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

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:

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

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:

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

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

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

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. Подробная информация о преимуществах каждого инструмента представлена в отчете на тему «Задача построения максимального потока в сети. Алгоритм Форда-Фалкерсона» Ниже представлены основные инструменты:

1. **JavaScript**
2. **Node.js**
3. **npm (Node Package Manager)**
4. **Jest**
5. **WebStorm**
6. **Git**
7. **ESLint и Prettier**

Использование перечисленных инструментов обеспечило эффективную разработку и тестирование алгоритма Диницы.

6. Описание реализации и процесса тестирования

1. Основные компоненты

1. Класс Graph:

```
class Graph {
    constructor(numberOfVertices) {
        this.V = numberOfVertices;
        this.adjMatrix = Array.from({ length: this.V }, () =>
Array(this.V).fill(0));
    }

    addEdge(from, to, capacity) {
        this.adjMatrix[from][to] = capacity;
    }

    getCapacity(from, to) {
        return this.adjMatrix[from][to];
    }

    setCapacity(from, to, capacity) {
        this.adjMatrix[from][to] = capacity;
```

```
    }
}
```

- Класс Graph предназначен для представления графа с использованием матрицы смежности. Основная цель класса — моделирование структуры графа, где вершины соединены ребрами с определенной пропускной способностью.
- constructor(numberOfVertices): Инициализирует граф с заданным количеством вершин. Создает матрицу смежности размером numberOfVertices × numberOfVertices, заполненную нулями.
- addEdge(from, to, capacity): Добавляет ребро между вершинами from и to с указанной пропускной способностью capacity.
- getCapacity(from, to): Возвращает пропускную способность ребра между вершинами from и to.
- setCapacity(from, to, capacity): Устанавливает новую пропускную способность capacity для ребра между вершинами from и to.

2. Функция Dinitz:

```
class Dinitz {
    constructor(graph) {
        this.graph = graph;
        this.level = [];
    }

    bfs(source, sink) {
        this.level = Array(this.graph.V).fill(-1);
        const queue = [];
        queue.push(source);
        this.level[source] = 0;

        while (queue.length > 0) {
            const u = queue.shift();
            for (let v = 0; v < this.graph.V; v++) {
                if (this.level[v] < 0 && this.graph.getCapacity(u, v) > 0) {
                    this.level[v] = this.level[u] + 1;
                    queue.push(v);
                }
```

```
            }
        }

        return this.level[sink] >= 0;
    }

    dfs(u, sink, flow, start) {
        if (u === sink) {
            return flow;
        }

        for (let v = start[u]; v < this.graph.V; v++) {
            if (this.level[v] === this.level[u] + 1 &&
this.graph.getCapacity(u, v) > 0) {
                const currentFlow = Math.min(flow, this.graph.getCapacity(u,
v));
                const tempFlow = this.dfs(v, sink, currentFlow, start);
                if (tempFlow > 0) {
                    this.graph.setCapacity(u, v, this.graph.getCapacity(u, v) -
tempFlow);
                    this.graph.setCapacity(v, u, this.graph.getCapacity(v, u) +
tempFlow);
                    return tempFlow;
                }
            }
            start[u]++;
        }

        return 0;
    }

    maxFlow(source, sink) {
        if (source === sink) {
            return 0;
        }

        let totalFlow = 0;
```

```
        while (this.bfs(source, sink)) {
            const start = Array(this.graph.V).fill(0);

            let flow;
            do {
                flow = this.dfs(source, sink, Infinity, start);
                totalFlow += flow;
            } while (flow > 0);
        }

        return totalFlow;
    }
}
```

-Класс Dinitz реализует алгоритм Диница для нахождения максимального потока в графе.
-constructor(graph): Инициализирует объект алгоритма Диница, принимая в качестве аргумента граф (graph), в котором будет производиться поиск максимального потока.
-bfs(source, sink): Выполняет поиск в ширину (BFS) для построения слоистой сети (level graph). Определяет уровни вершин от истока (source) и проверяет, достижим ли сток (sink). Возвращает true, если сток достижим, и false в противном случае.
-dfs(u, sink, flow, start): Выполняет поиск в глубину (DFS) для нахождения блокирующего потока в слоистой сети. Обновляет остаточные пропускные способности ребер и возвращает значение потока, которое можно протолкнуть от текущей вершины (u) до стока (sink).
-maxFlow(source, sink): Вычисляет максимальный поток от истока (source) к стоку (sink). Использует BFS для построения слоистой сети и DFS для нахождения блокирующего потока. Возвращает значение максимального потока.

