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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритмы на графах

Студент гр. 7383 _____ Бергалиев М.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2019

Цель работы

Реализовать и исследовать алгоритм A* поиска кратчайшего пути в ориентированном графе.

Постановка задачи

Каждая вершина в графе именуется целым числом, каждое ребро имеет неотрицательный вес. В первой строке через пробел указываются начальная и конечная вершины. Далее в каждой строке указываются ребра графа и их вес.

Ход работы

Была написана программа на языке программирования Python 3. Код представлен в приложении А.

Функция A_star реализует алгоритм A*.

```
def A_star(vfrom, vto, e):
    queue = PriorityQueue()
    for i in e[vfrom]:
        queue.put((i[0] + abs(int(vto) - int(i[1])), i[1], vfrom+' '+i[1]))
    cur = queue.get()
    while cur[1] != vto and queue.qsize() != 0:
        if cur[1] not in e:
            cur = queue.get()
            continue
        for i in e[cur[1]]:
            if i[1] in cur[2]:
                continue
            queue.put((cur[0]+i[0]-abs(int(vto) - int(cur[1]))+abs(int(vto) - int(i[1])), i[1], cur[2]+' '+i[1]))
        cur = queue.get()
    if queue.qsize() == 0:
```

```
            return None
        return cur[2]
```

Функция принимает следующие аргументы:

- vfrom — вершина, являющаяся началом пути.
- vto — вершина, являющаяся концом пути.
- e — множество ребер графа.

На первом шаге создается очередь с приоритетами, в которую записываются пути длины 1 из начальной вершины. В качестве приоритета берется сумма стоимости пути и эвристической функции, задаваемой как абсолютная разность между номером конечной вершины и текущей. Далее на каждом шаге из очереди изымается путь. Если конечная вершина является концом искомого пути, то алгоритм заканчивает работу. Из конечной вершины этого пути рассматриваются все инцидентные вершины. Если рассматриваемой вершины нет в данном пути, то в очередь добавляется изъятый путь, к которому добавляется ребро из конечной вершины данного пути в рассматриваемую. Если очередь оказалось пустой, то такого пути нет и алгоритм заканчивает работу.

Тестирование

Тестирование проводилось в Ubuntu 16.04 LTS. По результатам тестирования были выявлены ошибки в коде. Тестовые случаи представлены в приложении Б.

Исследование алгоритма

Исследование проводилось по количеству путей, побывавших в очереди с приоритетами. Данные представлены в приложении В.

Была исследована зависимость сложности алгоритма в зависимости от длины оптимального пути. Для этого были взяты случайные графы с количеством вершин равным 30 и количеством ребер — 300. По полученным

данным был построен график, представленный на рис. 1. Сложность алгоритма зависит линейно от длины оптимального пути.

