

**Московский авиационный институт (национальный
исследовательский университет)**

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа № 9 по курсу «Дискретный анализ»

Студент: Бачурин П.Д.
Преподаватель: Макаров Н.К
Даты:
Оценка:
Подпись:

Москва, 2024

Задача

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.

Формат ввода

В первой строке заданы $1 \leq n \leq 2000$ и $1 \leq m \leq 100000$. В следующих m строках записаны ребра. Каждая строка содержит три числа – номера вершин, соединенных ребром, и вес данного ребра. Вес ребра – целое число от 0 до 10^9 .

Формат вывода

Необходимо вывести одно число – искомую величину максимального потока. Если пути из истока в сток не существует, данная величина равна нулю.

Алгоритм

1. Инициализация данных:

- Создаем граф, представленный в виде списка смежности, и матрицу пропускных способностей `graph`, где `graph[from][to]` хранит пропускную способность ребра из вершины `from` в вершину `to`. Также создаем `flow`, где `flow[from][to]` хранить текущий поток, проходящий по ребру из вершины `from` в вершину `to`.

2. Алгоритм поиска пути (BFS):

- Используем BFS для поиска пути из источника в сток (по остаточной сети).
- Если путь найден, сохраняем его с помощью массива `prev`, где `prev[to]` указывает вершину, из которой пришли в `to`.

3. Обновление потока:

- После нахождения пути определяем минимальную пропускную способность на пути (т.е. минимальное значение `graph[from][to] — flow[from][to]` на ребрах пути).
- Обновляем остаточную сеть, уменьшая пропускные способности прямых ребер и увеличивая обратные пропускные способности (`flow[from][to] += min_flow, flow[to][from] -= min_flow`).

4. Повторение:

- Повторяем шаги 2–3 до тех пор, пока существует путь из источника в сток в остаточной сети.

5. Вывод результата:

- Суммируем потоки, добавленные на каждой итерации, чтобы получить максимальный поток.

Исходный код

```
#include <vector>
#include <algorithm>
#include <iostream>
#include <queue>
#include <cstdint>
#include <limits>

const int noPrevVertex = -1;

using Graph = std::vector<std::vector<int64_t>>>;

Graph make_graph(uint vertices) {
    return std::vector<std::vector<int64_t>>>(vertices,
std::vector<int64_t>(vertices));
}

void bfs(Graph const& graph, Graph const& flow, uint from,
std::vector<uint> & prev) {
    std::queue<uint> q;
    q.push(from);
    prev[from] = from;

    while (not q.empty()) {
        from = q.front();
        q.pop();

        for (size_t idx = 0; idx < graph[from].size(); ++idx) {
            uint to = idx;
            if (prev[to] == noPrevVertex and flow[from][to] <
graph[from][to]) {
                prev[to] = from;
                q.push(to);
            }
        }
    }
}

int main() {
    uint n, m;
    std::cin >> n >> m;
    auto graph = make_graph(n);
    auto flow = make_graph(n);

    for (size_t idx = 0; idx < m; ++idx) {
        uint from, to;
        int64_t weight;
        std::cin >> from >> to >> weight;
        graph[from-1][to-1] += weight;
    }

    uint start = 1, finish = n;
    uint64_t ans = 0;
```

```

while (true) {
    std::vector<uint> prev(n, noPrevVertex);
    bfs(graph, flow, start-1, prev);
    if (prev[finish - 1] == noPrevVertex) {
        break;
    }

    std::vector<uint> path;
    uint last = finish - 1;
    while (prev[last] != last) {
        path.push_back(last);
        last = prev[last];
    }
    path.push_back(last);
    std::reverse(path.begin(), path.end());

    int64_t min_flow = std::numeric_limits<int64_t>::max();
    for (size_t idx = 1; idx < path.size(); ++idx) {
        uint from = path[idx - 1], to = path[idx];
        min_flow = std::min(graph[from][to] - flow[from][to],
min_flow);
    }

    for (size_t i = 1; i < path.size(); ++i) {
        uint from = path[i - 1], to = path[i];
        flow[from][to] += min_flow;
        flow[to][from] -= min_flow;
    }
    ans += min_flow;
}
std::cout << ans << std::endl;
}

```

Тесты

1.

ВВОД:

2 1

1210

ВЫВОД:

10

2.

ВВОД:

4 5

1210

1 3 5

2315

2410

3410

ВЫВОД:

15

3.

ВВОД:

3 2

1 2 5

2 3 0

ВЫВОД:

0

4.

ВВОД:

4 6

1 2 3

1 3 7

2 3 2

2 4 5

3 4 6

1410

ВЫВОД:

19

5.

ВВОД:

6 8

1210

1310

2425

3415

4510

4610

5610

2 5 5

ВЫВОД:

20

6.

ВВОД:

5 6

1 2 4

1 3 3

1 4 1

2 5 3

3 5 3

4510

ВЫВОД:

7

Вывод

В ходе лабораторной работы я реализовал алгоритм Форда-Фалкерсона для нахождения максимального потока в ориентированном графе. Программа использует матрицу пропускных способностей для хранения остаточной сети и список смежности для эффективного обхода графа с использованием BFS. При тестировании алгоритм корректно нашёл максимальный поток на графах с большим числом вершин и рёбер, включая значения пропускных способностей до 10^9 , что подтверждает правильность и производительность решения.