2. Пример работы

Рассмотрим пример работы алгоритма на следующем графе (рис. 1). Сначала по алгоритму Диницы нам необходимо построить уровневую сеть, для начального графа она будет выглядеть следующим образом (рис. 2). Затем поиском в глубину мы находим первый увеличивающийся путь (рис. 3), им является 0 -> 1 -> 3 -> 5, затем по этому пути пускаем поток, ограниченный пропускной способностью ребра 1 -> 3, тогда итоговый поток на этом этапе будет равен 12, также обновляем пропускные способности ребер.

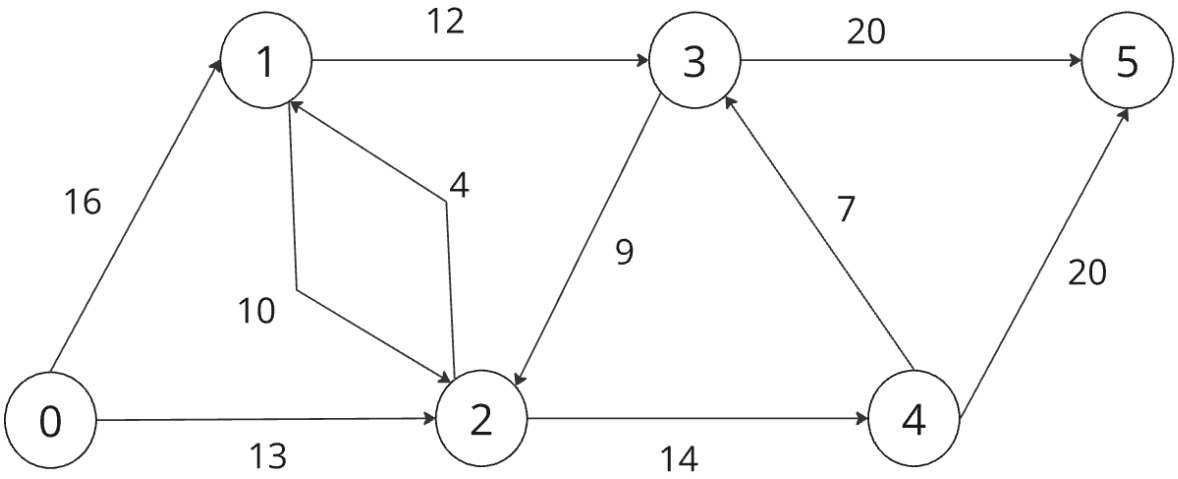

Рисунок 1 – Изначальный граф

Следующим этапом пустим поток по следующему увеличивающему пути, а именно 0 -> 2 -> 4 -> 5 (рис. 4), поток, который мы можем пустить по этому пути, будет ограничен пропускной способностью ребра 0 -> 2, тогда суммарный поток на этом этапе будет равен 25, затем также как и в первом шаге обновляем пропускные способности ребер.

В текущей уровневой сети больше нету путей из стока в сток, значит нужно заново построить уровневую сеть (рис. 5). На рисунке 5 бирюзовым цветом показаны ребра, по которым нельзя пустить поток, значит в построении уровневой сети они участвовать не будут. Теперь в этой уровневой сети у нас есть увеличивающий путь 0 -> 1 -> 2 -> 4 -> 5 и мы можем пустить по нему поток равный 1, потому что он ограничен пропускной способностью ребра 2->4. Тогда максимальный поток для этого графа будет равен 26 (рис. 6)

