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*

Студентка гр. 1304 _____ Хорошкова А.С.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург

2023

Цель работы.

Построение пути в *ориентированном* графе при помощи жадного алгоритма и алгоритма A*.

Задание.

1. Жадный алгоритм.

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. Пример входных данных:

a e
a b 3.0
b c 1.0
c d 1.0
a d 5.0
d e 1.0

В первой строке через пробел указываются начальная и конечная вершины
Далее в каждой строке указываются ребра графа и их вес.

В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной. Для приведённых в примере входных данных ответом будет:

abcde

2. Алгоритм A*.

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.

Пример входных данных:

a e
a b 3.0
b c 1.0
c d 1.0
a d 5.0
d e 1.0

В первой строке через пробел указываются начальная и конечная вершины. Далее в каждой строке указываются ребра графа и их вес. В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной. Для приведённых в примере входных данных ответом будет:

ade

Выполнение работы.

Описание алгоритма.

Жадный алгоритм.

Сначала считывается граф и заполняется структура HashMap<Character, PriorityQueue<Edge>> так, чтобы каждая вершина служила ключом для исходящих из неё рёбер. В свою очередь рёбра заполняют очередь с приоритетом так, чтобы наверху всегда было ребро с наименьшим весом.

После чего запускается жадный алгоритм, который идёт по рёбрам сверху очереди с приоритетом, автоматически удаляя их во время прохода. В случае, если алгоритм зашёл в тупик, он возвращается на одно ребро назад и продолжает поиск. Алгоритм начинается с вершины start и заканчивается тогда, когда достигнет вершины end.

Алгоритм A*.

Сначала считывается граф и заполняется структура HashMap<Character, PriorityQueue<Edge>> так, чтобы каждая вершина служила ключом для исходящих из неё рёбер. После чего запускается алгоритм A*. В ходе алгоритма создаётся две структуры: одна сожержит просмотренные вершины, другая ожидающие просмотра в порядке очереди. Порядок очереди

определяется весом вершины и её эвристикой. Далее алгоритм по очереди просматривает всех детей вершин из второй структуры, после чего отправляет выршины в первую структуру просмотренных. Во время просмотра дети вершины либо отправляются в очередь (если они ещё не были просмотрены), либо обновляют данные просмотренных вершин (если найден более оптимальный путь), либо остаются без изменений. После восстанавливается результирующий путь с конечной до начальной по меткам о предыдущей вершине. Алгоритм начинается с вершины start и заканчивается тогда, когда достигнет вершины end.

Алгоритм успешно прошёл все тесты на платформе Stepik для обоих заданий (Рисунок 1).

