МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Построение и анализ алгоритмов»

Тема: Максимальный поток

Студент гр. 9382	____________________	Дерюгин Д.А.
Преподаватель	____________________	Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритм Форда-Фалкерсона, а также написать программу поиска максимального потока при помощи данного алгоритма.

Задание.

Вариант 6. Поиск не в глубину и не в ширину, а по правилу: каждый раз выполняется переход по дуге, соединяющей вершины, имена которых в алфавите ближе всего друг к другу. Если таких дуг несколько, то выбрать ту, имя конца которой в алфавите ближайшее к началу алфавита.

Найти максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро, используя алгоритм Форда-Фалкерсона.

Сеть (ориентированный взвешенный граф) представляется в виде триплета из имён вершин и целого неотрицательного числа - пропускной способности (веса).

Входные данные:

N - количество ориентированных рёбер графа

v_0 - исток

v_n - сток

v_i v_j ω_{ij} - ребро графа

v_i v_j ω_{ij} - ребро графа

...

Выходные данные:

P_{max} - величина максимального потока

v_i v_j ω_{ij} - ребро графа с фактической величиной протекающего потока

v_i v_j ω_{ij} - ребро графа с фактической величиной протекающего потока

…

В ответе выходные рёбра отсортируйте в лексикографическом порядке по первой вершине, потом по второй (в ответе должны присутствовать все указанные входные рёбра, даже если поток в них равен 0).

Sample Input:

```
7
a
f
a b 7
a c 6
b d 6
c f 9
d e 3
d f 4
e c 2
```

Sample Output:

```
12
a b 6
a c 6
b d 6
c f 8
d e 2
d f 4
e c 2
```

Описание алгоритма:

Ищем путь от истока с стоку следующим образом:

Все посещенные вершины записываются в список. Среди всех этих вершин выбираем ребро, соединяющее вершины, имена которых находятся ближе друг к другу в алфавите. Если же таких ребер несколько, тогда берется ребро, имя конца которого находятся ближе к началу алфавита.

После того, как алгоритм дошел до конечной вершины, ищется максимальный поток, который может пройти по данному пути(минимум из всех весов ребер на данном пути). После из каждого ребра в пути вычитается найденный минимальный поток. Алгоритм заканчивает свою работу, когда больше невозможно найти путь от истока к стоку.

Сложность:

По времени

В худшем случае, если при каждом найденном пути поток будет увеличиваться на единицу и проходить все ребра, тогда сложность по времени будет равна:

O(F*E), где F - максимальный поток, E - число ребер

По памяти

Граф хранит информацию о всех вершинах и ребрах, тогда сложность по памяти будет O(V+E), где V – количество вершин в графе, E - количество ребер.

Описание функций и структур данных:

class Vertex - класс вершины графа

Поля:

name - имя вершины

edges - список ребер вершины с их весами

visited - переменная, которая показывает, была ли вершин посещена

Class Graph - класс графа

Поля:

vertexes - словарь, ключи которого - вершины, а значения - класс этой вершины flow - максимальный поток, пройденный через граф

source - исток графа

sunk - сток графа

Методы класса:

def add_edge(self, name_from, name_to, weight) - метод, который добавляет новое ребро в граф, где name_from - откуда ребро исходит, name_to - куда входит ребро, weight - вес ребра.

def find_min_flow(graph) - ищет максимальный поток на одном пути из истока к стоку в графе graph

def remove_path(graph, min) - изменяет пропускные способности ребер в графе graph, где min - минимальная пропускная способность на данном пути

def start_algorithm(graph) - функция, которая реализует алгоритм.

graph - граф

def filter_vertexes(vertexes) - функция сортировки вершин в алфавитном порядке

vertexes - словарь вершин

def main() - стартовая функция, которая создает и заполняет граф

Тестирование алгоритма.

№	**Входные данные**	**Выходные данные**
1	7 a f a b 7 a c 6 b d 6 c f 9	Array of visited vertexes: a Looking vertex a with edges: a -> b = 7

	d e 3 d f 4 e c 2	a -> c = 6 Looking edge: a -> b = 7 Found new minimum by alphabetic order edge: a -> b = 7 Looking edge: a -> c = 6 New choosing vertex: b Array of visited vertexes: a b Looking vertex a with edges: a -> b = 7 a -> c = 6 Looking edge: a -> b = 7 Vertex b has been already visited. Choosing other edge Looking edge: a -> c = 6 Found new minimum by alphabetic order edge: a -> c = 6 Looking vertex b with edges: b -> a = 0 b -> d = 6 Looking edge: b -> a = 0