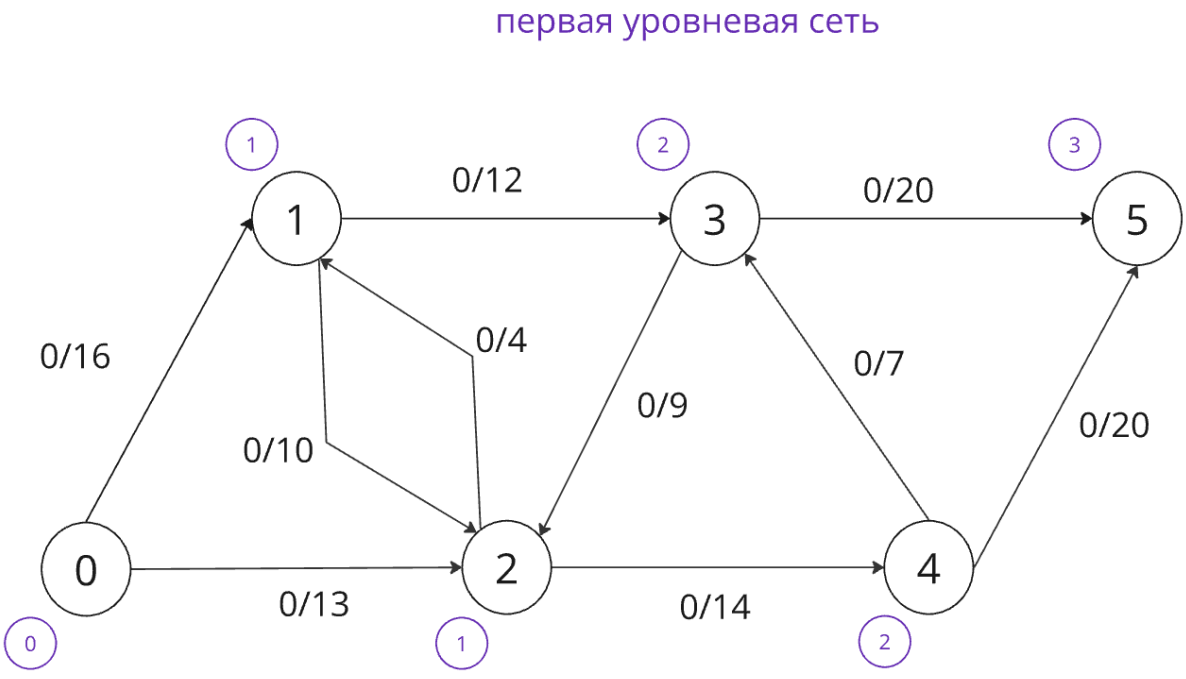

Рисунок 2 – Построение первой уровневой сети

totalFlow = 0 + 12 = 12

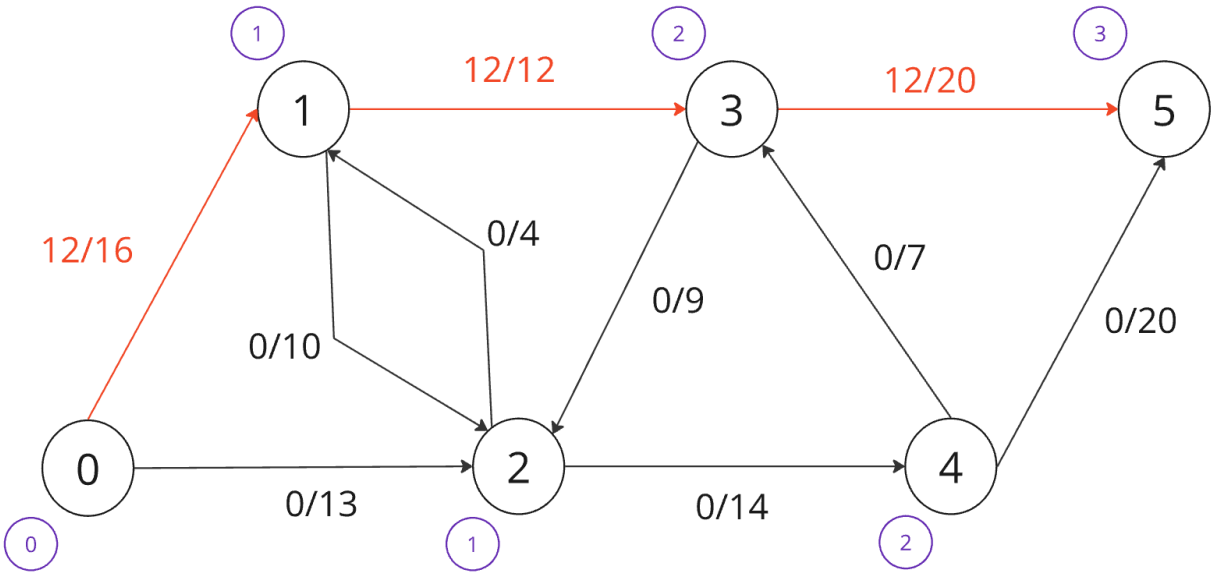

Рисунок 3 – Первый увеличивающий путь

totalFlow = 12 + 13 = 25

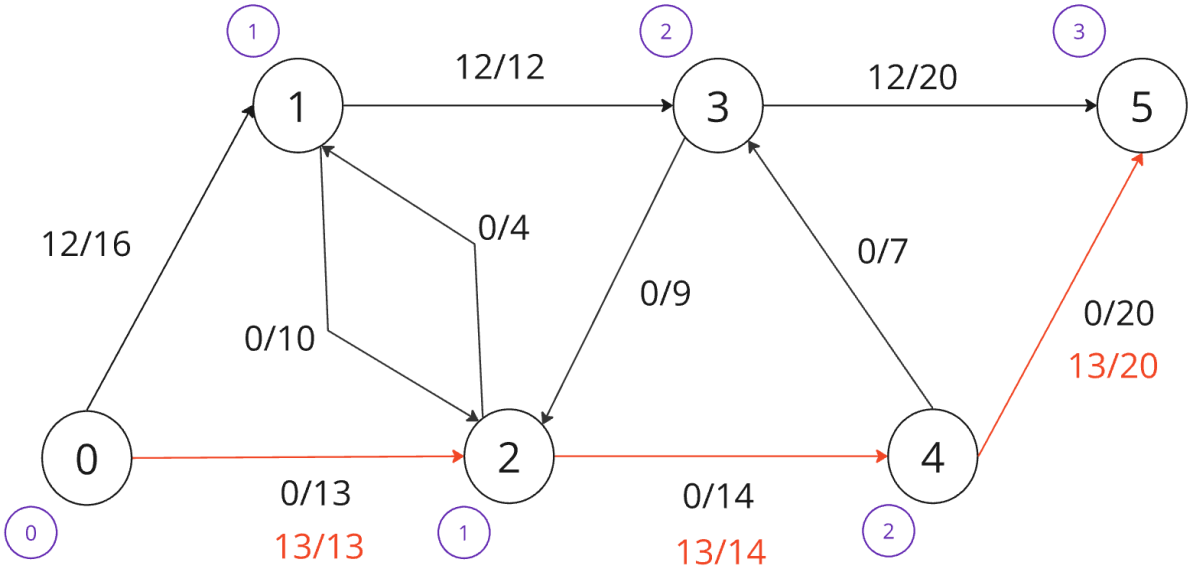

Рисунок 4 – Второй увеличивающий путь

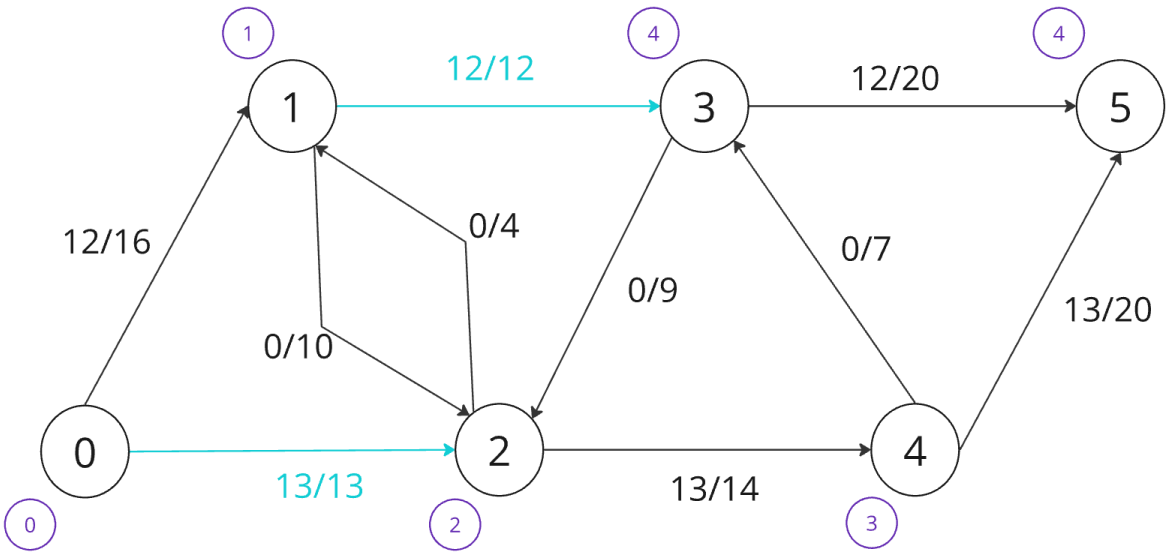

