

**Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ «МИСИС»**

Комбинаторика и теория графов

Отчет по теме

«Задача построения максимального потока в сети. Алгоритм Диницы»

Выполнил: Мочалов Артём Владимирович

Группа: БИВТ-23-1

Ссылка на репозиторий: <https://github.com/dxxmn/KITG>

Москва 2024

Оглавление

1. Введение.....	3
2. Формальная постановка задачи.....	3
1. Ограничение пропускной способности.....	3
2. Сохранение потока.....	4
3. Максимизация потока.....	4
4. Ограничения и допущения.....	4
3. Теоретическое описание алгоритма Диницы.....	5
1. Основная идея алгоритма:.....	5
2. Уровневая сеть.....	5
3. Временная сложность.....	5
4. Пространственная сложность.....	5
4. Сравнительный анализ с другими алгоритмами.....	6
1. Алгоритм Форда-Фалкерсона.....	6
2. Алгоритм Эдмондса-Карпа.....	6
3. Алгоритм Push-Relabel.....	6
4. Сравнение эффективности и практичности.....	7
5. Используемые инструменты для реализации.....	7
1. JavaScript.....	8
2. Node.js.....	8
3. npm (Node Package Manager).....	8
4. Jest.....	9
5. WebStorm.....	9
6. Git.....	9
7. ESLint и Prettier.....	10
6. Описание реализации и процесса тестирования.....	10
1. Структура данных.....	10
2. Основные компоненты.....	10
3. Процесс тестирования.....	11
4. Описание тестов.....	12
5. Результаты тестирования.....	12
7. Заключение.....	12

1. Введение

Максимальный поток в сети — это задача, которая имеет широкий спектр применения в таких областях, как логистика, распределение ресурсов, компьютерные и телекоммуникационные сети, а также в решении задач, связанных с транспортными системами. В общих чертах задача заключается в нахождении максимально возможного объема потока, который может быть передан из одной точки (истока) в другую (стока) через сеть каналов с ограниченной пропускной способностью. Например, в транспортных сетях это может означать максимальное количество товаров, которое можно доставить из одного города в другой с учетом ограничений на грузоподъемность дорог.

Алгоритм Диницы решает задачу нахождения максимального потока путем использования уровней сети и поиска блокирующих потоков. Основная идея заключается в том, что сеть разделяется на уровни, и на каждом уровне ищутся пути для увеличения потока. Этот итеративный процесс продолжается до тех пор, пока не будет найдено решение задачи, то есть пока не будут исчерпаны все возможные увеличивающие пути.

В этом отчете будет рассмотрен алгоритм Диницы, его применение, теоретические характеристики, а также анализ сложности. Помимо этого, проведен сравнительный анализ с другими алгоритмами, решающими задачу максимального потока, такими как Форда-Фалкерсона, Эдмондса-Карпа и алгоритм Push-Relabel.

2. Формальная постановка задачи

Задача максимального потока в сети заключается в следующем. Пусть имеется направленный граф $G = (V, E)$, где V — множество вершин, а E — множество ребер. Каждое ребро $e \in E$ имеет неотрицательную пропускную способность $c(e)$, которая обозначает максимальный объем потока, который может пройти по этому ребру. Кроме того, в графе имеется две выделенные вершины: исток s и сток t . Необходимо определить максимальный поток, который можно передать из истока в сток при соблюдении следующих ограничений:

1. Ограничение пропускной способности. Поток через любое ребро не должен превышать его пропускную способность:

$$f(e) \leq c(e) \quad \forall e \in E$$

где $f(e)$ — это поток через ребро e .

2. Сохранение потока. Для каждой вершины $v \in V$, кроме истока и стока, сумма потоков, входящих в вершину, должна равняться сумме потоков, выходящих из вершины. Это условие записывается как:

$$\sum_{e \in \text{in}(v)} f(e) = \sum_{e \in \text{out}(v)} f(e) \quad \forall v \in V, v \neq s, t$$

3. Максимизация потока. Необходимо максимизировать общий поток F из истока в сток:

$$F = \sum_{e \in \text{out}(s)} f(e) - \sum_{e \in \text{in}(s)} f(e)$$

Цель задачи — найти такое распределение потоков по сети, которое удовлетворяет вышеперечисленным условиям и при этом максимизирует общий поток.

