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:

Берем ребро до смежной вершины, которая в алфавите стоит ближе всех смежных вершин к данной. Текущая вершина записывается в пройденный путь. Переходим к смежной вершине. Проделываем данный шаг до тех пор, пока текущая вершина не окажется стоком. В найденном пути ищем ребро, ко которому проходит минимальный поток. После нахождения минимального потока на пути уменьшаем пропускные способности каждого ребра на этот минимальный поток.

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

Так как граф хранится в виде словаря, где ключи - это вершины, а значения - список смежных вершин, тогда сложность по памяти будет O(4*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 и направлениями

Class Graph - класс графа

Поля:

Vertexes - словарь, ключи которого - вершины, а значения - класс этой вершины

max_flow - максимальный поток, пройденный через граф

flows - список ребер с весами, пройденные, которые показывают путь от истока к стоку.

visited_vertexes - список посещенных вершин

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.

Min - минимальная пропускная способность на данном пути

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

graph - граф

current_vertex - текущая вершина

def filter_vertexes(vertexes) - функция сортировки смежных вершин согласно индивидуализации

Vertexes - словарь вершин

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

Тестирование сортировки

№	**Входные данные**	**Выходные данные**
1	3 a e a e 5 a b 3	Edges before sorting: a -> e = 5 a -> b = 3 a -> f = 1 Edges after sorting:

	a f 1	a -> b = 3 a -> e = 5 a -> f = 1
2	4 b e b c 2 b e 5 b a 1 b j 6	Edges before sorting: b -> c = 2 b -> e = 5 b -> a = 1 b -> j = 6 Edges after sorting: b -> a = 2 b -> c = 1 b -> e = 5 b -> j = 6
3	5 o l o l 7 o a 6 o p 7 o n 8 o j 6	Edges before sorting: o -> l = 7 o -> a = 6 o -> p = 7 o -> n = 8 o -> j = 6 Edges after sorting: o -> n = 7 o -> p = 8 o -> l = 7 o -> j = 6 o -> a = 6

Тестирование алгоритма.

№	Входные данные	Выходные данные
1	7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e c 2	--- Sorting vertexes--- Sorting vertex a Edges before sorting: a -> b = 7 a -> c = 6 Edges after sorting: a -> b = 7 a -> c = 6 Sorting vertex b Edges before sorting: b -> d = 6 Edges after sorting: b -> d = 6 Sorting vertex c Edges before sorting: c -> f = 9 Edges after sorting: c -> f = 9 Sorting vertex d Edges before sorting: d -> e = 3 d -> f = 4

		Edges after sorting: d -> e = 3 d -> f = 4 Sorting vertex f Edges before sorting: Edges after sorting: Sorting vertex e Edges before sorting: e -> c = 2 Edges after sorting: e -> c = 2 Current vertex: "a". Edges: a -> b = 7 a -> c = 6 View path: a->b = 7 Vertex "b" was added to path Current vertex: "b". Edges: b -> d = 6 View path: b->a = 0 Current path not good View path: b->d = 6

		Vertex "d" was added to path Current vertex: "d". Edges: d -> e = 3 d -> f = 4 View path: d->e = 3 Vertex "e" was added to path Current vertex: "e". Edges: e -> c = 2 View path: e->d = 0 Current path not good View path: e->c = 2 Vertex "c" was added to path Current vertex: "c". Edges: c -> f = 9 View path: c->a = 0 Current path not good

		View path: c->e = 0 Current path not good View path: c->f = 9 Vertex "f" was added to path View path: f->d = 0 Sunk vertex found --- Path was found--- Path: a->b[7; 0] b->d[6; 0] d->e[3; 0] e->c[2; 0] c->f[9; 0] Minimum flow in this path: 2 --- Updating flows--- Old flow: a->b = [7, 0] New flow: a->b = [5, 2] Old flow: b->d = [6, 0] New flow: b->d = [4, 2] Old flow: d->e = [3, 0] New flow: d->e = [1, 2] Old flow: e->c = [2, 0]

