Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский технологический университет «МИСИС»

Комбинаторика и теория графов

Задача построения максимального потока в сети. Алгоритм Эдмондса-Карпа

Авад Фатхи Абделмонем Мохамед Ахмед

https://github.com/FATHEY12352/alg_cm3

Содержание

Введение

В теории графов задача максимального потока является одной из центральных. Она имеет множество приложений, начиная от оптимизации транспортных сетей и заканчивая распределением ресурсов в компьютерных системах. Алгоритм Эдмондса-Карпа, модификация алгоритма Форда-Фалкерсона, позволяет эффективно решать эту задачу, используя стратегию поиска в ширину для нахождения увеличивающих путей.

1. Формальная постановка задачи

Рассматривается сеть G=(V,E)G = (V, E)G=(V,E), где:

- VVV — множество вершин графа,
- EEE — множество рёбер,
- c(u,v)c(u, v)c(u,v) — пропускная способность ребра (u,v)(u, v)(u,v).

Задача: Найти максимальный поток f(s,t)f(s, t)f(s,t) из истока sss в сток ttt, удовлетворяющий следующим условиям:

1. Поток через любое ребро f(u,v)f(u, v)f(u,v) не превышает его пропускную способность: 0≤f(u,v)≤c(u,v)0 \leq f(u, v) \leq c(u, v)0≤f(u,v)≤c(u,v), (u,v)∈E(u, v) \in E(u,v)∈E.
2. Закон сохранения потока: сумма входящих потоков в вершину равна сумме выходящих потоков, за исключением истока sss и стока ttt: ∑u∈Vf(u,v)=∑w∈Vf(v,w), ∀v∈V∖{s,t}.\sum_{u \in V} f(u, v) = \sum_{w \in V} f(v, w), \; \forall v \in V \setminus \{s, t\}.u∈V∑f(u,v)=w∈V∑f(v,w),∀v∈V∖{s,t}.

Цель: Максимизировать поток ∑u∈Vf(s,u)\sum_{u \in V} f(s, u)∑u∈Vf(s,u).

2. Теоретическое описание алгоритма

Алгоритм Эдмондса-Карпа

Алгоритм решает задачу максимального потока, используя поиск в ширину (BFS) для нахождения увеличивающего пути в остаточной сети.

Этапы работы алгоритма:

1. **Инициализация**:
    - Поток через все рёбра устанавливается равным нулю: $f(u,v)=0, (u,v)\in E$
    - Создаётся остаточная сеть G_f, где остаточная пропускная способность определяется как: c_f(u, v) = c(u, v) - f(u, v).
2. **Поиск увеличивающего пути**:
    - Используется BFS для нахождения пути из s в **t** в **Gf**.
    - Если путь найден, определяется минимальная остаточная пропускная способность вдоль пути (бутылочное горлышко): $\delta=\min (u,v)\in Pcf(u,v)$.
3. **Обновление потоков**:
    - Для каждого ребра в пути увеличивается поток: $f(u,v)=f(u,v)+\delta$.
    - Для обратных рёбер уменьшается поток: $f(v,u)=f(v,u)-\delta$
4. **Повтор**:
    - Алгоритм повторяет шаги 2–3, пока существует увеличивающий путь.

Временная сложность:

Алгоритм имеет временную сложность O(VE^2), где V — число вершин, E — число рёбер. Это объясняется тем, что каждый BFS выполняется за O(E), а количество итераций ограничено O(E).

3. Сравнительный анализ алгоритма

Алгоритм	**Временная сложность**	**Подход**
Форда-Фалкерсона	Экспоненциальная	DFS
Эдмондса-Карпа	O(VE^2)	BFS
Алгоритм Диница	O(V^2.E)	Уровневые графы
Алгоритм Каргера	O(V^3)	Случайное сокращение

Эдмондс-Карп является более предсказуемым и легко реализуемым, чем классический Форда-Фалкерсона, хотя уступает в производительности алгоритму Диница.