4. Ограничения и допущения

При формализации задачи максимального потока предполагаются следующие упрощения и ограничения:

1. Целостность потока: Потоки рассматриваются как непрерывные величины, что позволяет использовать вещественные числа для их представления.
2. Детерминированность: Пропускные способности рёбер известны и постоянны на протяжении всего процесса передачи потока.
3. Ориентированность рёбер: Направление рёбер фиксировано, и поток может перемещаться только в указанном направлении.
4. Отсутствие циклов с отрицательным потоком: В силу неотрицательности потоков такие циклы невозможны.
5. Независимость рёбер: Потоки по различным рёбрам не взаимодействуют друг с другом напрямую, однако ограничение пропускных способностей обеспечивает косвенную взаимосвязь.

Формальная постановка задачи предоставляет основу для понимания того, как алгоритм функционирует и какие условия должны быть выполнены для корректного его применения.

3. Теоретическое описание алгоритма Диницы

Алгоритм Диницы является одним из наиболее эффективных методов решения задачи максимального потока. Он использует концепцию уровней сети и блокирующих потоков для более эффективного нахождения максимального потока.

1. Основная идея алгоритма:

1. Инициализация: начать с нулевого потока во всех ребрах сети.
2. Построение уровневой сети: разделить сеть на уровни, где каждый уровень содержит вершины, до которых можно добраться из истока за определенное количество шагов.
3. Поиск блокирующего потока: найти поток, который блокирует все пути из истока в сток в текущей уровневой сети.
4. Обновление потока: увеличить поток в сети на величину блокирующего потока.
5. Повторение: повторять шаги 2-4 до тех пор, пока не будет достигнут сток в уровневой сети.
6. Завершение: когда нет больше путей из истока в сток в уровневой сети, алгоритм завершается, и текущий поток является максимальным потоком в сети.

2. Уровневая сеть

Уровневая сеть — это ключевое понятие в алгоритме Диницы. Она представляет собой граф, в котором вершины разделены на уровни в зависимости от расстояния от истока. Это позволяет более эффективно искать пути для увеличения потока, так как на каждом уровне рассматриваются только те вершины, которые находятся на одном и том же или на следующем уровне.

3. Временная сложность

Временная сложность алгоритма Диницы составляет $O(V^2 \cdot E)$, где V — количество вершин, а E — количество ребер. Это делает его одним из самых эффективных алгоритмов для задач максимального потока в плотных сетях.

4. Пространственная сложность

Пространственная сложность алгоритма Диницы также зависит от количества вершин и ребер в графе. Основные структуры данных, используемые в алгоритме, включают в себя массивы для хранения уровневой сети и массива для поиска путей, что приводит к

пространственной сложности $O(V + E)$, где V — количество вершин, а E — количество ребер.

Алгоритм Диницы является продвинутым методом для решения задачи максимального потока в транспортной сети. Его теоретические характеристики делают его основой для понимания более сложных и эффективных алгоритмов, таких как алгоритм Эдмондса-Карпа и алгоритм Push-Relabel, которые будут рассмотрены дальше.

4. Сравнительный анализ с другими алгоритмами

Алгоритм Диницы является одним из наиболее эффективных методов для нахождения максимального потока в транспортной сети. Однако, существует несколько иных алгоритмов, которые также решают задачу максимального потока, каждый из которых обладает своими особенностями, преимуществами и недостатками. В данном разделе проводится сравнительный анализ алгоритма Диницы с основными аналогами.

1. Алгоритм Форда-Фалкерсона

Алгоритм Форда-Фалкерсона является одним из первых и наиболее известных методов для решения задачи максимального потока. Он использует итеративный подход для поиска увеличивающих путей и обновления потока. Временная сложность алгоритма Форда-Фалкерсона составляет $O(F \cdot E)$, где F — максимальный поток, а E — количество ребер. Несмотря на простоту реализации, алгоритм может работать медленно на больших графах.