		New flow: e->c = [0, 2] Old flow: c->f = [9, 0] New flow: c->f = [7, 2] Current vertex: "a". Edges: a -> b = 5 a -> c = 6 View path: a->b = 5 Vertex "b" was added to path Current vertex: "b". Edges: b -> a = -2 b -> d = 4 View path: b->a = -2 Current path not good View path: b->d = 4 Vertex "d" was added to path Current vertex: "d". Edges:

		d -> e = 1 d -> b = -2 d -> f = 4 View path: d->e = 1 Vertex "e" was added to path Current vertex: "e". Edges: e -> d = -2 View path: e->d = -2 Current path not good View path: e->c = 0 Current path not good View path: d->b = -2 Current path not good View path: d->f = 4 Vertex "f" was added to path View path: f->d = 0 Sunk vertex found --- Path was found--- Path: a->b[5; 2] b->d[4; 2] d-

		>f[4; 0] Minimum flow in this path: 4 --- Updating flows--- Old flow: a->b = [5, 2] New flow: a->b = [1, 6] Old flow: b->d = [4, 2] New flow: b->d = [0, 6] Old flow: d->f = [4, 0] New flow: d->f = [0, 4] Current vertex: "a". Edges: a -> b = 1 a -> c = 6 View path: a->b = 1 Vertex "b" was added to path Current vertex: "b". Edges: b -> a = -6 View path: b->a = -6

		Current path not good View path: b->d = 0 Current path not good View path: a->c = 6 Vertex "c" was added to path Current vertex: "c". Edges: c -> e = -2 c -> f = 7 View path: c->a = 0 Current path not good View path: c->e = -2 Vertex "e" was added to path Current vertex: "e". Edges: e -> d = -2 View path: e->d = -2 Vertex "d" was added to path

		Current vertex: "d". Edges: d -> e = 1 d -> b = -6 View path: d->e = 1 Current path not good View path: d->b = -6 Current path not good View path: d->f = 0 Current path not good View path: e->c = 0 Current path not good View path: c->f = 7 Vertex "f" was added to path View path: f->d = -4 Sunk vertex found --- Path was found--- Path: a->c[6; 0] c->f[7; 2] Minimum flow in this path: 6

		--- Updating flows--- Old flow: a->c = [6, 0] New flow: a->c = [0, 6] Old flow: c->f = [7, 2] New flow: c->f = [1, 8] Current vertex: "a". Edges: a -> b = 1 View path: a->b = 1 Vertex "b" was added to path Current vertex: "b". Edges: b -> a = -6 View path: b->a = -6 Current path not good View path: b->d = 0 Current path not good View path: a->c = 0

		Current path not good 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 2 a c 1 a d 1 b e 2 c e 1 d e 0 d f 1 e f 2 e g 1 f h 3 g h 1
3	8 a f	10 a b 4 a c 6