Рисунок 5 – Вторая уровневая сеть

totalFlow = 25 + 1 = 26

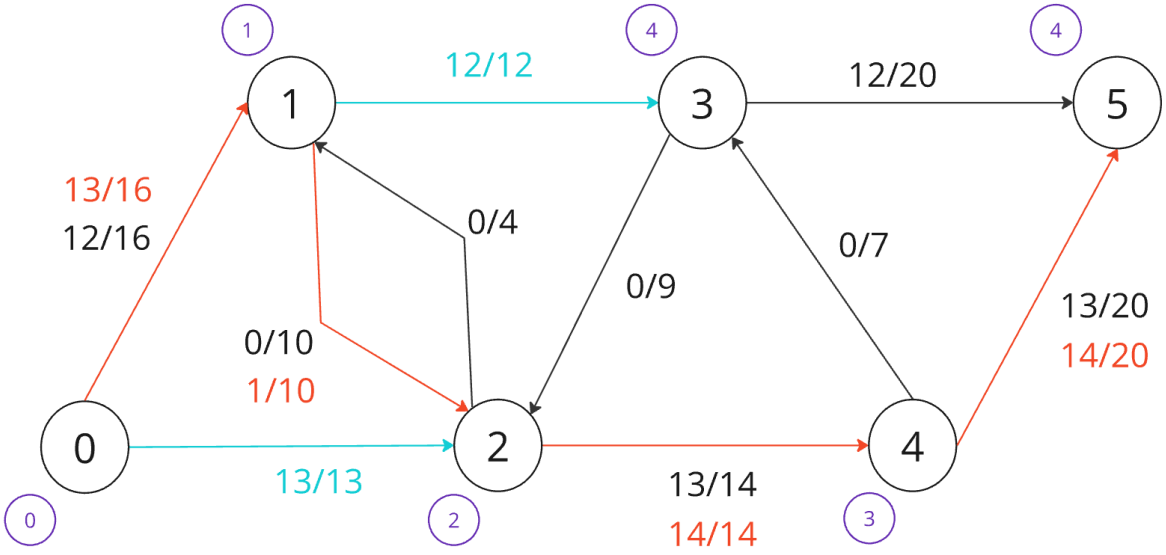

Рисунок 6 – Третий увеличивающий путь

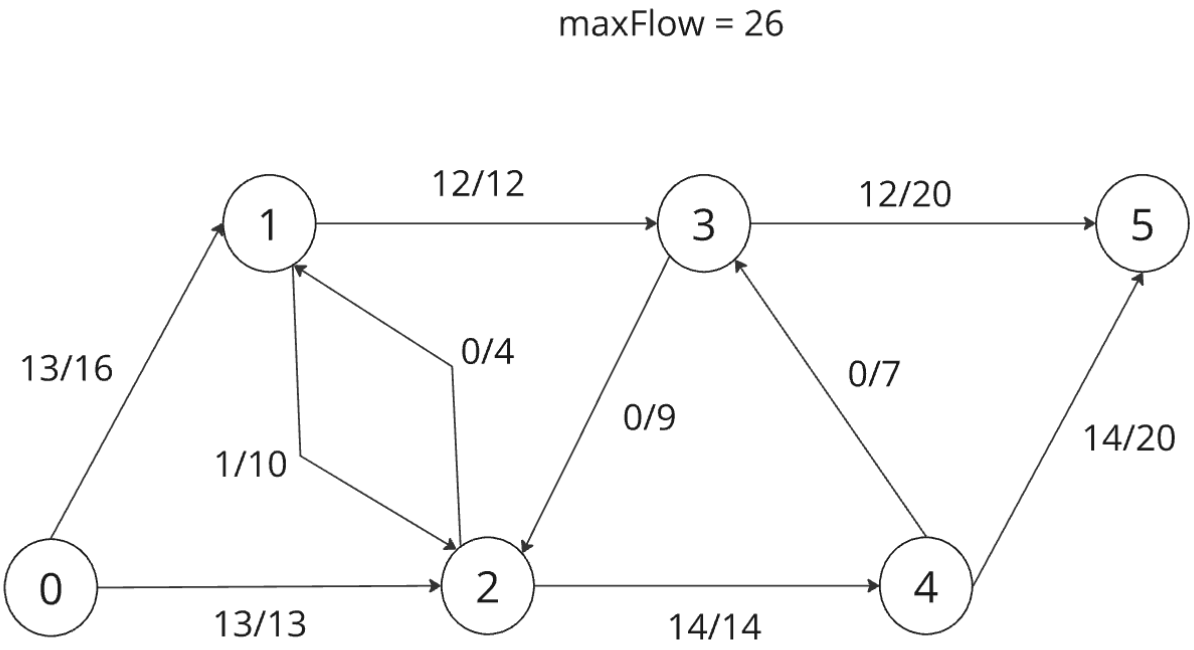

Рисунок 7 – Итоговый граф

3. Процесс тестирования

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

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. Оценка временной сложности для графов с разным количеством вершин и ребер.

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.

Тест 6: Проверка временной сложности алгоритма Диницы. Тест включает в себя сравнение теоретической и фактической временной сложности алгоритма, полученных при нахождении максимального потока в трех различных графах.