		Vertex a has been already visited. Choosing other edge Looking edge: b -> d = 6 New choosing vertex: c Array of visited vertexes: a b c Looking vertex a with edges: a -> b = 7 a -> c = 6 Looking edge: a -> b = 7 Vertex b has been already visited. Choosing other edge Looking edge: a -> c = 6 Vertex c has been already visited. Choosing other edge Looking vertex b with edges: b -> a = 0 b -> d = 6 Looking edge: b -> a = 0 Vertex a has been already visited. Choosing other edge

		Looking edge: b -> d = 6 Found new minimum by alphabetic order edge: b -> d = 6 Looking vertex c with edges: c -> a = 0 c -> f = 9 c -> e = 0 Looking edge: c -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: c -> f = 9 Looking edge: c -> e = 0 New choosing vertex: d Array of visited vertexes: a b c d Looking vertex a with edges: a -> b = 7 a -> c = 6 Looking edge: a -> b = 7 Vertex b has been already visited. Choosing other edge

		Looking edge: a -> c = 6 Vertex c has been already visited. Choosing other edge Looking vertex b with edges: b -> a = 0 b -> d = 6 Looking edge: b -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: b -> d = 6 Vertex d has been already visited. Choosing other edge Looking vertex c with edges: c -> a = 0 c -> f = 9 c -> e = 0 Looking edge: c -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: c -> f = 9 Found new minimum by alphabetic order edge: c -

		> f = 9 Looking edge: c -> e = 0 Looking vertex d with edges: d -> b = 0 d -> e = 3 d -> f = 4 Looking edge: d -> b = 0 Vertex b has been already visited. Choosing other edge Looking edge: d -> e = 3 Found new minimum by alphabetic order edge: d -> e = 3 Looking edge: d -> f = 4 New choosing vertex: e Array of visited vertexes: a b c d e Looking vertex a with edges: a -> b = 7 a -> c = 6 Looking edge: a -> b = 7 Vertex b has been already visited. Choosing other

		edge Looking edge: a -> c = 6 Vertex c has been already visited. Choosing other edge Looking vertex b with edges: b -> a = 0 b -> d = 6 Looking edge: b -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: b -> d = 6 Vertex d has been already visited. Choosing other edge Looking vertex c with edges: c -> a = 0 c -> f = 9 c -> e = 0 Looking edge: c -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: c -> f = 9 Found new minimum by

		alphabetic order edge: c -> f = 9 Looking edge: c -> e = 0 Vertex e has been already visited. Choosing other edge Looking vertex d with edges: d -> b = 0 d -> e = 3 d -> f = 4 Looking edge: d -> b = 0 Vertex b has been already visited. Choosing other edge Looking edge: d -> e = 3 Vertex e has been already visited. Choosing other edge Looking edge: d -> f = 4 Found new minimum by alphabetic order edge: d -> f = 4 Looking vertex e with edges: e -> d = 0 e -> c = 2 Looking edge: e -> d = 0

		Vertex d has been already visited. Choosing other edge Looking edge: e -> c = 2 Vertex c has been already visited. Choosing other edge New choosing vertex: f Counting max flow on current path Previous weight of edge f -> d was 4. New weight - 0 Previous weight of edge d -> b was 6. New weight - 2 Previous weight of edge b -> a was 7. New weight - 3 Current path: abdf Array of visited vertexes: a Looking vertex a with edges: a -> b = 3 a -> c = 6

		Looking edge: a -> b = 3 Found new minimum by alphabetic order edge: a -> b = 3 Looking edge: a -> c = 6 New choosing vertex: b Array of visited vertexes: a b Looking vertex a with edges: a -> b = 3 a -> c = 6 Looking edge: a -> b = 3 Vertex b has been already visited. Choosing other edge Looking edge: a -> c = 6 Found new minimum by alphabetic order edge: a -> c = 6 Looking vertex b with edges: b -> a = 0 b -> d = 2 Looking edge: b -> a = 0 Vertex a has been already visited. Choosing other

		edge Looking edge: b -> d = 2 New choosing vertex: c Array of visited vertexes: a b c Looking vertex a with edges: a -> b = 3 a -> c = 6 Looking edge: a -> b = 3 Vertex b has been already visited. Choosing other edge Looking edge: a -> c = 6 Vertex c has been already visited. Choosing other edge Looking vertex b with edges: b -> a = 0 b -> d = 2 Looking edge: b -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: b -> d = 2

		Found new minimum by alphabetic order edge: b -> d = 2 Looking vertex c with edges: c -> a = 0 c -> f = 9 c -> e = 0 Looking edge: c -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: c -> f = 9 Looking edge: c -> e = 0 New choosing vertex: d Array of visited vertexes: a b c d Looking vertex a with edges: a -> b = 3 a -> c = 6 Looking edge: a -> b = 3 Vertex b has been already visited. Choosing other edge Looking edge: a -> c = 6