4. Перечень используемых инструментов

- **Язык программирования**: C#.
- **Среда разработки**: Microsoft Visual Studio.
- **Дополнительные библиотеки**: System.Collections.Generic для работы с очередями.

5. Описание реализации

Основные компоненты реализации:

1. **Класс `EdmondsKarp`**:
   - Содержит матрицы пропускной способности и потока.
   - Реализует BFS для поиска пути.
   - Обновляет потоки.
2. **Методы**:
   - `AddEdge(int from, int to, int capacity)` — добавление ребра.
   - `MaxFlow(int source, int sink)` — вычисление максимального потока.

Код на C#

```
using System;
using System.Collections.Generic;

class EdmondsKarp
{
    private int[,] capacity;
    private int[,] flow;
    private int[] parent;
    private int nodeCount;

    public EdmondsKarp(int nodeCount)
    {
        this.nodeCount = nodeCount;
        capacity = new int[nodeCount, nodeCount];
        flow = new int[nodeCount, nodeCount];
        parent = new int[nodeCount];
    }

    public void AddEdge(int from, int to, int capacityValue)
    {
        capacity[from, to] = capacityValue;
    }

    private bool BFS(int source, int sink)
    {
        Array.Fill(parent, -1);
        var visited = new bool[nodeCount];
        var queue = new Queue<int>();
        queue.Enqueue(source);
        visited[source] = true;

        while (queue.Count > 0)
        {
            int current = queue.Dequeue();
            for (int next = 0; next < nodeCount; next++)
            {
                if (!visited[next] && capacity[current, next] - flow[current, next] > 0)
                {
                    parent[next] = current;
                    visited[next] = true;
                    queue.Enqueue(next);
                    if (next == sink) return true;
                }
            }
        }
        return false;
    }
```

Эдмондс-Карп | Program | Main()

```csharp

    public int MaxFlow(int source, int sink)
    {
        int maxFlow = 0;
        while (BFS(source, sink))
        {
            int pathFlow = int.MaxValue;
            for (int v = sink; v != source; v = parent[v])
            {
                int u = parent[v];
                pathFlow = Math.Min(pathFlow, capacity[u, v] - flow[u, v]);
            }
            for (int v = sink; v != source; v = parent[v])
            {
                int u = parent[v];
                flow[u, v] += pathFlow;
                flow[v, u] -= pathFlow;
            }
            maxFlow += pathFlow;
        }
        return maxFlow;
    }
}

class Program
{
    static void Main()
    {
        var ek = new EdmondsKarp(6);
        ek.AddEdge(0, 1, 16);
        ek.AddEdge(0, 2, 13);
        ek.AddEdge(1, 2, 10);
        ek.AddEdge(1, 3, 12);
        ek.AddEdge(2, 4, 14);
        ek.AddEdge(3, 2, 9);
        ek.AddEdge(3, 5, 20);
        ek.AddEdge(4, 3, 7);
        ek.AddEdge(4, 5, 4);
        Console.WriteLine("Maximum Flow: " + ek.MaxFlow(0, 5));
    }
}