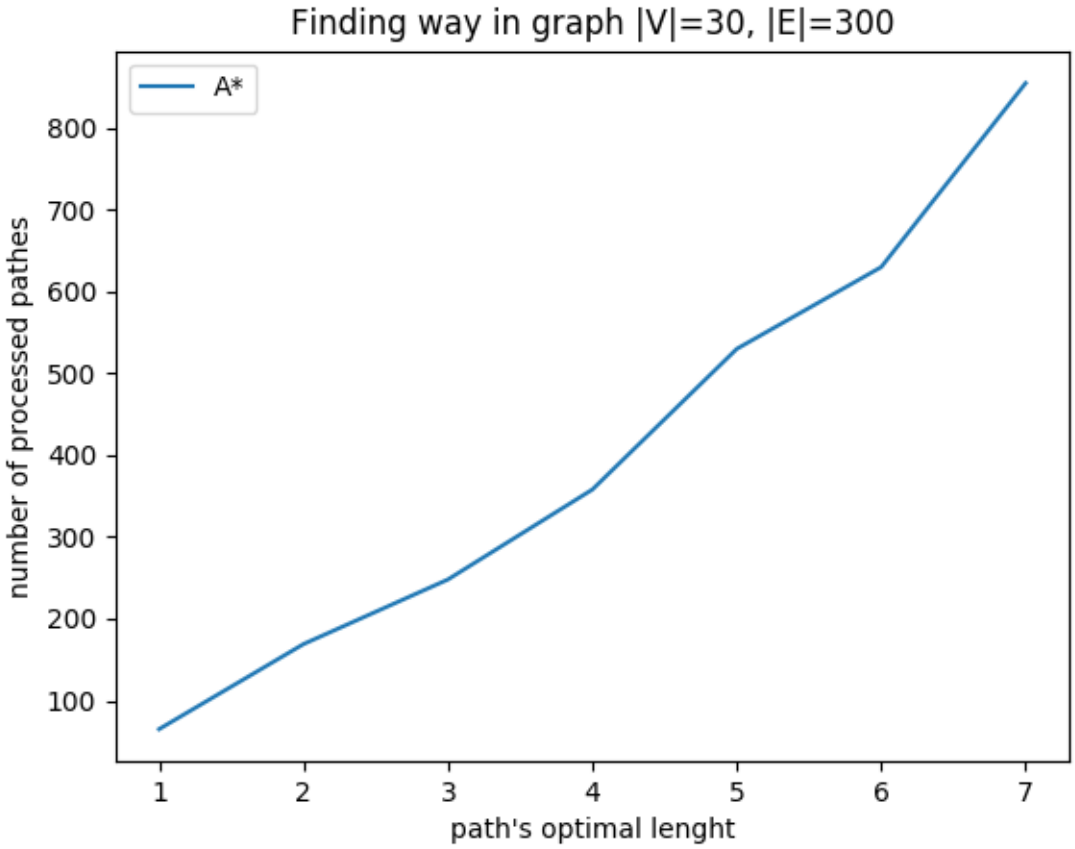

Рисунок 1 — Зависимость сложности алгоритма относительно длины оптимального пути

Была исследована зависимость сложности алгоритма в зависимости от количества ребер в графе. Для этого были взяты случайные графы с количеством вершин равным 30 и длиной оптимального пути равной 5. По полученным данным был построен график, представленный на рис. 2. Сложность алгоритма зависит логарифмически от числа ребер в графе.

Была исследована зависимость сложности алгоритма в зависимости от количества вершин в графе. Для этого были взяты случайные графы с количеством ребер равным 300 и длиной оптимального пути равной 5. По полученным данным был построен график, представленный на рис. 3. Сложность алгоритма зависит линейно от числа вершин в графе.

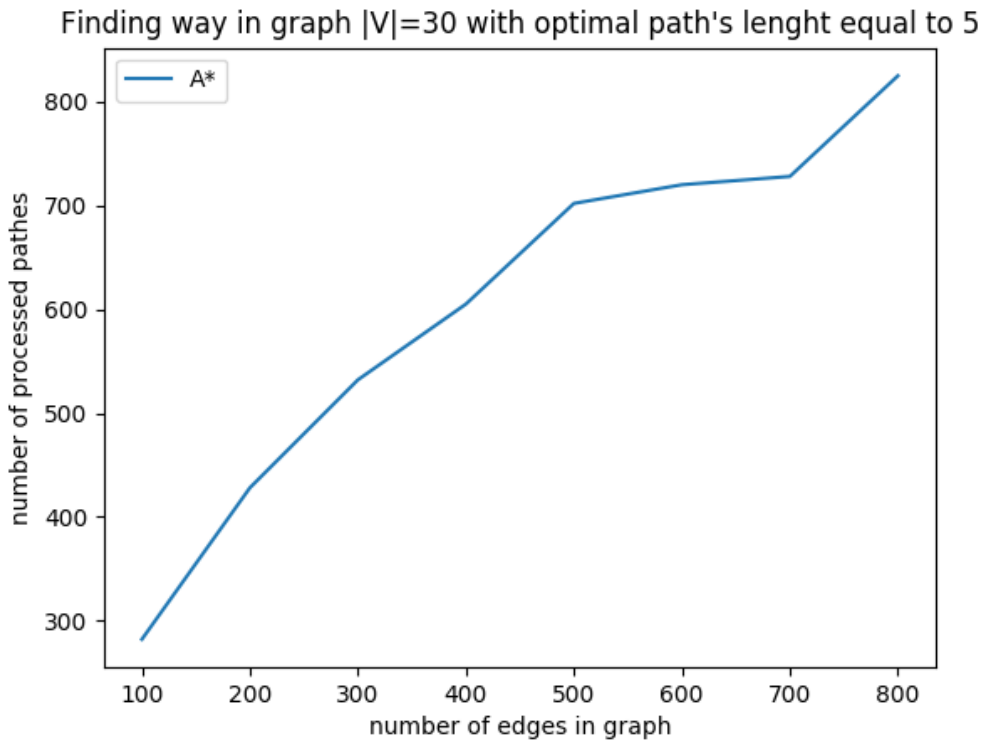

Рисунок 2 — Зависимость сложности алгоритма относительно количества ребер в графе

