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 Диницы.

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 Диница (JavaScript)

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 Диница:
Алгоритм использует метод построения уровневого графа и поиска блокирующих потоков. Основные шаги:

1. **Построение уровневого графа (BFS)**:

    - Выполняется обход в ширину (BFS) от истока s, чтобы назначить каждому узлу уровень.
    - Если сток t недостижим, алгоритм завершает работу.
2. **Поиск блокирующего потока (DFS)**:
    - Выполняется обход в глубину (DFS), начиная с истока s, для нахождения путей до стока t в уровневом графе.
    - Потоки по найденным путям увеличиваются до тех пор, пока хотя бы одно ребро остаётся не полностью заполненным.
3. **Повторение**:
    - Если блокирующий поток был найден, уровневый граф перестраивается, и процесс повторяется.

Характеристики алгоритма:

- **Временная сложность**:
    - O(V^2 E) для общего случая.
- **Пространственная сложность**:
    - O(V+E) для хранения графа и уровневого графа.
- **Применимость**:
    - Эффективен для плотных графов и графов с большими потоками.

---

3. Сравнительный анализ с аналогичными алгоритмами

Критерий	Алгоритм Диница	Форд-Фалкерсон	Эдмондс-Карп
Временная сложность	O(V2E)O(V^2 E)	O(E·max_flow)O(E \cdot \text{max_flow})	O(V·E2)O(V \cdot E^2)
Подход	Уровневый граф	Любой путь увеличения	BFS для кратчайших путей
Скорость на практике	Быстрая	Медленная	Средняя
Сложность реализации	Средняя	Простая	Средняя
Применимость	Плотные графы	Графы с малыми потоками	Универсальный

Вывод:
Алгоритм Диница превосходит другие методы для графов с большими потоками и плотной структурой, используя уровневые графы и блокирующие потоки.

---

4. Перечень инструментов, используемых для реализации

Для реализации алгоритма Диница использовались следующие инструменты:

- **Язык программирования**: JavaScript.

- **Среда выполнения**: Node.js (для запуска алгоритма).
- **Редактор**: Visual Studio Code (с расширением для JavaScript).
- **Модуль fs**: Для чтения входных данных из файла.
- **Jest**: Для автоматизированного тестирования алгоритма.

---

5. Описание реализации и процесса тестирования

Реализация алгоритма

Код алгоритма Диница реализован в файле `dinic.js`. Основные компоненты:

1. **Класс Edge**:
   - Представляет ребро графа с пропускной способностью, текущим потоком и обратным ребром.
2. **Класс Dinic**:
   - Методы:
     - `addEdge`: Добавляет прямое и обратное рёбра.
     - `buildLevelGraph`: Строит уровневый граф с помощью BFS.
     - `sendFlow`: Выполняет DFS для нахождения блокирующего потока.
     - `maxFlow`: Возвращает максимальный поток между истоком и стоком.
3. **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, циклами и параллельными рёбрами.
4. **Большие графы**:
    - Тестирование на графах с тысячами рёбер для проверки производительности.

---

6. Преимущества реализации на JavaScript

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

---

7. Заключение

Алгоритм Диница успешно реализован на JavaScript и протестирован. Реализация демонстрирует эффективность алгоритма на графах с большими потоками и плотной структурой. Node.js позволяет запускать алгоритм на любой платформе с минимальными настройками. Автоматические тесты на Jest подтверждают корректность работы реализации.