом жадного алгоритма
            """
            solver.__create_graph_from_input()
            solver.__sort_graph_by_length()
            self.__solve_by_greedy_algorythm(self.start_node,
self.answer + f"{self.start_node}")

        def get_solution_by_a_star(self) -> None:
            """
            Задает последовательность действий для
            решения задачи методом алгоримта A*
            """
            solver.__create_graph_from_input()
            self.__solve_by_a_star_algorythm()

        def print_answer(self) -> None:
            """
            Печатает ответ на задачу в консоль
            """
            print(self.answer)


    if __name__ == "__main__":
        solver = Solution()
        # solver.get_solution_by_greedy()
        solver.get_solution_by_a_star()
```