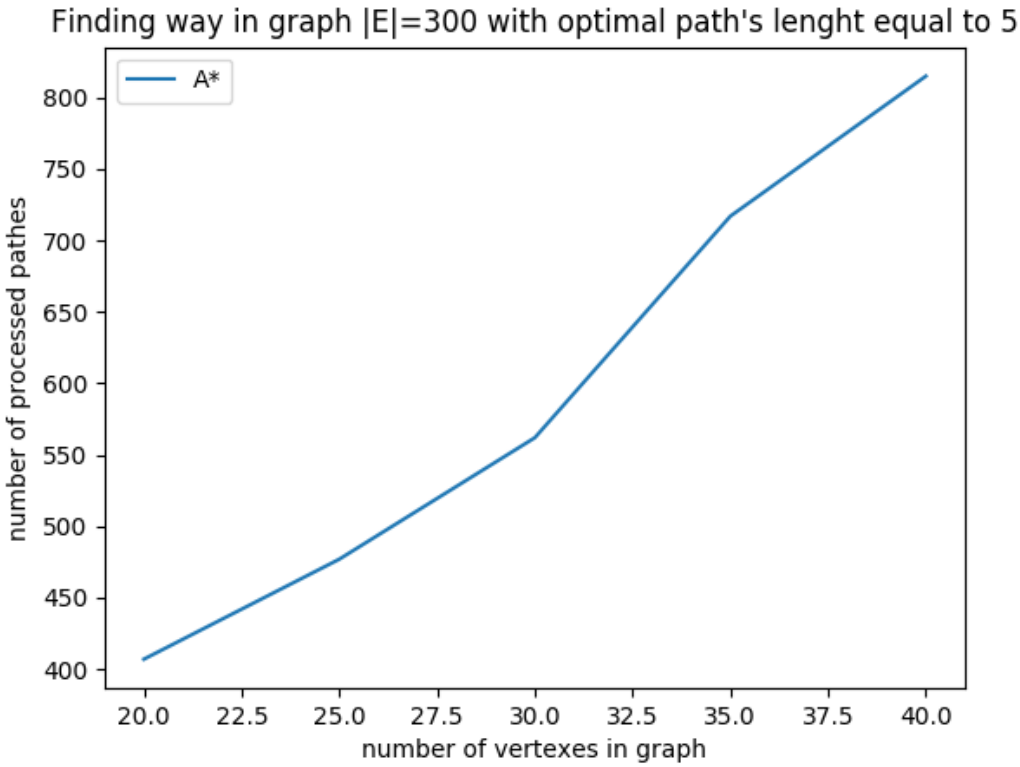

Рисунок 3 — Зависимость сложности алгоритма относительно количества вершин в графе

По полученным результатам можно сказать, что сложность алгоритма равна O(L|V|log(|E|)), где L — длина оптимального пути, |V| - количество вершин в графе, |E| - количество ребер в графе.

Вывод

Был изучен алгоритм A* поиска кратчайшего пути между вершинами в графе. Была реализована версия алгоритма на языке Python 3. Была исследована сложность алгоритма и по результатам сложность равна O(L|V| log(|E|)).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
    from sys import *
    from queue import *

    def A_star(vfrom, vto, e):
        queue = PriorityQueue()
        for i in e[vfrom]:
            queue.put((i[0]    +    abs(int(vto)    -
int(i[1])), i[1], vfrom+' '+i[1]))
        cur = queue.get()
        while cur[1] != vto and queue.qsize() != 0:
            if cur[1] not in e:
                cur = queue.get()
                continue
            for i in e[cur[1]]:
                if i[1] in cur[2]:
                    continue
                queue.put((cur[0]+i[0]-abs(int(vto)   -
int(cur[1]))+abs(int(vto)  -  int(i[1])),  i[1],  cur[2]+'
'+i[1]))
            cur = queue.get()
        if queue.qsize() == 0 and cur[1] != vto:
            return None
        return cur[2]


    def main():
        e = {}
        s = input().split()
        for i in stdin:
```

```
            i = i.split()
            if i[0] not in e:
                e[i[0]] = []
            e[i[0]].append((float(i[2]), i[1]))
        print(A_star(*s, e))

if __name__ == '__main__':
    main()
```

ПРИЛОЖЕНИЕ Б

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Тестовые случаи

Входные данные	Результат
1 -1 1 2 3.0 2 3 1.0 3 -7 1.0 1 -7 6.0 1 4 3.0 4 -1 4.0 -7 -1 1.0	1 4 -1
-13 -14 -13 21 2.0 -13 -15 9.0 -13 -20 7.0 -20 -15 1.0 -15 -14 1.5 21 -15 3.0	-13 -15 -14
0 200 0 -174 1.0 0 150 2.0 150 200 3.0 -174 200 1.0 0 174 4.0 174 200 1.0	0 174 200

ПРИЛОЖЕНИЕ В

ДАННЫЕ ИССЛЕДОВАНИЯ

Таблица 2 — Результаты исследования зависимости числа операций от длины оптимального пути

Длина оптимального пути	Количество операций
1	65
2	169
3	248
4	358
5	530
6	630
7	855
8	1002

Таблица 3 — Результаты исследования зависимости числа операций от числа ребер в графе

Количество ребер в графе	Количество операций
100	282
200	428
300	532
400	605
500	702
600	720
700	728
800	825

Таблица 4 — Результаты исследования зависимости числа операций от числа вершин в графе

Количество вершин в графе	Количество операций
20	407
25	477
30	562
35	717
40	815