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: 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):
    def evristic(x):
        for i in range(0, len(v)):
            if x == v[i]:
                x_i = i
            if vto == v[i]:
                vto_i = i
        return ((vto_i - x_i) / len(v)) * min_edge
    v = [i for i in e]
    for i in e:
        for j in e[i]:
            if j[1] not in v:
                v.append(j[1])
    if vto not in v:
        return None
    v.sort(key=int)
    min_edge = 0
    for i in e:
```

```
        for j in e[i]:
            if j[1] == vto and (j[0] < min_edge or
min_edge == 0):
                min_edge = j[0]
    queue = PriorityQueue()
    for i in e[vfrom]:
        queue.put((i[0] + evristic(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] - evristic(cur[1])
+ evristic(i[1]), i[1], cur[2] + ' ' + i[1]))
        cur = queue.get()
    if queue.qsize() == 0 and cur[1] != vto:
        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. Эвристическая функция задается как минимальный вес ребер, инцидентных конечной

вершине, разделенный на число вершин в графе и умноженный на расстояние в отсортированном списке вершин между данной вершиной и конечной. 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 количества ребер в графе. По полученным данным был построен график, представленный на рис. 1. Сложность алгоритма зависит квадратично от количества вершин.

Была исследована зависимость сложности алгоритма в зависимости от количества ребер в графе. По полученным данным был построен график, представленный на рис. 2. Сложность алгоритма зависит логарифмически от числа ребер в графе.

По полученным результатам можно сказать, что сложность алгоритма равна $O(||V||^2*\log(|E|))$, где $||V||$ - количество вершин в графе, $||E||$ - количество ребер в графе. Результирующий график представлен на рис. 3.