2. Алгоритм Эдмондса-Карпа

Алгоритм Эдмондса-Карпа является модификацией алгоритма Форда-Фалкерсона, в котором для поиска увеличивающих путей используется поиск в ширину (BFS) вместо поиска в глубину (DFS). Это изменение приводит к улучшению времени выполнения в худших случаях. Временная сложность алгоритма Эдмондса-Карпа составляет $O(V \cdot E^2)$, где V — количество вершин, а E — количество ребер. Несмотря на большую сложность по количеству ребер, этот алгоритм на практике часто работает быстрее для плотных графов.

3. Алгоритм Push-Relabel

Алгоритм префлоу-пуш, предложенный Эдгаром Фиоренцини и Роберто Фишером, использует стратегию локальных операций "пуш" и "релейбл" для увеличения потоков в сети. В отличие от методов, основанных на поиске увеличивающих путей, этот алгоритм работает с префлоу — состоянием, где сумма входящих потоков может превышать сумму

исходящих. Временная сложность в лучшем случае составляет $O(V^2 \cdot \sqrt{V})$, также для хранения префлоу и меток вершин необходима дополнительная память.

4. Сравнение эффективности и практичности

При сравнении алгоритмов важно учитывать не только теоретическую временную и пространственную сложность, но и практическую эффективность на различных типах графов:

- Редкие vs. Плотные графы: Алгоритмы, такие как Эдмондс-Карпа и Диница, демонстрируют лучшую производительность на плотных графах, где количество ребер велико. В то же время алгоритмы префлоу-пуш могут быть более эффективны на разреженных графах.

- Размер графа: На очень больших графах алгоритмы с низкой асимптотической сложностью, такие как Диница или префлоу-пуш, показывают значительные преимущества.

- Структура сети: Некоторые алгоритмы лучше справляются с сетями, имеющими специфические структуры, например, с сетями с множеством параллельных путей или с узкими местами.

- Простота реализации: Алгоритм Эдмондса-Карпа проще в реализации, особенно в образовательных целях или в системах с ограниченными требованиями к производительности.

- Поддержка динамических изменений: Алгоритмы префлоу-пуш более гибки в условиях динамически изменяющихся сетей, где требуется частое обновление потоков.

- Параллельность: Некоторые современные реализации алгоритмов максимального потока, такие как параллельные версии префлоу-пуш, могут эффективно использовать многопроцессорные системы, что важно для высокопроизводительных вычислений.

Выбор конкретного алгоритма для решения задачи максимального потока зависит от особенностей рассматриваемой задачи, типа и размера графа, а также от требований к производительности и ресурсам системы. Понимание преимуществ и ограничений каждого алгоритма позволяет выбрать наиболее подходящий метод для конкретного применения, обеспечивая эффективное решение задачи максимального потока в различных условиях.

5. Используемые инструменты для реализации

Для успешной реализации и тестирования алгоритма Диницы были использованы различные инструменты и технологии, каждый из которых сыграл ключевую роль в обеспечении эффективности разработки, повышения качества кода и упрощения процесса

тестирования. Ниже представлены основные инструменты вместе с их преимуществами в контексте задачи.

1. JavaScript

- Универсальность: JavaScript является универсальным языком программирования, позволяющим разрабатывать как фронтенд, так и бэкенд приложения. В контексте задачи построения алгоритма максимального потока, JavaScript обеспечивает гибкость при реализации логики алгоритма.

- Большое сообщество и обширные ресурсы: Благодаря широкому сообществу разработчиков, доступно множество библиотек и инструментов, упрощающих разработку и отладку алгоритмов.

- Асинхронность: Встроенная поддержка асинхронных операций позволяет эффективно обрабатывать большие объемы данных и улучшать производительность приложения.

2. Node.js

- Серверная платформа на базе JavaScript: Node.js позволяет запускать JavaScript-код на сервере, что упрощает разработку полноценных приложений без необходимости использования различных языков для фронтенда и бэкенда.

- Высокая производительность: Благодаря V8-движку и неблокирующей модели ввода-вывода, Node.js обеспечивает высокую производительность и масштабируемость, что важно для обработки больших графов в алгоритме Диницы.

- Расширяемость: Обширная экосистема модулей через npm позволяет быстро интегрировать необходимые библиотеки и инструменты, ускоряя процесс разработки.

