

**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: Алгоритм Дейкстры.**

Студент гр. 8304

Мухин А. М.

Преподаватель

Размочаева Т. В.

Санкт-Петербург

## Цель работы.

Вар. 5. Реализовать алгоритм Дейкстры поиска пути в графе (на основе кода A\*).

## Задание.

Разработать программу, которая решает задачу поиска кратчайшего пути в ориентированном графе до каждой вершины от текущей с помощью алгоритма Дейкстры. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес.

## Описание алгоритма.

Был реализован класс приоритетной очереди, которая хранила объекты в списке и при каждом новом добавлении сортировала его. Для корректной работы алгоритмы были созданы следующие методы: `push(item, value)`, `pop()`, `empty()`. UML-диаграмма этого класса представлена ниже.

class PriorityQueue	
+self.repository = []	
+pop(self)	
+push(self, item, value)	
+empty(self)	

Так как необходимо реализовать алгоритм на основе A\*, то необходимо просто в цикле вызвать A\* для каждой конечной вершины. Основная идея алгоритма – это поиск в ширину с приоритетом обрабатываемой вершины. Также во время работы алгоритмы запоминается путь до каждой вершины, правда в обратном порядке. После того, как основной костяк программы завершил работу, восстанавливается путь до текущей вершины и создаётся строка из вершин, по которым надо пройти чтобы достичь необходимую. Вершина и результат записываются в словарь, который в конечном итоге выводится программой.

Также стоит упомянуть о двух вспомогательных функциях. Это

s  
a  
m  
e

e  
n  
d  
p  
o  
s

который ~~пример входы, который~~ входит в значениях словаря, притом, что этот ключ

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

В первой строке через указывается начальная вершина. Далее в каждой строке указываются ребра графа и их вес

### **Вывод промежуточной информации.**

Во время основной части работы алгоритма происходит вывод промежуточной информации, а именно стоимость пути для каждой посещенной вершины и путь до неё в виде словаря с ключом вершины назначения и значением вершины отправления.

### **Выходные данные.**

В качестве выходных данных представлен словарь, с ключами в качестве вершин, до которых найден кратчайший путь и значениями в качестве кратчайшего пути. В случае, если к какой-то вершине графа нет пути, выведется сообщение об этом.

### **Сложность алгоритма.**

Сложность алгоритма по памяти:  $O(V+E)$

Сложность алгоритма по времени:  $O(n^2)$

## Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	a a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	{'a': 0, 'b': 3.0, 'd': 5.0, 'c': 4.0, 'e': 6.0}
2.	a a b 1.0 a f 3.0 b c 5.0 b g 3.0 f g 4.0 c d 6.0 d m 1.0 g e 4.0 g i 5.0 e h 1.0 e n 1.0 n m 2.0 i j 6.0 i k 1.0 j l 5.0 m j 3.0	{'a': 0, 'b': 1.0, 'f': 3.0, 'c': 6.0, 'g': 4.0, 'e': 8.0, 'i': 9.0, 'd': 12.0, 'h': 9.0, 'n': 9.0, 'm': 11.0, 'j': 14.0, 'k': 10.0, 'l': 19.0}
3.	j b a 1.0 f a 3.0 c b 5.0 g b 3.0 g f 4.0 d c 6.0 m d 1.0 e g 4.0	you can't achieve this vertex: h you can't achieve this vertex: k {'b': 'jmnegb', 'f': 'jmnegf', 'c': 'jmdc', 'g': 'jmneg', 'd': 'jmd', 'm': 'jm', 'e': 'jmne', 'i': 'ji', 'h': 'jmneg', 'n': 'jmn', 'k': 'jmneg', 'l': 'jl', 'a': 'jmnegba', 'j': 'j'}

i g 5.0	
h e 1.0	
n e 1.0	
m n 2.0	
j i 6.0	
k i 1.0	
l j 5.0	
j l 5.0	
j m 3.0	

### **Вывод.**

В ходе подготовки в данной лабораторной работе и её выполнение мы узнали, что такое жадные алгоритмы. Реализовали такие алгоритмы как A\*, алгоритм Дейкстры и жадный алгоритм.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
from sys import stdin

class PriorityQueue:
    def __init__(self):
        self.repository = []

    def push(self, item, value):
        self.repository.append((item, value))
        self.repository.sort(key=lambda pair: pair[1])

    def pop(self):
        return self.repository.pop(0)

    def empty(self):
        return len(self.repository) == 0

def del_same_value(reverse_dict, end_pos):
    for key in list(reverse_dict):
        if key not in reverse_dict.values() and key != end_pos:
            del reverse_dict[key]

def same_value_in(reverse_dict):
    for value in list(reverse_dict.values()):
        if list(reverse_dict.values()).count(value) > 1:
            return True

    return False

start_position = input()
graph = {}
result = {}

for line in stdin:
    source, destination, value = line.split()
    if source not in graph:
        graph[source] = {destination: float(value)}
    else:
        graph[source].update({destination: float(value)})

vertexes = [vertex for vertex in graph if vertex != start_position]
for key in graph:
    for vertex in graph[key]:
        if vertex not in vertexes:
            vertexes.append(vertex)

for end_position in vertexes:
    queue = PriorityQueue()
```

```

queue.push(start_position, 0)

path_to = {start_position: 0}
came_from = {start_position: None}

while not queue.empty():
    current_element = queue.pop()

    if current_element[0] == end_position:
        break

    if current_element[0] in graph:
        for neighbour in graph[current_element[0]]:
            path_length = path_to[current_element[0]] +
graph[current_element[0]][neighbour]
            if neighbour not in path_to or path_length <=
path_to[neighbour]:
                path_to[neighbour] = path_length
                queue.push(neighbour, path_length)
                came_from[neighbour] = current_element[0]

while same_value_in(came_from):
    del_same_value(came_from, end_position)

reverse_list = {}

for key in came_from:
    reverse_list[came_from[key]] = key
path = start_position

while path[-1] != end_position:
    try:
        path += reverse_list[path[-1]]
    except KeyError:
        print(f'you can\'t achieve this vertex: {end_position}')
        break

result[end_position] = path

print(result)

```