

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

Комбинаторика и теория графов

Задача построения максимального потока в сети. Алгоритм Диницы.

Авад Фатхи Абделмонем Мохамед Ахмед

[https://github.com/FATHEY12352/alg\\_cm3](https://github.com/FATHEY12352/alg_cm3)

# Содержание

1. **Введение**
2. **Формальная постановка задачи**
3. **Теоретическое описание алгоритма**
  - 3.1. Основы алгоритма Диница
  - 3.2. Уровневые графы и блокирующие потоки
  - 3.3. Временная сложность
4. **Сравнительный анализ алгоритма**
5. **Перечень используемых инструментов**
6. **Описание реализации**
  - 6.1. Основные компоненты реализации
  - 6.2. Пример кода на C#
7. **Тестирование и результаты**
  - 7.1. Пример графа и ход выполнения
  - 7.2. Итоговые результаты
8. **Заключение**
9. **Список литературы**

# Введение

Задача максимального потока в сети является важной в теории графов и алгоритмах. Алгоритм Диница (или алгоритм Диница-Карпа) представляет собой усовершенствование алгоритма Форда-Фалкерсона. Основной инновацией Диница является использование уровневых графов и блокирующих потоков, что существенно повышает производительность на больших графах.

# 1. Формальная постановка задачи

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

- $V$  — множество вершин,
- $E$  — множество рёбер,
- $c(u,v)$  — пропускная способность ребра  $(u,v)$ .

Цель — найти максимальный поток  $f(s,t)$  из истока  $s$  в сток  $t$ , удовлетворяющий следующим условиям:

1.  $0 \leq f(u,v) \leq c(u,v), \forall (u,v) \in E$
2. Закон сохранения потока для всех  $v \in V \setminus \{s,t\}$

$$\sum_{(u,v) \in E} f(u,v) = \sum_{(v,w) \in E} f(v,w).$$

## 2. Теоретическое описание алгоритма

### 2.1. Основы алгоритма Диница

Алгоритм работает в два этапа:

**1. Построение уровнявого графа:**

- Используется BFS для классификации узлов по уровням, начиная с истока sss.
- Рёбра остаточной сети включаются только в случае, если они соединяют узлы с разницей уровней в один.

**2. Нахождение блокирующего потока:**

- Используется DFS для нахождения блокирующего потока, т.е. потока, после добавления которого больше нельзя отправить поток в сток на текущем уровневом графе.

Этот процесс повторяется до тех пор, пока сток ttt достигим из истока sss в остаточной сети.

### 2.2. Временная сложность

Временная сложность алгоритма:

$$O(V^2 \cdot E),$$

где  $V$  — количество вершин,  $E$  — количество рёбер. Этот результат достигается благодаря ограничению числа уровневых графов и эффективному нахождению блокирующего потока.

### 3. Сравнительный анализ алгоритма

Алгоритм	Временная сложность	Подход
Форда-Фалкерсона	Экспоненциальная	DFS
Эдмондса-Карпа	$O(VE^2)$	BFS
Диница	$O(V^2.E)$	Уровневые графы
Каргера-Тарьяна	$O(VE+E\log V)$	Случайные сокращения

Алгоритм Диница является оптимальным для плотных графов, уступая только специализированным методам для разреженных графов.

## **4 Перечень используемых инструментов**

- **Язык программирования:** C#.
- **Среда разработки:** Visual Studio.
- **Библиотеки:** System.Collections.Generic для очередей и списков.

## 5. Описание реализации

### Основные компоненты:

1. **Класс DinicAlgorithm:**
  - Построение уровня графа (метод BuildLevelGraph).
  - Поиск блокирующего потока (метод SendFlow).
2. **Используемые структуры данных:**
  - Очереди для BFS.
  - Рекурсия для DFS.