	a b 6 a c 7 b d 4 c f 6 d e 3 d f 5 d m 2 e c 2	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 a c 300 b d 3000 b c 100 c d 1	1001 a b 1000 a c 1 b c 0 b d 1000 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.

```
class Vertex:

    def __init__(self, name):
        self.name = name
        self.edges = []


class Graph:
    vertexes = {}
    max_flow = 0
    flows = []
    visited_vertexes = []
    source = None
    sunk = None

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


def find_min_flow(graph):
    print('--- Path was found---\n Path:')
    weights = []
    for edge in graph.flows:
        print(f'{edge[0]}->{edge[1]}[{edge[2]}; {edge[3]}]  ', end=' ')
        if edge[4] == 1:
            weights.append(edge[2])
        else:
            weights.append(edge[3])
    print(f'\n Minimum flow in this path: {min(weights)}')
    return min(weights)


def remove_path(graph, min):
    graph.visited_vertexes = []  # clear visited vertexes
    print('\n--- Updating flows---')
    for edge in graph.flows:

        if edge[4] == 1:
            for edge_inner in graph.vertexes[edge[0]].edges:
                if edge_inner[0] == edge[1] and edge_inner[3] == 1:
                    print(f'Old flow: {edge[0]}->{edge[1]} = [{edge[2]},
{edge[3]}]')
                    print(f'New flow: {edge[0]}->{edge[1]} = [{edge[2] -
min}, {edge[3] + min}]\n')
```

```
                    edge_inner[1] -= min
                    edge_inner[2] += min
            for edge_inner_negative in graph.vertexes[edge[1]].edges:
                if edge_inner_negative[0] == edge[0] and
edge_inner_negative[3] == -1:
                    edge_inner_negative[1] -= min
                    edge_inner_negative[2] += min

        if edge[4] == -1:
            for edge_inner in graph.vertexes[edge[0]].edges:
                if edge_inner[0] == edge[1] and edge_inner[3] == -1:
                    print(f'Old flow: {edge[0]}->{edge[1]} = [{edge[2]},
{edge[3]}]')
                    print(f'New flow: {edge[0]}->{edge[1]} = [{edge[2] +
min}, {edge[3] - min}]\n')
                    edge_inner[1] += min
                    edge_inner[2] -= min
            for edge_inner_positive in graph.vertexes[edge[1]].edges:
                if edge_inner_positive[0] == edge[0] and
edge_inner_positive[3] == 1:
                    edge_inner_positive[1] += min
                    edge_inner_positive[2] -= min
    graph.flows = []


def start_algorithm(graph, current_vertex):
    if current_vertex != graph.sunk:
        print(f'\nCurrent vertex: "{current_vertex}". Edges:')
        for edge in graph.vertexes[current_vertex].edges:
            if edge[3] == 1 and edge[1] != 0:
                print(f'{current_vertex} -> {edge[0]} = {edge[1]}')
            elif edge[3] == -1 and edge[2] != 0:
                print(f'{current_vertex} -> {edge[0]} = {edge[2] * -1}')

    for edge in graph.vertexes[current_vertex].edges:
        if edge[3] == 1 :
            print(f'View path: {current_vertex}->{edge[0]} = {edge[1]}')
        elif edge[3] == -1:
            print(f'View path: {current_vertex}->{edge[0]} = {edge[2] * -1}')
        if current_vertex == graph.sunk:
            print('Sunk vertex found')
            minimum = find_min_flow(graph)  # find minimum flow in current
step
            graph.max_flow += minimum  # add up previous flow and new flow
            remove_path(graph, minimum)  # clear path and recalculation flows
            return True
        if edge[1] > 0 and edge[3] == 1 and edge[0] not in
graph.visited_vertexes \
                or edge[2] > 0 and edge[3] == -1 and edge[0] not in
graph.visited_vertexes:
            print(f'Vertex "{edge[0]}" was added to path')
            graph.visited_vertexes.append(edge[0])  # add new vertex as
visited
            graph.flows.append([current_vertex, edge[0], edge[1], edge[2],
edge[3]])
            if start_algorithm(graph, edge[0]):
                return True
            graph.flows.pop()  # remove last vertex from path
        else:
            print('Current path not good\n')
    return False


def filter_vertexes(vertexes):
```

```
    print('--- Sorting vertexes---')
    for vertex in vertexes.values():
        if not vertex.edges:
            continue
        print(f'Sorting vertex {vertex.name}')
        print('Edges before sorting:')
        for edge in vertex.edges:
            if edge[3] == -1:
                continue
            print(f'{vertex.name} -> {edge[0]} = {edge[1]}', sep=' ')

        # sorting edges by alphabetic order
        vertex.edges = sorted(vertex.edges, key=lambda item: abs(ord(item[0])
- ord(vertex.name)))

        print('Edges after sorting:')
        for edge in vertex.edges:
            if edge[3] == -1:
                continue
            print(f'{vertex.name} -> {edge[0]} = {edge[1]}', sep=' ')


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)

    graph.visited_vertexes.append(graph.source)  # add source as visited
vertex

    while start_algorithm(graph, graph.source):
        graph.visited_vertexes.append(graph.source)

    print(graph.max_flow)
    edges = []
    for vertex in graph.vertexes.values():
        for edge in vertex.edges:
            if edge[3] == 1:
                edges.append([f"{vertex.name}{edge[0]}", edge[2]])

    sorted_edges = sorted(edges, key=lambda item: item[0])
    for edge in sorted_edges:
        print(edge[0][0], edge[0][1], edge[1])


main()
```