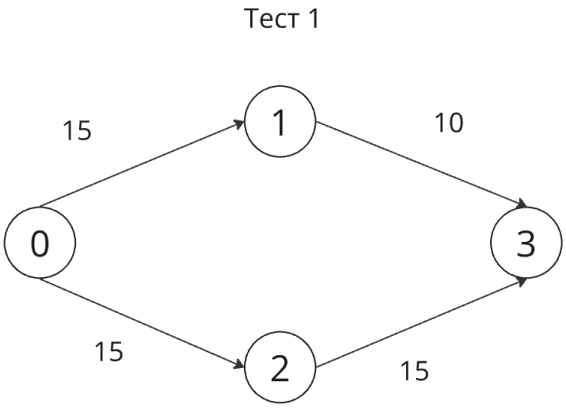

Рисунок 8 – Тест 1

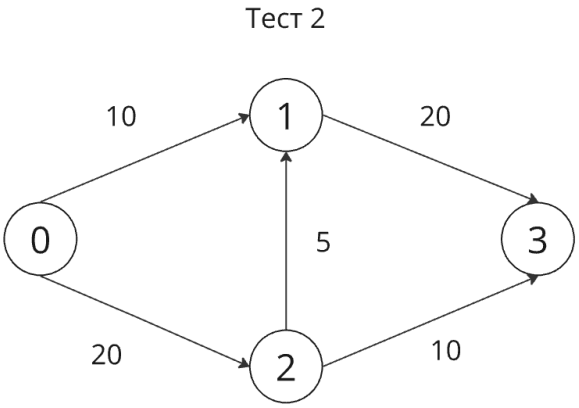

Рисунок 9 – Тест 2

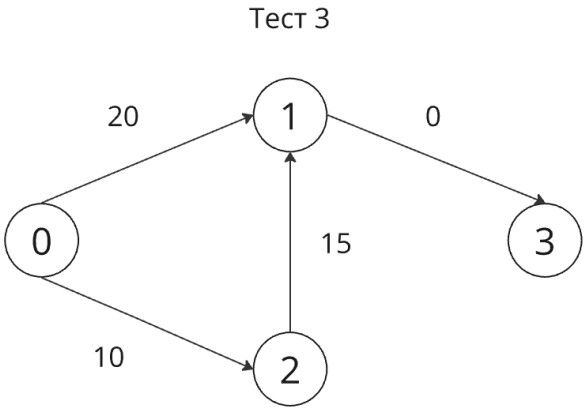

Рисунок 10 – Тест 3

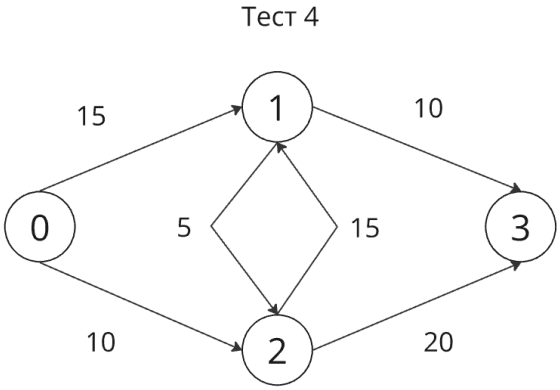

Рисунок 11 – Тест 4

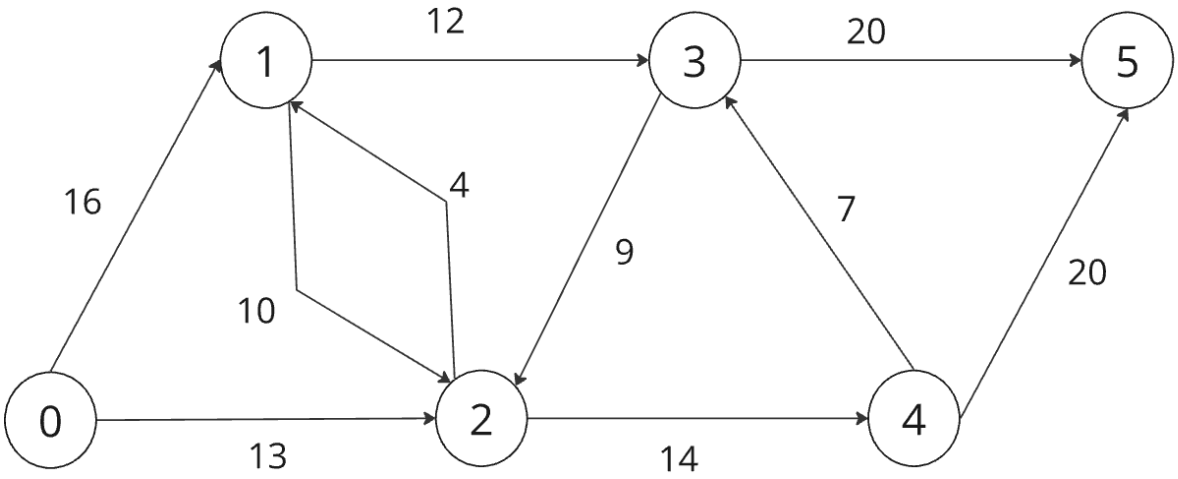

Рисунок 12 – Тест 5

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. В выводе консоли