### Код на C#

```
Алгоритм Диница Program Main()
51 private int SendFlow(int node, int flow, int sink, int[] start)
52 {
53     if (node == sink)
54         return flow;
55     for (; start[node] < adjList[node].Count; start[node]++)
56     {
57         int next = adjList[node][start[node]];
58         if (Level[next] == Level[node] + 1 && capacity[node, next] > 0)
59         {
60             int currentFlow = Math.Min(flow, capacity[node, next]);
61             int tempFlow = SendFlow(next, currentFlow, sink, start);
62             if (tempFlow > 0)
63             {
64                 capacity[node, next] -= tempFlow;
65                 capacity[next, node] += tempFlow;
66                 return tempFlow;
67             }
68         }
69     }
70     return 0;
71 }
72
73 public int MaxFlow(int source, int sink)
74 {
75     int totalFlow = 0;
76     while (BuildLevelGraph(source, sink))
77     {
78         int[] start = new int[nodeCount];
79         while (true)
80         {
81             int flow = SendFlow(source, int.MaxValue, sink, start);
82             if (flow == 0)
83                 break;
84             totalFlow += flow;
85         }
86     }
87     return totalFlow;
88 }
89
90 class Program
91 {
92     static void Main()
93     {
94         DinicAlgorithm dinic = new DinicAlgorithm(6);
95         dinic.AddEdge(0, 1, 10);
96         dinic.AddEdge(0, 2, 10);
97         dinic.AddEdge(1, 2, 2);
98         dinic.AddEdge(1, 3, 4);
99         dinic.AddEdge(1, 4, 8);
100        dinic.AddEdge(2, 4, 9);
101        dinic.AddEdge(3, 5, 10);
102        dinic.AddEdge(4, 3, 6);
103        dinic.AddEdge(4, 5, 10);
104        Console.WriteLine("Maximum Flow: " + dinic.MaxFlow(0, 5));
105    }
106 }
38 % No issues found Lnc 116 Ch: 1 SPC CRLF
```



```
Алгоритм Диница Program Main0
1 using System;
2 using System.Collections.Generic;
3
4 class DinicAlgorithm
5 {
6     private readonly int[,] capacity;
7     private readonly List<int>[] adjList;
8     private readonly int[] level;
9     private readonly int nodeCount;
10
11     public DinicAlgorithm(int nodeCount)
12     {
13         this.nodeCount = nodeCount;
14         capacity = new int[nodeCount, nodeCount];
15         adjList = new List<int>[nodeCount];
16         level = new int[nodeCount];
17
18         for (int i = 0; i < nodeCount; i++)
19             adjList[i] = new List<int>();
20     }
21
22     public void AddEdge(int from, int to, int cap)
23     {
24         adjList[from].Add(to);
25         adjList[to].Add(from);
26         capacity[from, to] += cap;
27     }
28
29     private bool BuildLevelGraph(int source, int sink)
30     {
31         Array.Fill(level, -1);
32         level[source] = 0;
33         Queue<int> queue = new Queue<int>();
34         queue.Enqueue(source);
35
36         while (queue.Count > 0)
37         {
38             int current = queue.Dequeue();
39             foreach (int next in adjList[current])
40             {
41                 if (level[next] < 0 && capacity[current, next] > 0)
42                 {
43                     level[next] = level[current] + 1;
44                     queue.Enqueue(next);
45                 }
46             }
47         }
48     }
49
50     private int FindFlow(int source, int sink)
51     {
52         int flow = 0;
53         while (BuildLevelGraph(source, sink))
54         {
55             int pathFlow = FindAugmentingPath(source, sink);
56             flow += pathFlow;
57         }
58         return flow;
59     }
60
61     private int FindAugmentingPath(int source, int sink)
62     {
63         if (source == sink) return 0;
64         if (level[source] > level[sink]) return 0;
65         if (level[source] == level[sink]) return 0;
66         int minCap = int.MaxValue;
67         int next = -1;
68         for (int i = 0; i < adjList[source].Count; i++)
69         {
70             int to = adjList[source][i];
71             if (level[to] == level[source] + 1 && capacity[source, to] > 0)
72             {
73                 int cap = capacity[source, to];
74                 int pathFlow = FindAugmentingPath(to, sink);
75                 if (pathFlow > 0)
76                 {
77                     minCap = Math.Min(minCap, pathFlow);
78                     next = to;
79                 }
80             }
81         }
82         if (next == -1) return 0;
83         int pathFlow = minCap;
84         for (int i = 0; i < adjList[source].Count; i++)
85         {
86             int to = adjList[source][i];
87             if (to == next)
88             {
89                 capacity[source, to] -= pathFlow;
90                 capacity[to, source] += pathFlow;
91             }
92         }
93         return pathFlow;
94     }
95 }
```