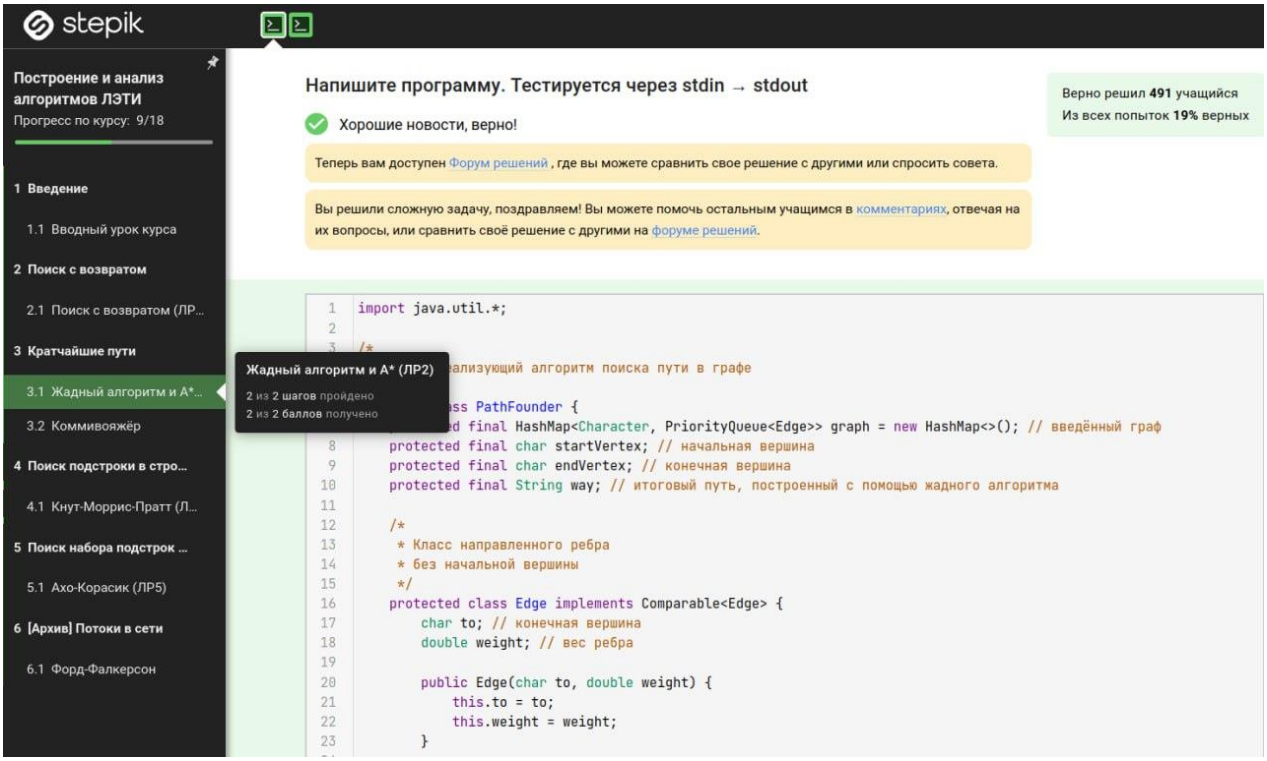

Рисунок 1 - скриншот выполненного задания на платформе stepik

Описание функций и структур данных.

Абстрактный класс, который является каким-либо алгоритмом, находящим путь в заданном графе, назван PathFounder.

Класс имеет поля. graph — ориентированный граф, заданный в формате, описанном при описании алгоритма. startVertex — начальная вершина графа. endVertex — конечная вершина графа. way — итоговый путь, построенный с помощью жадного алгоритма

Класс имеет абстрактный метод без параметров runAlgorithm(), который возвращает строку – найденный путь. В этом методе наследники должны реализовать построение пути по собственному алгоритму.

У класса единственный конструктор без параметров, который считывает сначала начальную и конечную вершины для алгоритма, а далее рёбра графа в формате:
<начальная вершина> <конечная вершина> <вес>. После запускает алгоритм для поиска пусти из начальной точки в конечную с помощью метода runAlgorithm() . Результат сохраняется в поле way.

Также внутри класса определён внутренний класс Edge, реализующий интерфейс Comparable<Edge>, — класс, хранящий одно исходящее ребро. Класс имеет поля to — конечная вершина, weight — вес ребра, конструктор, принимающий на вход оба поля, и переопределённый метод compareTo().

Класс, реализующий жадный алгоритм, назван GreedyAlgorithm, наследуется от класса PathFounder. Внутри класса переопределён метод runAlgorithm() — метод без параметров, реализующий жадный алгоритм для полей класса. Имеет возвращаемое значение типа String — полный найденный путь.

Класс, реализующий алгоритм A*, назван AStar, наследуется от класса PathFounder. Внутри класса переопределён метод runAlgorithm() — метод без параметров, реализующий алгоритм A* для полей класса. Имеет возвращаемое значение типа String — полный найденный путь.

Также внутри определены внутренние классы:

EdgeForAStar, наследуемый от Edge, с новыми полями fx — длина пути от начальной вершины до текущей, gx — эвристика, prev — предыдущий элемент и конструктором, заполняющим все поля. Класс необходим для хранения вершин в очереди во время работы алгоритма.

SettledEdgeInfo, с полями currentDX — длина пути от начальной вершины до текущей, currentPrev — предыдущий элемент, и конструктором, заполняющим все поля. Класс необходим для хранения информации для пройденной вершины.

AStarEdgeComparator, реализующий интерфейс Comparator<EdgeForAStar> . В классе есть единственный переопределённый

метод compare(), сравнивающий две вершины. Класс необходим для верного задания очереди в алгоритма A*.

Также создан публичный метод Main, содержащий единственный метод main() – точки входа в программу. В методе main() создаются экземпляры классов необходимых алгоритмов и печатает найденный путь для введённого графа.

Выводы.

В ходе работы был изучен на практике жадный алгоритм для графа и алгоритм A*. Был реализован поиск пусти в ориентированном графе с помощью двух алгоритмов.

Во время работы жадный алгоритм в каждый момент времени получал оптимальное локально решение, не опираясь на информацию обо всём графе полностью. Алгоритм A* получал оптимальное решение с учётом эвристической функции (разница ASCII кодов символов, являющими вершинами графа), которая помогала предсказывать оптимальность конкретного шага для всего пути.

Алгоритм успешно прошёл все тесты на платформе stepik.