		Vertex c has been already visited. Choosing other edge Looking vertex b with edges: b -> a = 0 b -> d = 2 Looking edge: b -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: b -> d = 2 Vertex d has been already visited. Choosing other edge Looking vertex c with edges: c -> a = 0 c -> f = 9 c -> e = 0 Looking edge: c -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: c -> f = 9 Found new minimum by alphabetic order edge: c -> f = 9

		Looking edge: c -> e = 0 Looking vertex d with edges: d -> b = 4 d -> e = 3 d -> f = 0 Looking edge: d -> b = 4 Vertex b has been already visited. Choosing other edge Looking edge: d -> e = 3 Found new minimum by alphabetic order edge: d -> e = 3 Looking edge: d -> f = 0 New choosing vertex: e Array of visited vertexes: a b c d e Looking vertex a with edges: a -> b = 3 a -> c = 6 Looking edge: a -> b = 3 Vertex b has been already visited. Choosing other edge

		Looking edge: a -> c = 6 Vertex c has been already visited. Choosing other edge Looking vertex b with edges: b -> a = 0 b -> d = 2 Looking edge: b -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: b -> d = 2 Vertex d has been already visited. Choosing other edge Looking vertex c with edges: c -> a = 0 c -> f = 9 c -> e = 0 Looking edge: c -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: c -> f = 9 Found new minimum by alphabetic order edge: c -

		> f = 9 Looking edge: c -> e = 0 Vertex e has been already visited. Choosing other edge Looking vertex d with edges: d -> b = 4 d -> e = 3 d -> f = 0 Looking edge: d -> b = 4 Vertex b has been already visited. Choosing other edge Looking edge: d -> e = 3 Vertex e has been already visited. Choosing other edge Looking edge: d -> f = 0 Looking vertex e with edges: e -> d = 0 e -> c = 2 Looking edge: e -> d = 0 Vertex d has been already visited. Choosing other edge Looking edge: e -> c = 2

		Vertex c has been already visited. Choosing other edge New choosing vertex: f Counting max flow on current path Previous weight of edge f -> c was 9. New weight - 3 Previous weight of edge c -> a was 6. New weight - 0 Current path: acf Array of visited vertexes: a Looking vertex a with edges: a -> b = 3 a -> c = 0 Looking edge: a -> b = 3 Found new minimum by alphabetic order edge: a -> b = 3 Looking edge: a -> c = 0 New choosing vertex: b Array of visited vertexes:

		a b Looking vertex a with edges: a -> b = 3 a -> c = 0 Looking edge: a -> b = 3 Vertex b has been already visited. Choosing other edge Looking edge: a -> c = 0 Looking vertex b with edges: b -> a = 0 b -> d = 2 Looking edge: b -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: b -> d = 2 Found new minimum by alphabetic order edge: b -> d = 2 New choosing vertex: d Array of visited vertexes: a b d

		Looking vertex a with edges: a -> b = 3 a -> c = 0 Looking edge: a -> b = 3 Vertex b has been already visited. Choosing other edge Looking edge: a -> c = 0 Looking vertex b with edges: b -> a = 0 b -> d = 2 Looking edge: b -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: b -> d = 2 Vertex d has been already visited. Choosing other edge Looking vertex d with edges: d -> b = 4 d -> e = 3 d -> f = 0 Looking edge: d -> b = 4 Vertex b has been already

		visited. Choosing other edge Looking edge: d -> e = 3 Found new minimum by alphabetic order edge: d -> e = 3 Looking edge: d -> f = 0 New choosing vertex: e Array of visited vertexes: a b d e Looking vertex a with edges: a -> b = 3 a -> c = 0 Looking edge: a -> b = 3 Vertex b has been already visited. Choosing other edge Looking edge: a -> c = 0 Looking vertex b with edges: b -> a = 0 b -> d = 2 Looking edge: b -> a = 0 Vertex a has been already visited. Choosing other

		edge Looking edge: b -> d = 2 Vertex d has been already visited. Choosing other edge Looking vertex d with edges: d -> b = 4 d -> e = 3 d -> f = 0 Looking edge: d -> b = 4 Vertex b has been already visited. Choosing other edge Looking edge: d -> e = 3 Vertex e has been already visited. Choosing other edge Looking edge: d -> f = 0 Looking vertex e with edges: e -> d = 0 e -> c = 2 Looking edge: e -> d = 0 Vertex d has been already visited. Choosing other edge Looking edge: e -> c = 2