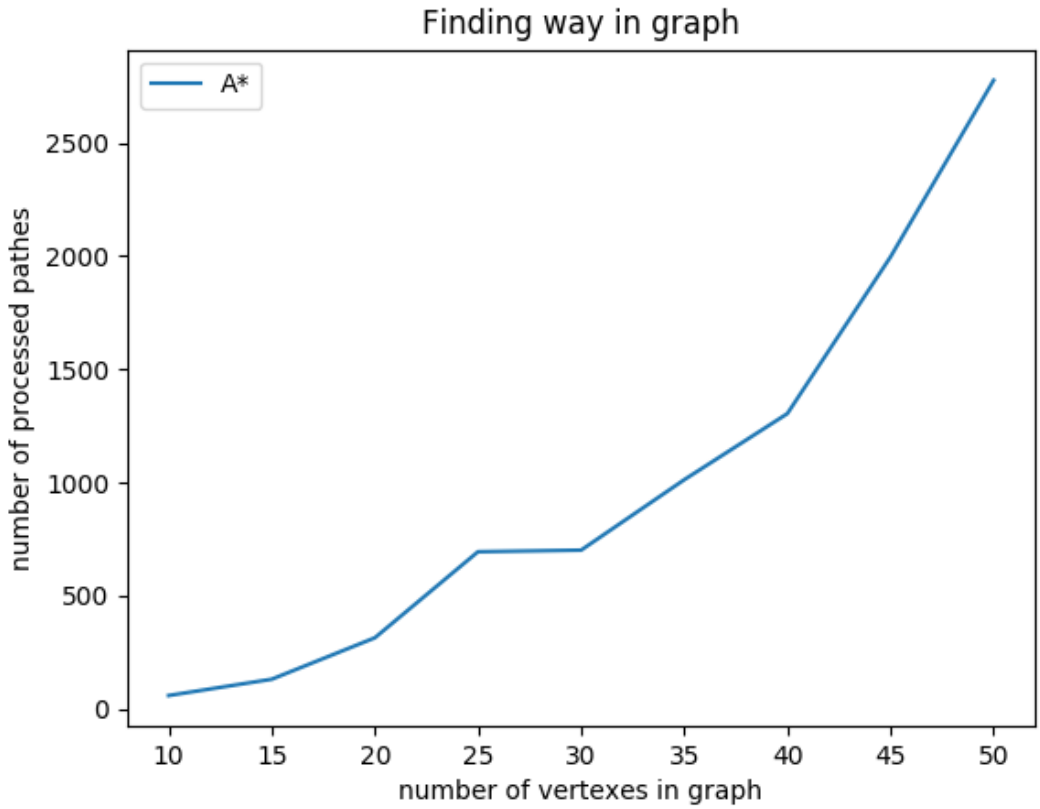

Рисунок 1 — Зависимость сложности алгоритма относительно числа вершин

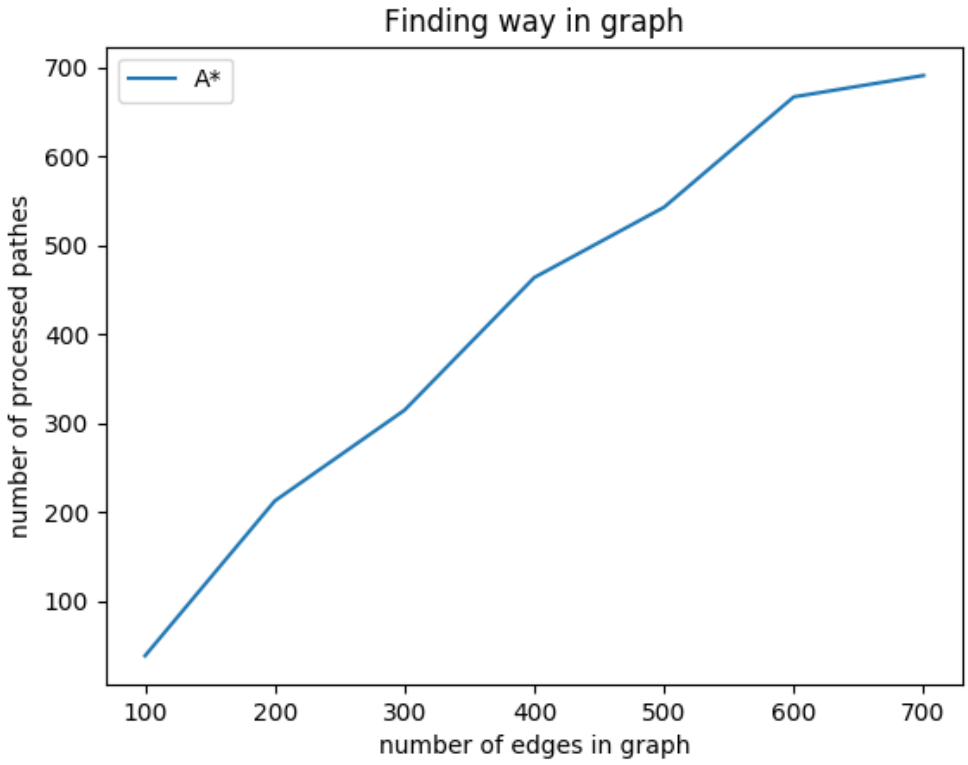

Рисунок 2 — Зависимость сложности алгоритма относительно количества ребер в графе

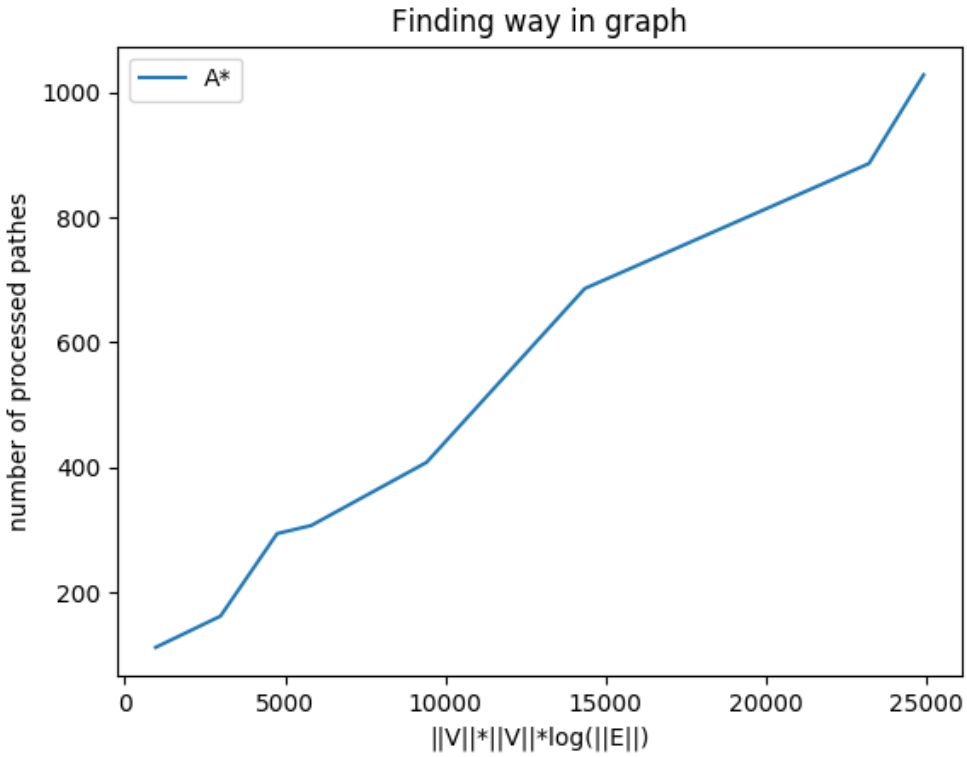

Рисунок 3 — Сложность алгоритма

Вывод

Был изучен алгоритм A* поиска кратчайшего пути между вершинами в графе. Была реализована версия алгоритма на языке Python 3, исследована сложность алгоритма, по результатам сложность равна $O(||V||^2 \log(||E||))$.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
from sys import *
from queue import *

def A_star(vfrom, vto, e):
    def evristic(x):
        for i in range(0, len(v)):
            if x == v[i]:
                x_i = i
            if vto == v[i]:
                vto_i = i
        return ((vto_i - x_i) / len(v)) * min_edge
    v = [i for i in e]
    for i in e:
        for j in e[i]:
            if j[1] not in v:
                v.append(j[1])
    if vto not in v:
        return None
    v.sort(key=int)
    min_edge = 0
    for i in e:
        for j in e[i]:
            if j[1] == vto and (j[0] < min_edge or
min_edge == 0):
                min_edge = j[0]
    queue = PriorityQueue()
    for i in e[vfrom]:
```

```
            queue.put((i[0]  +  evristic(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]     -
evristic(cur[1]) + evristic(i[1]), i[1], cur[2] + ' ' +
i[1]))
            cur = queue.get()
        if queue.qsize() == 0 and cur[1] != vto:
            return None
        return cur[2]
```

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 количества вершин в графе

Число вершин	Количество операций
10	59
15	131
20	314
25	694
30	701
35	1012
40	1304
45	1996
50	2776

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

Количество ребер в графе	Количество операций
100	39
200	213
300	315
400	464
500	543
600	667
700	691

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

$\|\|V\|\|^2\log(\|\|E\|\|)$	Количество операций
5815	307
2983	162
23210	886
9412	408
960	112

4748	294
14345	686
24914	1028