МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

АЛГОРИТМ ФОРДА-ФАЛКЕРСОНА

ОТЧЁТ

студента 2 курса 211 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Пицика Харитона Николаевича

Проверил

______________ ______________ ______________

Саратов 2025

СОДЕРЖАНИЕ

1 Постановка проблематики, терминология

В теории графов, транспортная сеть (или же flow network) представляет собой ориентированный граф $G(V, E)$, в котором каждое ребро имеет так называемую пропускную способность, которая задаётся функцией $c : V \times V \to \mathbb{R}_+$, причем для любого $e \notin E$ значение функции равно 0. Потоком называется функция $f : V \times V \to \mathbb{R}_+$

Одна из самых распространённых задач в графах — задача о максимальном потоке. В теории оптимизации и в теории графов, задача о максимальном потоке заключается в нахождении такого потока в транспортной сети, что сумма потоков из истока максимальная. В 1955 году Лестер Форд и Делберт Фалкерсон впервые построили алгоритм, специально предназначенный для этой задачи. Их алгоритм получил название алгоритм Форда-Фалкерсона.

2 Идея алгоритма

Идея алгоритма заключается в следующем. Изначально величине потока присваивается 0: $f(u, v) = 0 \; \forall (u, v) \in V$. Затем величина потока итеративно увеличивается посредством поиска увеличивающегося пути. Рассмотрим алгоритм пошагово:

— В начальный момент времени поток, который мы хотим провести через нашу сеть, должен быть равен нулю. Остаточная сеть совпадает с исходной сетью;
— Находим любой путь из истока в сток в остаточной сети. Если путь не находим, утверждается, что поток является максимальным;
— Пускаем через найденный путь поток равный минимальному весу ребра, которое входит в множество рёбер найденного пути;
— Из веса рёбер на этом пути высчитываем размер потока, который мы пустили;
— А к весу обратных рёбер (будем считать, что они существуют в остаточной сети и равны 0) прибавляем размер потока. Другими словами, на предыдущем шаге мы отправили некоторое количество потока из текущей вершины в следующую, а теперь при желании можем вернуть это же количество потока обратно в текущую;
— Возвращаемся обратно к нахождению пути в остаточной сети после модификации.

Алгоритм Форда-Фалкерсона реализован, используя обход в глубину (DFS).

3 Реализация, код программы

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

const int INF = 1e9; // большое число, для обозначения бесконечности

// Структура графа
struct Graph {
    int n; // число вершин
    vector<vector<int>> capacity; // матрица пропускных способностей
    vector<vector<int>> adj; // список смежности

    Graph(int n) : n(n) {
        capacity.assign(n, vector<int>(n, 0));
        adj.resize(n);
    }

    // Добавление ребра
    void addEdge(int u, int v, int cap) {
        capacity[u][v] = cap;
        adj[u].push_back(v);
        adj[v].push_back(u); // добавляем обратную связь для поиска путей
    }
};

// Функция поиска пути увеличения с помощью DFS
int dfs(int u, int t, vector<bool>& visited, vector<int>& parent, const
↪   vector<vector<int>>& capacity, const vector<vector<int>>& adj) {
    visited[u] = true;
    if (u == t) return INF; // достигли стока, возвращаем бесконечность (максимальное
    ↪   возможное увеличение)

    for (int v : adj[u]) {
        if (!visited[v] && capacity[u][v] > 0) {
            parent[v] = u; // запоминаем предка для восстановления пути
            int flow = dfs(v, t, visited, parent, capacity, adj);
            if (flow > 0) {
```

```
            return min(flow, capacity[u][v]); // минимальная пропускная способность по
             ↪  пути
        }
    }
}
return 0; // путь не найден
}

// Основная функция для поиска максимального потока
int fordFulkerson(Graph& g, int s, int t) {
    int maxFlow = 0;
    vector<vector<int>> capacity = g.capacity; // копируем матрицу пропускных
     ↪  способностей

    while (true) {
        vector<bool> visited(g.n, false);
        vector<int> parent(g.n, -1);

        // ищем путь увеличения с помощью DFS
        int flow = dfs(s, t, visited, parent, capacity, g.adj);
        if (flow == 0) break; // больше путей нет — алгоритм завершен

        maxFlow += flow;

        // обновляем остаточные пропускные способности по найденному пути
        int v = t;
        while (v != s) {
            int u = parent[v];
            capacity[u][v] -= flow;   // уменьшаем пропускную способность по прямому рёбру
            capacity[v][u] += flow;   // увеличиваем по обратному рёбру (обратный поток)
            v = u;
        }
    }

    return maxFlow;
}
```

4 Анализ сложности

В случае, если значения пропускных способностей являются иррациональными числами, алгоритм может зациклиться в бесконечность. В случае с целыми числами, сложность алгоритма составляет $O(|E|f)$, где E — множество рёбер, f — максимальный поток в графе. Это обусловлено тем, что каждый увеличивающий путь можно найти за $O(E)$ и увеличивает поток как минимум на единицу. В худшем случае алгоритм будет работать $O(|V + E|f)$

5 Достоинства и недостатки алгоритма, особенности использования

К плюсам данного алгоритма можно отнести:

- Теоретически доказанное гарантированное нахождение максимального пути;
- Распространённость. Алгоритм можно применять везде, где в сетевом потоке необходимо найти максимальный путь, но при условии, что пропускные способности неотрицательные;
- Гибкость. Можно изменить алгоритм поиска пути увеличения. Например, заменить обход в глубину на обход в ширину, что даёт другую сложность $O(VE^2)$

Как уже описывалось выше, одним из главных недостатков данного алгоритма является невозможность работы с вещественными иррациональными числами, поскольку существует вероятность, что в таком случае алгоритм не сможет завершить свою работу. Также к минусам можно отнести сложность алгоритма в худшем случае, так как в худшем случае происходит слишком много итераций. Обычно, для оптимизации вычислений, вместо обхода в глубину используют обход в ширину, что является реализацией другого алгоритма, Эдмонса-Карпа.