3. npm (Node Package Manager)

- Управление зависимостями: npm предоставляет удобный способ управления зависимостями проекта, позволяя легко устанавливать, обновлять и удалять пакеты, необходимые для реализации алгоритма.

- Большое количество пакетов: С помощью npm доступно огромное количество пакетов, включая тестовые фреймворки, инструменты для сборки и линтинга, что упрощает интеграцию различных инструментов в проект.

- Автоматизация: Возможность использования скриптов npm для автоматизации различных задач, таких как запуск тестов, сборка проекта и форматирование кода, повышает эффективность разработки.

4. Jest

- Простота использования: Jest предоставляет интуитивно понятный API для написания и запуска тестов, что упрощает процесс разработки тестов для алгоритма.
- Быстрота и производительность: Jest оптимизирован для быстрого выполнения тестов, что особенно важно при большом количестве тестовых случаев.
- Поддержка снимков (Snapshot Testing): Возможность создания снимков состояния приложения облегчает проверку корректности изменений в коде.
- Встроенная поддержка покрытия кода: Jest интегрируется с инструментами для измерения покрытия кода, что позволяет контролировать полноту тестирования алгоритма.

5. WebStorm

- Мощная IDE для JavaScript: WebStorm предлагает широкий набор инструментов для разработки на JavaScript, включая автодополнение, рефакторинг и интеграцию с системами контроля версий.
- Отладка и тестирование: Встроенные инструменты для отладки и запуска тестов позволяют эффективно выявлять и исправлять ошибки в реализации алгоритма.
- Интеграция с инструментами разработки: Поддержка таких инструментов, как ESLint, Prettier и Git, упрощает настройку рабочего процесса и поддержание высокого качества кода.
- Интеллектуальные функции: Функции, такие как анализ кода на лету и навигация по проекту, повышают производительность разработчика и облегчают работу над сложными алгоритмами.

6. Git

- Система контроля версий: Git позволяет отслеживать изменения в коде, что облегчает управление различными версиями реализации алгоритма и способствует командной работе.
- Ветвление и слияние: Возможность создания веток для разработки новых функций или исправления ошибок без риска нарушить основную версию кода.
- История изменений: Подробная история коммитов позволяет отслеживать эволюцию проекта и быстро находить места, где были внесены изменения.
- Интеграция с платформами: Поддержка интеграции с такими платформами, как GitHub и GitLab, упрощает совместную работу и автоматизацию процессов развертывания и тестирования.

7. ESLint и Prettier

- Статический анализ кода и автоматическое форматирование: ESLint и Prettier помогает выявлять потенциальные ошибки и нарушения стиля кодирования на ранних этапах разработки, что предотвращает появление дефектов в реализации алгоритма.
- Настраиваемость: Возможность настройки правил проверки и форматирования в соответствии с требованиями проекта позволяет поддерживать единый стиль кода и стандарты качества.
- Интеграция с IDE: Поддержка интеграции с WebStorm и другими редакторами обеспечивает автоматическую проверку кода во время написания, повышая эффективность работы разработчика.
- Расширяемость: Большое количество доступных плагинов и конфигураций позволяет адаптировать ESLint и Prettier под нужды проекта.

Использование перечисленных инструментов обеспечило эффективную разработку и тестирование алгоритма Диницы. JavaScript и Node.js предоставили гибкую и производительную среду для реализации алгоритма, а npm облегчила управление зависимостями и интеграцию дополнительных библиотек. Jest позволил создать надежную систему тестирования, обеспечивая высокое покрытие кода и обнаружение дефектов на ранних стадиях разработки. WebStorm предоставил мощную интегрированную среду разработки, ускоряя процесс кодирования и отладки. Git обеспечил надежное управление версиями. ESLint и Prettier способствовали поддержанию высокого качества кода, снижая вероятность ошибок и улучшая читаемость. В совокупности эти инструменты создали прочную основу для разработки надежного и эффективного решения задачи построения максимального потока в сети.

