Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Дискретный анализ»

Студент: П.А. Земсков

Преподаватель: Н.К. Макаров

Группа: М8О-301Б

Дата: _____

Оценка: _____

Подпись: _____

Москва, 2025

Содержание

1 Лабораторная работа №8

Задача

Необходимо найти величину максимального потока в графе при помощи алгоритма Форда-Фалкерсона.
Вариант алгоритма: Поиск максимального потока алгоритмом Форда-Фалкерсона

2 Описание

Задан взвешенный ориентированный граф, состоящий из n вершин и m ребер. Вершины пронумерованы целыми числами от 1 до n. Необходимо найти величину максимального потока в графе при помощи алгоритма Форда-Фалкерсона. Для достижения приемлемой производительности в алгоритме рекомендуется использовать поиск в ширину, а не в глубину. Истоком является вершина с номером 1, стоком – вершина с номером n. Вес ребра равен его пропускной способности. Граф не содержит петель и кратных ребер.

3 Исходный код

Общая идея: Алгоритм находит максимальный поток путем многократного поиска кратчайшего увеличивающего пути от истока к стоку с помощью BFS, затем пропускает поток по найденному пути, обновляя остаточные пропускные способности ребер (уменьшая прямые и увеличивая обратные), и повторяет процесс до тех пор, пока не останется увеличивающих путей, что гарантирует оптимальность за $O(V*E^2)$ благодаря выбору кратчайших путей, где V - количество вершин, E - количетсво ребер.

Листинг 1: Исходный код

```
#include <bits/stdc++.h>

using namespace std;

struct Edge {
    int to;
    long long capacity;
    int rev;
};

bool bfs(int source, int sink, vector<vector<Edge>>& graph, vector<
   int>& parent, vector<int>& parentEdge) {
    fill(parent.begin(), parent.end(), -1);
    queue<int> q;
    q.push(source);
    parent[source] = -2;

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (int i = 0; i < graph[u].size(); i++) {
            Edge& e = graph[u][i];
            if (parent[e.to] == -1 && e.capacity > 0) {
                parent[e.to] = u;
                parentEdge[e.to] = i;
                if (e.to == sink) {
                    return true;
                }
                q.push(e.to);
            }
        }
    }
    return false;
}

long long findMaxFlow(vector<vector<Edge>>& graph, int n, int
   source, int sink) {
    long long maxFlow = 0;
    vector<int> parent(n + 1);
    vector<int> parentEdge(n + 1);

    while (bfs(source, sink, graph, parent, parentEdge)) {
        long long pathFlow = LLONG_MAX;

        for (int v = sink; v != source; v = parent[v]) {
            int u = parent[v];
            int edgeIdx = parentEdge[v];
            pathFlow = min(pathFlow, graph[u][edgeIdx].capacity);
        }
```

```
49
50          for (int v = sink; v != source; v = parent[v]) {
51              int u = parent[v];
52              int edgeIdx = parentEdge[v];
53              graph[u][edgeIdx].capacity -= pathFlow;
54              graph[v][graph[u][edgeIdx].rev].capacity += pathFlow;
55          }
56
57          maxFlow += pathFlow;
58      }
59
60      return maxFlow;
61  }
62
63  int main() {
64      ios_base::sync_with_stdio(false);
65      cin.tie(NULL);
66
67      int n, m;
68      cin >> n >> m;
69
70      vector<vector<Edge>> graph(n + 1);
71
72      for (int i = 0; i < m; i++) {
73          int u, v;
74          long long capacity;
75          cin >> u >> v >> capacity;
76          graph[u].push_back({v, capacity, (int)graph[v].size()});
77          graph[v].push_back({u, 0, (int)graph[u].size() - 1});
78      }
79
80      cout << findMaxFlow(graph, n, 1, n) << endl;
81
82      return 0;
83  }
```

4 Консоль

Пример компиляции и демонстрация работы программы:

```
C:\Users\jocke\Documents\diskran\lab9> g++ --std=c++20 main.cpp -o main
C:\Users\jocke\Documents\diskran\lab9> ./main
5 6
1 2 4
1 3 3
1 4 1
2 5 3
3 5 3
4 5 10
7
```

5 Тест производительности

Алгоритм протестирован на графах разного размера в пределах ограничений задачи ($n < 2000$, $m < 10000$). На графах с 100 вершинами и 500 ребрами, 500 вершинами и 2000 ребрами, 1000 вершинами и 5000 ребрами, а также на максимальном размере (2000 вершин, 10000 ребер) время выполнения не превышает 1 миллисекунды. Это подтверждает эффективность алгоритма с теоретической сложностью $O(V \cdot E^2)$: использование BFS для поиска кратчайших увеличивающих путей обеспечивает полиномиальное время работы и практическую применимость для графов указанного размера.

```
C:\Users\jocke\Documents\diskran\lab9> g++ --std=c++20 benchmark.cpp -o benchmark
C:\Users\jocke\Documents\diskran\lab9> ./benchmark
Граф: 100 вершин, 500 ребер
Максимальный поток: 36503617
Время выполнения: 0 мс

Граф: 500 вершин, 2000 ребер
Максимальный поток: 2038560580
Время выполнения: 0 мс

Граф: 1000 вершин, 5000 ребер
Максимальный поток: 1552714143
Время выполнения: 0 мс

Граф: 2000 вершин, 10000 ребер
Максимальный поток: 216826489
Время выполнения: 0 мс
```

Результат:

Искомая величина потока.

6 Выводы

В ходе лабораторной работы реализован алгоритм Форд-Факельсона для поиска максимального потока в ориентированном графе. Алгоритм использует поиск в ширину для нахождения кратчайших увеличивающих путей, что обеспечивает полиномиальную сложность $O(V \cdot E^2)$ вместо экспоненциальной. Реализация корректно обрабатывает графы с количеством вершин до 2000 и рёбер до 10000, успешно проходит тестовые примеры и демонстрирует высокую производительность — время выполнения на максимальных размерах графов не превышает 1 миллисекунды.

7 Список литературы

1. Кормен Т. Х., Лейзерсон Ч. И., Ривест Р. Л., Штайн К. *Алгоритмы: построение и анализ. 3-е изд.* – М.: Издательский дом «Вильямс», 2010.

2. Knuth D. E. *The Art of Computer Programming. Vol. 3: Sorting and Searching. 2nd ed.* – Addison-Wesley, 1998.

3. *Алгоритм Форда-Фалкерсона - ИТМО.* `https://neerc.ifmo.ru/wiki`.