виден результат теста производительности, а именно сравнение теоретической и фактической временной сложности.

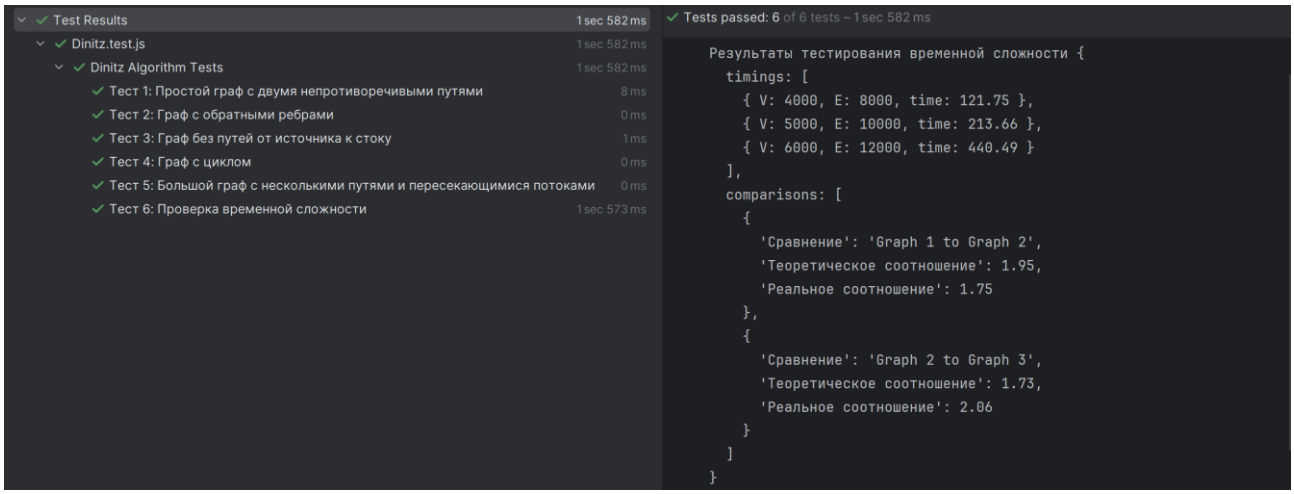

Рисунок 8 – Результаты тестирования

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 цикламициклами, а также была подтверждена временная сложность алгоритма $O(V^2 \cdot E)$.

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.