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/Dmitry27912/Graphs

Голощапов Дмитрий Вячеславович

Группа БИВТ-23-6

Отчёт: Построение максимального потока в сети с использованием алгоритма Форда-Фалкерсона

Содержание

---

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 $c(u,v) \geq 0$ для каждого ребра $(u,v) \in E$.
- Две выделенные вершины: исток $s \in V$ и сток $t \in V$.

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 основан на жадном поиске путей увеличения. Основные шаги:

1. Находит пути увеличения из истока s в сток t с помощью поиска в ширину (BFS) или в глубину (DFS).

2. Вычисляет минимальную пропускную способность с вдоль найденного пути.
3. Увеличивает поток по пути на с.
4. Обновляет остаточные пропускные способности рёбер:
    - Уменьшает остаточную пропускную способность на с вдоль прямого пути.
    - Увеличивает остаточную пропускную способность на с вдоль обратного пути.
5. Повторяет процесс, пока существует путь увеличения.

Характеристики алгоритма:

- **Временная сложность**:
    - O(E·max_flow), где:
        - E — количество рёбер;
        - max_flow — значение максимального потока.
- **Пространственная сложность**:
    - O(V^2) при использовании матрицы смежности.
- **Применимость**:
    - Подходит для небольших графов и графов с малыми потоками.

---

3. Сравнительный анализ с аналогичными алгоритмами

Критерий	Форд-Фалкерсон	Диниц	Эдмондс-Карп
Метод	Жадный поиск путей увеличения	Уровневый граф + блокирующие потоки	BFS для кратчайших путей
Временная сложность	O(E·max_flow)	O(V2·E)	O(V·E2)
Скорость на практике	Медленная при больших потоках	Быстрая на плотных графах	Средняя
Сложность реализации	Простая	Средняя	Средняя
Применимость	Небольшие графы	Большие графы или графы с большими потоками	Универсальный

Вывод:
Алгоритм Форда-Фалкерсона прост в реализации и подходит для небольших графов. Однако для больших графов и графов с большими потоками он работает медленно из-за необходимости многократного поиска путей увеличения.

---

4. Перечень инструментов, используемых для реализации

Для реализации алгоритма Форда-Фалкерсона использовались следующие инструменты:

- **Язык программирования**: JavaScript.
- **Среда выполнения**: Node.js.

- **Редактор**: Visual Studio Code.
- **Модуль `fs`**: Для чтения входных данных из файла.
- **Jest**: Для автоматизированного тестирования алгоритма.

---

5. Описание реализации и процесса тестирования

Реализация алгоритма

Код алгоритма реализован в файле `ford_fulkerson.js`. Основные компоненты:

1. **Класс `Graph`**:
   - Представляет граф с пропускными способностями рёбер.
   - Методы:
     - `addEdge`: Добавляет ребро с заданной пропускной способностью.
     - `bfs`: Выполняет поиск в ширину для нахождения пути увеличения.
     - `maxFlow`: Возвращает максимальный поток между истоком и стоком.
2. **Функция `main`**:
   - Читает входные данные из файла `input.txt`, строит граф и вычисляет максимальный поток.

Пример входных данных:

```
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
```

Процесс тестирования

Тестирование проводилось с использованием библиотеки Jest. Были проверены следующие сценарии:

1. **Пустой граф**:
    - Ожидаемый результат: f = 0.
2. **Граф с одним ребром**:
    - Ожидаемый результат: f=c(u,v).
3. **Сложные графы**:
    - Проверка графов с несколькими путями и параллельными рёбрами.
4. **Большие графы**:
    - Тестирование на графах с тысячами рёбер для проверки производительности.

---

6. Преимущества реализации на JavaScript

1. **Преимущества**:
    - Простота разработки и отладки.
    - Кроссплатформенность благодаря Node.js.
    - Удобство тестирования с использованием Jest.
2. **Ограничения**:
    - Производительность уступает C++ при работе с большими графами.

---

7. Заключение

Алгоритм Форда-Фалкерсона успешно реализован на JavaScript и протестирован. Реализация демонстрирует корректность работы алгоритма на различных типах графов. Node.js обеспечивает кроссплатформенность и простоту настройки.

Основные выводы:

1. Алгоритм подходит для небольших графов и графов с малыми потоками.
2. Реализация на JavaScript удобна для разработки и тестирования, но уступает C++ по производительности на больших графах.