		Found new minimum by alphabetic order edge: e -> c = 2 New choosing vertex: c Array of visited vertexes: a b d e c Looking vertex a with edges: a -> b = 3 a -> c = 0 Looking edge: a -> b = 3 Vertex b has been already visited. Choosing other edge Looking edge: a -> c = 0 Vertex c has been already visited. Choosing other edge Looking vertex b with edges: b -> a = 0 b -> d = 2 Looking edge: b -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: b -> d = 2

		Vertex d has been already visited. Choosing other edge Looking vertex d with edges: d -> b = 4 d -> e = 3 d -> f = 0 Looking edge: d -> b = 4 Vertex b has been already visited. Choosing other edge Looking edge: d -> e = 3 Vertex e has been already visited. Choosing other edge Looking edge: d -> f = 0 Looking vertex e with edges: e -> d = 0 e -> c = 2 Looking edge: e -> d = 0 Vertex d has been already visited. Choosing other edge Looking edge: e -> c = 2 Vertex c has been already visited. Choosing other

		edge Looking vertex c with edges: c -> a = 0 c -> f = 3 c -> e = 0 Looking edge: c -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: c -> f = 3 Found new minimum by alphabetic order edge: c -> f = 3 Looking edge: c -> e = 0 Vertex e has been already visited. Choosing other edge New choosing vertex: f Counting max flow on current path Previous weight of edge f -> c was 3. New weight - 1 Previous weight of edge c -> e was 2. New weight - 0 Previous weight of edge e

		-> d was 3. New weight - 1 Previous weight of edge d -> b was 2. New weight - 0 Previous weight of edge b -> a was 3. New weight - 1 Current path: abdecf Array of visited vertexes: a Looking vertex a with edges: a -> b = 1 a -> c = 0 Looking edge: a -> b = 1 Found new minimum by alphabetic order edge: a -> b = 1 Looking edge: a -> c = 0 New choosing vertex: b Array of visited vertexes: a b Looking vertex a with

		edges: a -> b = 1 a -> c = 0 Looking edge: a -> b = 1 Vertex b has been already visited. Choosing other edge Looking edge: a -> c = 0 Looking vertex b with edges: b -> a = 0 b -> d = 0 Looking edge: b -> a = 0 Vertex a has been already visited. Choosing other edge Looking edge: b -> d = 0 12 a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2

2	11 a h a b 4 b e 2 a c 2 c e 3 a d 3 d e 4 e g 3 e f 2 f h 3 g h 1 d f 1	4 a b 0 a c 2 a d 2 b e 0 e g 1 e f 2 c e 2 d e 1 d f 1 g h 1 f h 3
3	8 a f a b 6 a c 7 b d 4 c f 6 d e 3 d f 5 d m 2 e c 2	10 a b 4 a c 6 b d 4 c f 6 d e 0 d f 4 d m 0 e c 0
4	5 a d a b 1000	1001 a b 1000 a c 1 b d 1000

	a c 300 b d 3000 b c 100 c d 1	b c 0 c d 1

Выводы.