```

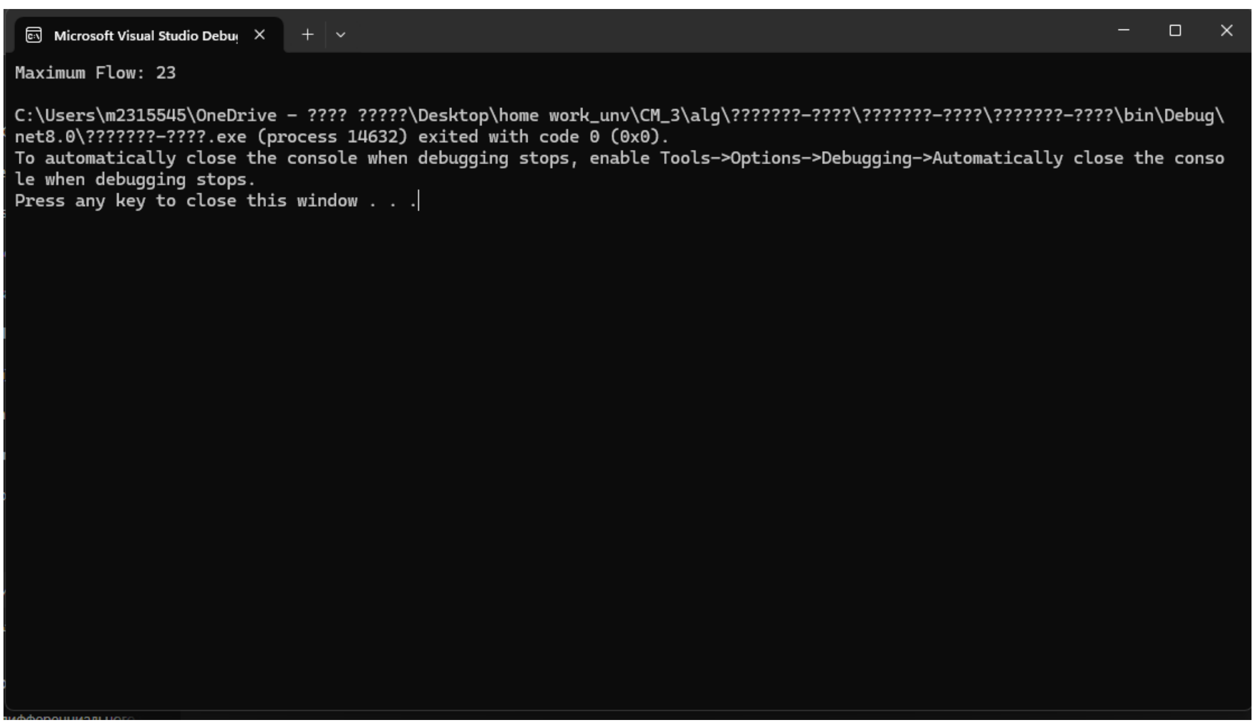
51 % | No issues found | Ln: 75 | Ch: 6 | SPC | CRLF

6. Тестирование и результаты

Для проверки работы алгоритма использовался следующий граф с шестью вершинами:

- Ребро из 0 в 1 с пропускной способностью 16.
- Ребро из 0 в 2 с пропускной способностью 13.
- Ребро из 1 в 2 с пропускной способностью 10.
- Ребро из 1 в 3 с пропускной способностью 12.
- Ребро из 2 в 4 с пропускной способностью 14.
- Ребро из 3 в 2 с пропускной способностью 9.
- Ребро из 3 в 5 с пропускной способностью 20.
- Ребро из 4 в 3 с пропускной способностью 7.
- Ребро из 4 в 5 с пропускной способностью 4.

Результат работы программы:
Максимальный поток из вершины 0 в вершину 5 составляет **23**.



Журнал выполнения алгоритма

1. Первый увеличивающий путь: 0→1→3→5 бутылочным горлышком δ=12.
2. Второй увеличивающий путь: 0→2→4→ бутылочным горлышком δ=4.
3. Третий увеличивающий путь: 0→2→1→3→5 бутылочным горлышком δ=7.

Сумма потоков: 12+4+7=23.

7. Заключение

Алгоритм Эдмондса-Карпа продемонстрировал эффективность при решении задачи максимального потока в сети. Преимущества алгоритма:

1. Простота реализации благодаря использованию поиска в ширину.
2. Предсказуемая временная сложность O(VE^2), что позволяет решать задачи среднего масштаба.

Основной недостаток алгоритма — сравнительно высокая временная сложность по сравнению с алгоритмом Диница, особенно для графов с большим числом рёбер. Тем не менее, алгоритм остаётся популярным благодаря своей универсальности и ясности.

8. Список литературы

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. "Алгоритмы: Построение и анализ".
2. Edmonds J., Karp R.M. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems", 1972.
3. Официальная документация C#: docs.microsoft.com.