6. Описание реализации и процесса тестирования

1. Структура данных

Для представления графа использован вид списка смежности, где каждая вершина хранит список исходящих ребер с информацией о целевой вершине и максимальной пропускной способности. Также для реализации уровневой сети и обратных ребер использованы соответствующие структуры данных.

2. Основные компоненты

1. Класс Graph:

-Хранит информацию о вершинах и ребрах.

-Предоставляет методы для добавления ребер и получения соседей.

2. Функция Dinic:

-Принимает граф, источник и сток.

-Инициализирует поток.

-Повторно строит уровневую сеть и ищет блокирующие потоки.

-Обновляет поток и остаточную сеть.

-Возвращает максимальный поток.

3. Поиск пути увеличения:

-Используется поиск в ширину (BFS) для построения уровневой сети.

-Альтернативно можно использовать итеративный BFS для реализации алгоритма Эдмондса-Карпа.

4. Поиск блокирующего потока:

-Используется рекурсивный DFS для поиска пути от источника к стоку в уровневой сети.

-Альтернативно можно использовать итеративный BFS для реализации алгоритма Эдмондса-Карпа.

5. Обновление потоков:

-После нахождения пути увеличения определяется минимальная пропускная способность на пути.

-Поток на ребрах пути увеличивается, а на обратных ребрах уменьшается.

3. Процесс тестирования

Цель тестирования: Проверка корректности реализации алгоритма Диницы на различных типах графов, включая типичные случаи, граничные условия и специальные сценарии.

Подход к тестированию:

1. Разработка тест-кейсов: Создание нескольких графов с известными максимальными потоками для проверки правильности реализации.

2. Автоматизация тестирования: Написание функций тестирования, которые создают графы, запускают алгоритм и сравнивают полученный результат с ожидаемым.

3. Проверка различной структуры графов:

- Простые графы с одним путем между источником и стоком

- Графы с несколькими параллельными путями.
- Графы с обратными ребрами и циклами.
- Графы, где максимальный поток равен нулю (например, отсутствие путей между источником и стоком).
- Графы с разными пропускными способностями, включая большие и малые значения.

4. Описание тестов

Тест 1: Проверка простого графа, где есть два непротиворечивых пути от источника к стоку. Ожидаемый максимальный поток — сумма пропускных способностей обоих путей.

Тест 2: Проверка графа с обратными ребрами, которые могут влиять на поток через перераспределение потоков.

Тест 3: Проверка графа без путей от источника к стоку, ожидание максимального потока равного нулю.

Тест 4: Проверка графа с циклом, чтобы убедиться, что алгоритм правильно обрабатывает циклические потоки.

Тест 5: Проверка большего графа с несколькими возможными путями и пересекающимися потоками, чтобы оценить эффективность алгоритма.

5. Результаты тестирования

Каждый тест создает отдельный граф, запускает алгоритм и сравнивает полученный результат с ожидаемым. В случае несоответствия выводится сообщение об ошибке с подробной информацией. После успешного прохождения всех тестов выводится сообщение об успешном прохождении всех тестов.

7. Заключение

Задача построения максимального потока в сети является фундаментальной в области комбинаторики и теории графов, имея широкий спектр применений в реальных задачах, таких как логистика, распределение ресурсов и сетевые коммуникации. Алгоритм Диницы, несмотря на свою относительную сложность, обеспечивает эффективное решение этой задачи, используя итеративный подход для поиска и увеличения потоков в сети.

Однако, учитывая его зависимость от метода поиска блокирующих потоков и потенциальную неэффективность на больших графах, важно рассмотреть и другие алгоритмы, такие как Форда-Фалкерсона, Эдмондса-Карпа и Push-Relabel, каждый из которых предлагает свои преимущества и оптимизации.

В процессе реализации и тестирования алгоритма Диницы были использованы современные инструменты разработки и тестирования, такие как JavaScript, Node.js, Jest и WebStorm, что обеспечило высокую производительность и надежность кода. Проведенные тесты подтвердили корректность работы алгоритма на различных типах графов, включая сложные сценарии с обратными ребрами и циклами.

В целом, алгоритм Диницы остается важным инструментом в арсенале разработчиков, однако выбор оптимального алгоритма для конкретной задачи требует учета особенностей графа и требований к производительности.