В результате выполнения работы был изучен алгоритм Форда-Фалкерсона а также написана программа, которая ищет максимальный поток в графе при помощи данного алгоритма.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.py.

```
import math


class Vertex:
    visited = False  # visited or not
    vertex_from = [None, 0]  # from vertex vertex_from[0] with weight
vertex_from[1]

    def __init__(self, name):
        self.name = name
        self.edges = {}  # list of neighbours


class Graph:
    vertexes = {}  # dict of vertexes
    source = None
    sunk = None
    flow = 0

    def add_edge(self, name_from, name_to, weight):
        if name_from in self.vertexes.keys():

            self.vertexes[name_from].edges[name_to] = [weight, 0, 1]
        else:

            self.vertexes[name_from] = Vertex(name_from)
            self.vertexes[name_from].edges[name_to] = [weight, 0, 1]
        if name_to in self.vertexes.keys():

            self.vertexes[name_to].edges[name_from] = [0, 0, -1]
        else:

            self.vertexes[name_to] = Vertex(name_to)
            self.vertexes[name_to].edges[name_from] = [0, 0, -1]


# sorted vertex by alphabetic order
def filter_vertexes(vertexes):
    l = list(vertexes.keys())
    l.sort()
    new_v = []
    for item in l:
        new_v.append(vertexes[item])
    return new_v


# found minimum flow on path
def found_min(visited_vertexes, graph):
    minimum = math.inf
    minimum_vertex = 'z'
    previous_vertex = None
    weight = 0
    for visit_vertex in visited_vertexes:
        print(f'Looking vertex {visit_vertex} with edges:')
        for name, edge in graph.vertexes[visit_vertex].edges.items():
```

```
            print(f'{visit_vertex} -> {name} = {edge[0]}')
        for name, edge in graph.vertexes[visit_vertex].edges.items():
            print(f'Looking edge: {visit_vertex} -> {name} = {edge[0]}')
            if graph.vertexes[name].visited:
                print(f'Vertex {name} has been already visited. Choosing
other edge')
                continue
            if (abs(ord(visit_vertex) - ord(name)) < minimum or abs(
                    ord(visit_vertex) - ord(name)) == minimum and
minimum_vertex > name) and edge[0] > 0:
                print(f'Found new minimum by alphabetic order edge:
{visit_vertex} -> {name} = {edge[0]}')
                previous_vertex = visit_vertex
                minimum = abs(ord(visit_vertex) - ord(name))
                minimum_vertex = name
                weight = edge[0]
    if minimum == math.inf:
        return False
    return [previous_vertex, minimum_vertex, weight]


def start_algorithm(graph):
    visited_vertexes = [graph.source]
    graph.vertexes[graph.source].visited = True
    current_vertex = graph.source

    while current_vertex != graph.sunk:
        print('Array of visited vertexes:')
        for vertex in visited_vertexes:
            print(vertex, end=' ')
        print('\n')
        edge = found_min(visited_vertexes, graph)


        if not edge:
            return False
        print('New choosing vertex: ', edge[1])
        visited_vertexes.append(edge[1])
        graph.vertexes[edge[1]].visited = True
        graph.vertexes[edge[1]].vertex_from = [edge[0], edge[2]]
        current_vertex = edge[1]
    return True


def find_min_flow(graph):
    min = graph.vertexes[graph.sunk].vertex_from[1]
    current_vertex = graph.sunk
    while graph.vertexes[current_vertex].vertex_from[0]:
        if min > graph.vertexes[current_vertex].vertex_from[1]:
            min = graph.vertexes[current_vertex].vertex_from[1]
        current_vertex = graph.vertexes[current_vertex].vertex_from[0]
    return min


def remove_path(graph, minimum):
    current_vertex = graph.vertexes[graph.sunk].vertex_from[0]
    previous_vertex = graph.sunk
    path = graph.sunk
    print('Counting max flow on current path')
    while graph.vertexes[current_vertex].vertex_from[0]:
        path += current_vertex
        print(
            f'Previous weight of edge {previous_vertex} -> {current_vertex}
was \
```

```
{graph.vertexes[current_vertex].edges[previous_vertex][0]}. New weight - \
{graph.vertexes[current_vertex].edges[previous_vertex][0] - minimum}')
        graph.vertexes[current_vertex].edges[previous_vertex][0] -= minimum
        graph.vertexes[current_vertex].edges[previous_vertex][1] += minimum

        graph.vertexes[previous_vertex].edges[current_vertex][0] += minimum
        graph.vertexes[previous_vertex].edges[current_vertex][1] -= minimum

        previous_vertex = current_vertex
        current_vertex = graph.vertexes[current_vertex].vertex_from[0]

    path += current_vertex
    print(f'Previous weight of edge {previous_vertex} -> {current_vertex} was
\
{graph.vertexes[current_vertex].edges[previous_vertex][0]}. New weight - \
{graph.vertexes[current_vertex].edges[previous_vertex][0] - minimum}')
    graph.vertexes[current_vertex].edges[previous_vertex][0] -= minimum
    graph.vertexes[current_vertex].edges[previous_vertex][1] += minimum
    print('Current path:')
    print(path[::-1])


def print_result(graph):
    for vertex in graph.vertexes:
        for name, edge in graph.vertexes[vertex].edges.items():
            print(f'{vertex} {name} {edge[1]}') if edge[1] >= 0 and edge[2]
!= -1 else None


def main():
    count_of_edges = int(input())  # count of edges
    graph = Graph()  # create graph
    graph.source = input()  # add source
    graph.sunk = input()  # add sunk

    # add edges to graph
    for _ in range(0, count_of_edges):
        name_from, name_to, weight = input().split(" ")
        graph.add_edge(name_from, name_to, int(weight))

    # filter vertexes
    filter_vertexes(graph.vertexes)

    while start_algorithm(graph):
        minimum_flow = find_min_flow(graph)
        graph.flow += minimum_flow
        remove_path(graph, minimum_flow)

        for vertex in graph.vertexes:
            graph.vertexes[vertex].vertex_from = [None, 0]
            graph.vertexes[vertex].visited = False
    print(graph.flow)
    print_result(graph)


main()
```