МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО АВТОНОМНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ

**«Национальный исследовательский технологический университет «МИСИС»**

**Институт Компьютерных Наук**

**Отчет**

**Задача построения максимального потока в сети. Алгоритм Форда-Фалкерсона**

**По курсу:** Комбинаторика и теория графов

**Ссылка на репозиторий:**

[**https://github.com/SadLiter/Combinatorics-and-graph-theory**](https://github.com/SadLiter/Combinatorics-and-graph-theory)

Волков Валентин Александрович

Группа БИВТ-23-6

**Отчет: Задача построения максимального потока в сети. Алгоритм Форда-Фалкерсона**

**Содержание**

1. Формальная постановка задачи
2. Теоретическое описание алгоритма и его характеристики
3. Сравнительный анализ с другими алгоритмами
4. Перечень инструментов, используемых для реализации
5. Описание реализации и процесса тестирования
6. Преимущества реализации на Python и C++
7. Заключение

**1. Формальная постановка задачи**

**Задача:**  
Построение максимального потока в сети, представленной ориентированным графом. Поток должен быть максимальным, удовлетворяя следующим условиям:

1. **Ограничение пропускной способности:** Поток по любому ребру не может превышать его пропускную способность.
2. **Сохранение потока:** Для каждой вершины, кроме истока и стока, сумма входящих потоков должна быть равна сумме исходящих потоков.

**Входные данные:**

* Ориентированный граф G=(V,E), где:
  + V — множество вершин;
  + E — множество рёбер с пропускными способностями c(u,v) ≥ 0 для каждого ребра (u,v) ∈ E
* Две выделенные вершины: исток s ∈ V и сток t ∈ V.

**Выходные данные:**  
Максимальный поток f, который можно передать из истока s в сток t.

**2. Теоретическое описание алгоритма и его характеристики**

**Описание алгоритма Форда-Фалкерсона**

Алгоритм Форда-Фалкерсона является жадным методом, использующим поиск путей увеличения потока. Основные шаги:

1. **Инициализация:**  
   Установить начальный поток f(u,v)=0 для всех рёбер (u,v) ∈ E.
2. **Поиск пути увеличения потока:**  
   Использовать поиск в ширину (BFS) или поиск в глубину (DFS) для нахождения пути от истока s до стока t, на котором ещё имеется остаточная пропускная способность.
3. **Увеличение потока:**  
   Найти минимальную остаточную пропускную способность вдоль найденного пути и увеличить поток на этом пути.
4. **Повторение:**  
   Повторять шаги 2 и 3, пока можно найти пути увеличения потока.

**Характеристики алгоритма**

* **Временная сложность:**  
  Зависит от метода поиска пути увеличения:
  + O(E⋅f), где f — величина максимального потока, если используется DFS.
  + O(V⋅E), если используется BFS (в этом случае алгоритм эквивалентен методу Эдмондса-Карпа).
* **Пространственная сложность:**  
  O(V^2), если граф представлен матрицей смежности, и O(V+E), если представлен списком смежности.
* **Применимость:**  
  Подходит для графов с малыми потоками f. Эффективен для разреженных графов.

**3. Сравнительный анализ с другими алгоритмами**

| **Критерий** | **Форд-Фалкерсон** | **Эдмондс-Карп** | **Диниц** |
| --- | --- | --- | --- |
| **Метод** | DFS/BFS + жадный поиск | BFS + улучшение | Уровневый граф + блокирующий поток |
| **Временная сложность** | O(E⋅f) | O(V⋅E^2) | O(V^2 E) |
| **Производительность** | Медленная для больших ff | Быстрая на разреженных графах | Высокая для плотных графов |
| **Сложность реализации** | Простая | Средняя | Сложная |

**Вывод:**

Алгоритм Форда-Фалкерсона уступает по производительности методам Эдмондса-Карпа и Диница, но прост в реализации и удобен для обучения.

**4. Перечень инструментов, используемых для реализации**

**Языки программирования:**

* Python 3.9+: Для быстрой разработки и тестирования.
* C++: Для оптимизированной производительности.

**Среда разработки:**

* Visual Studio Code (Python и C++).
* GCC/Clang (C++) для компиляции.

**Библиотеки:**

* Python:
  + collections.deque для BFS.
  + unittest для тестирования.
* C++:
  + Стандартная библиотека STL.

**5. Описание реализации и процесса тестирования**

**Реализация на Python**

Код реализован в файле ford\_fulkerson.py. Основные компоненты:

1. **Метод add\_edge(u, v, capacity):**  
   Добавляет ребро с заданной пропускной способностью и обратное ребро с нулевой пропускной способностью.
2. **Метод bfs(source, sink, parent):**  
   Выполняет поиск пути увеличения потока с использованием BFS.
3. **Метод max\_flow(source, sink):**  
   Вычисляет максимальный поток между истоком и стоком.
4. **Тестирование:**  
   Модуль test\_ford\_fulkerson.py проверяет корректность реализации.

**Реализация на C++**

Код на C++ представлен в файле ford\_fulkerson.cpp. Основные компоненты:

1. **Метод add\_edge(u, v, capacity):**  
   Добавляет прямое и обратное рёбра в граф.
2. **Метод bfs(source, sink, parent):**  
   Выполняет поиск пути увеличения потока.
3. **Метод max\_flow(source, sink):**  
   Возвращает величину максимального потока между истоком и стоком.
4. **Тестирование:**  
   Модуль test\_ford\_fulkerson.cpp проверяет корректность реализации с использованием assert.

**Пример тестирования**

**Входные данные:**

6 10

0 1 16

0 2 13

1 2 10

1 3 12

2 1 4

2 4 14

3 2 9

3 5 20

4 3 7

4 5 4

0 5

**Ожидаемый вывод:**

Максимальный поток: 23

**Тесты:**

* **Python:** Выполняются через unittest с использованием тестов в test\_ford\_fulkerson.py.
* **C++:** Выполняются через assert с использованием тестов в test\_ford\_fulkerson.cpp.

**6. Преимущества реализации на Python и C++**

**Python:**

* Удобство разработки.
* Легкость тестирования.
* Подходит для небольших графов и начального обучения.

**C++:**

* Высокая производительность.
* Эффективность для больших и плотных графов.
* Применимость в производственных системах.

**7. Заключение**

Алгоритм Форда-Фалкерсона является основным методом для построения максимального потока в сети. Его простота делает его подходящим для обучения и понимания концепции потоков в графах. Однако для задач с большими графами или большими потоками рекомендуется использовать более эффективные методы, такие как алгоритмы Диница или Эдмондса-Карпа.

**Основные выводы:**

1. **Python-реализация:** Удобна для разработки и тестирования.
2. **C++-реализация:** Подходит для обработки графов с большим количеством рёбер и вершин.
3. **Ограничения:** Медлительность на графах с большими потоками из-за линейной зависимости от величины потока.