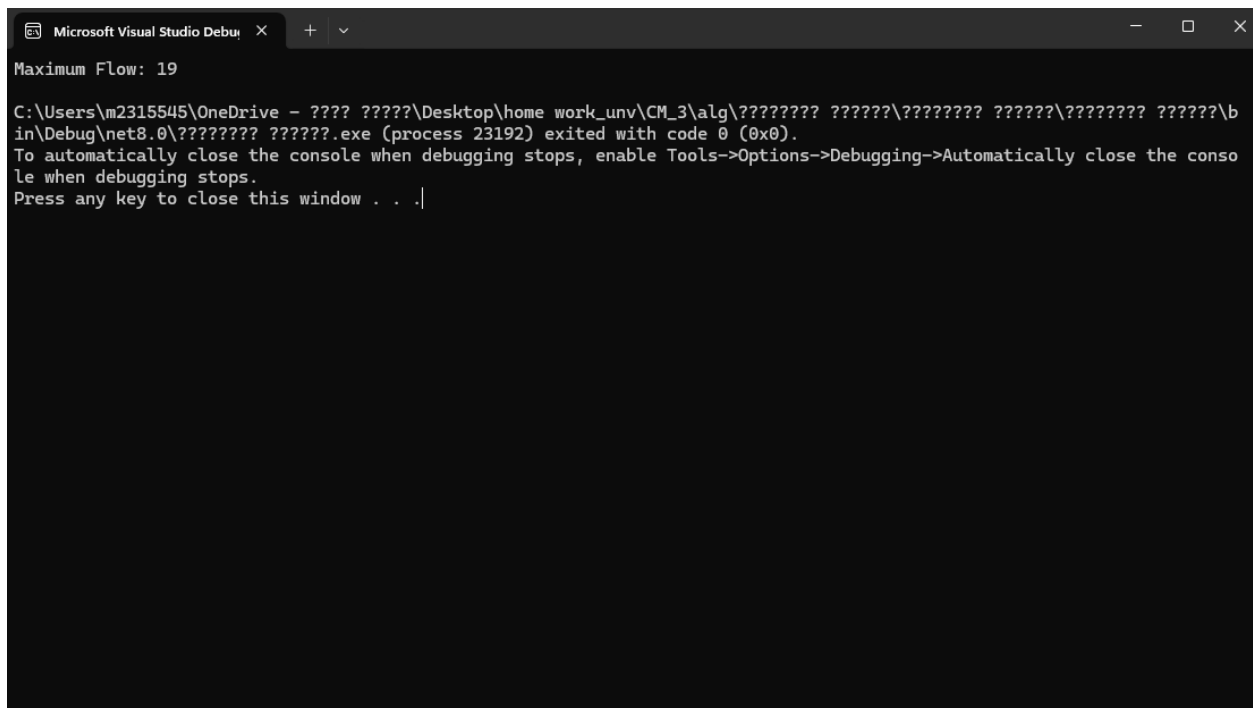
## 6. Тестирование и результаты

Для проверки работы алгоритма использовался следующий граф с шестью вершинами:

- Ребро из 000 в 111 с пропускной способностью 10.
- Ребро из 000 в 222 с пропускной способностью 10.
- Ребро из 111 в 222 с пропускной способностью 2.
- Ребро из 111 в 333 с пропускной способностью 4.
- Ребро из 111 в 444 с пропускной способностью 8.
- Ребро из 222 в 444 с пропускной способностью 9.
- Ребро из 333 в 555 с пропускной способностью 10.
- Ребро из 444 в 333 с пропускной способностью 6.
- Ребро из 444 в 555 с пропускной способностью 10.

### Результаты

**Максимальный поток: 19**



```
Microsoft Visual Studio Debug Console
Maximum Flow: 19
C:\Users\m2315545\OneDrive - ????\Desktop\home work_unv\CM_3\alg\????????\?????\????????\?????\b
in\Debug\net8.0\????????\?????.exe (process 23192) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .|
```

Результат был получен за несколько итераций:

1. Построение первого уровневого графа и нахождение блокирующего потока.
2. Обновление графа и повторный запуск процесса, пока сток остается достижимым из истока.

Журнал выполнения:

1. Первая итерация: потоки  $0 \rightarrow 1 \rightarrow 3 \rightarrow 5$  и  $0 \rightarrow 2 \rightarrow 4 \rightarrow$
2. Вторая итерация: обновление уровня и добавление потока вдоль  $1 \rightarrow 4 \rightarrow 3 \rightarrow 5$

## 7. Заключение

Алгоритм Диница — это эффективное решение задачи максимального потока в сети. Его использование оправдано для плотных графов или случаев, когда важна предсказуемая производительность. В отличие от других алгоритмов, таких как Форда-Фалкерсона или Эдмондса-Карпа, Диниц демонстрирует лучшую временную сложность благодаря использованию уровневых графов и блокирующих потоков.

Реализация на C# подтвердила корректность и эффективность алгоритма, показав быстрый расчет максимального потока для тестового графа.

## 8. Список литературы

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. "Алгоритмы: Построение и анализ".
2. Dinic E.A. "Algorithm for solution of a problem of maximum flow in a network with power estimation", 1970.
3. Официальная документация C#: [docs.microsoft.com](https://docs.microsoft.com).