Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт № 8 информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №9 по курсу**

**«Дискретный анализ»**

**Алгоритм Форда-Фалкерсона**

Студент: Пермяков Никита Александрович

Группа: М80 – 208Б-19

Вариант: 7

Преподаватель: [*Кухтичев Антон Алексеевич*](https://mai.ru/education/schedule/ppc.php?guid=a3f854e1-f771-11e7-ae95-485b3919ee6d)

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2020

**Содержание**

1. Постановка задачи
2. Метод и алгоритм решения
3. Описание программы
4. Дневник отладки
5. Тестирование производительности
6. Вывод

**Постановка задачи**

Задан взвешенный ориентированный граф, состоящий из n вершин и m ребер. Вершины пронумерованы целыми числами от 1 до n.

Необходимо найти величину максимального потока в графе при помощи алгоритма Форда-Фалкерсона.

Для достижения приемлемой производительности в алгоритме рекомендуется использовать поиск в ширину, а не в глубину.

Истоком является вершина с номером 1, стоком – вершина с номером n. Вес ребра равен его пропускной способности. Граф не содержит петель и кратных ребер.

**Метод и алгоритм решения**

Задача состоит в минимизации необходимого пропускного значения ребер графа. Задача о максимальном потоке сводится к задаче линейного про- граммирования.

Изначально величине потока присваивается значение 0: f (u, v) = 0 для всех u, v из V. Затем величина потока увеличивается на каждой итерации, происходит поиск пути от источника s к стоку t, вдоль которого можно послать ненулевой поток.

Шаг алгоритма состоит в выборе пути из истока в сток в остаточной сети и увеличении потока вдоль него, при этом ограничивает ребро с наименьшей пропускной способностью в остаточной сети. Процесс повторяется, пока можно найти увеличивающий путь.

1. Обнуляем все потоки, остаточная сеть совпадает с исходной сетью.
2. В остаточной сети находим кратчайший путь из источника в сток. Если такого пути нет, останавливаемся.
3. Пускаем через пройденный путь максимально возможный поток, ищем в нем ребро с минимальной пропускной способностью, для каждого ребра на найденном пути увеличиваем потока на это число, а в противоположном ему уменьшаем.
4. Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных им рёбер, вычисляем новую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной сети. Если обнулилась, стираем ребро.

**Описание программы**

int from, to - номера вершин, соединенных ребром

graph[from][to] - вес данного ребра

uint64\_t FordFulkerson(std::vector<std::vector<int> > &graph, uint64\_t&& start, uint64\_t& end) - функция с циклом для поиска в ширину, записывает и возвращает максимальный поток

bool BFS(std::vector<std::vector<int> > &graph, uint64\_t& start, uint64\_t& end, std::vector<int> &parent) - функция поиска в ширину

bool DFS(std::vector<std::vector<int> > &graph, std::vector<int> &parent, std::vector<bool>& visited, uint64\_t& x, uint64\_t& t) - функция поиска в глубину для сравнения

**Дневник отладки**

1)Кодирование программы

2) Удаление лишних библиотек

**Тестирование производительности**

генератор: gen\_tests.py

Формат входных данных

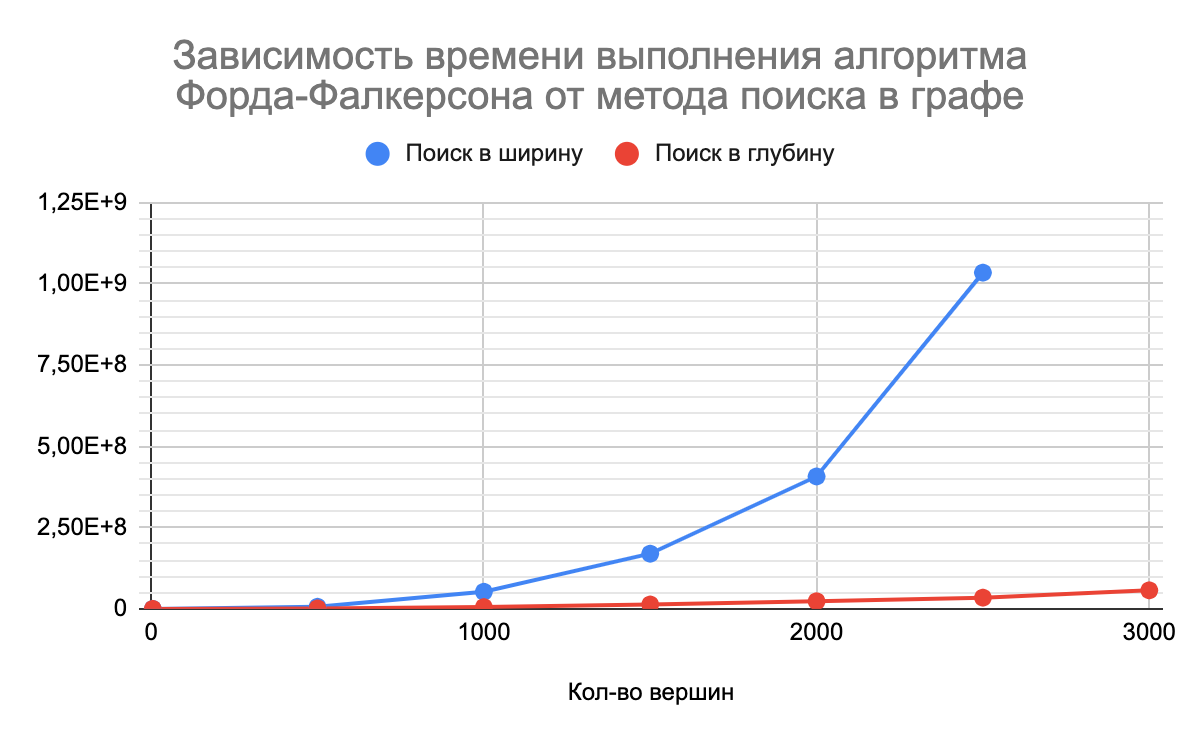
В первой строке задано n = N, m = N \* N, где N - количество вершин

В последующих n строках через пробел заданы три случайных числа

from, to, weight, таких, что from < to, 1 < weight < 20

**Таблица 1.** Зависимость времени выполнения алгоритма Форда-Фалкерсона от метода поиска в графе

|  |  |  |
| --- | --- | --- |
| Кол-во вершин | Поиск в ширину | Поиск в глубину |
| 5 | 484 | 497 |
| 500 | 6537544 | 1503381 |
| 1000 | 52655260 | 5599458 |
| 1500 | 169789801 | 13591139 |
| 2000 | 407439842 | 23495470 |
| 2500 | 1034432434 | 34597154 |
| 3000 | none | 57140525 |

****

**Вывод**

В работе был реализован алгоритм Форда-Фалкерсона.

Было проведено тестирование производительности с поиском в ширину и поиском в глубину. Усвоена на практике разница между двумя способами поиска в графе.

* BFS - использует структуру данных очереди для поиска кратчайшего пути. DFS - использует структуру данных стек.
* BFS можно использовать для поиска кратчайшего пути из одного источника в невзвешенном графе, потому что в BFS мы достигаем вершины с минимальным количеством ребер из исходной вершины.
* BFS больше подходит для поиска вершин, которые находятся ближе к заданному источнику. DFS больше подходит, когда есть решения вдали от источника.
* Временная сложность BFS и DFS составляет O(V + E), когда используется список смежности, и O (V^2), когда используется матрица смежности, где V обозначает вершины, а E обозначает ребра.