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:	Д. С. Ляшун
Преподаватель:	А. А. Кухтичев
Группа:	М8О-207Б
Дата:	
Оценка:	
Подпись:	

Москва, 2021

Лабораторная работа №9

Задача: Разработать программу на языке C или C++, реализующую указанный алгоритм согласно заданию:

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.

1 Описание

Требуется написать алгоритм Форда-Фалкерсона, его реализация с помощью поиска в ширину также известна как алгоритм Эдмондса-Карпа [1]. Основная идея заключается в том, чтобы на каждом шаге работы алгоритма в текущей остаточной сети (начальная остаточная сеть – входной граф) искать с помощью поиска в ширину увеличивающий путь от вершины 1 до n такой, что в него входят рёбра с ненулевыми остаточными пропускными способностями. После нахождения данного пути необходимо изменить остаточную сеть – уменьшить значения остаточных пропускных способностей у входящих в увеличивающий путь ребёр на min – минимальное значение остаточной пропускной способности среди этих рёбер в пути, а также увеличить значение остаточных пропускных способностей у обратных рёбер на это же min. Таким образом, мы нашли один из путей насыщения к вершине n и текущую на данном шаге работы алгоритма величину максимального потока можно увеличить на min. Полученная новая остаточная сеть используется далее при поиске следующего увеличивающего пути. Работа алгоритма прекращается в случае, если в текущей остаточной сети нет никакого увеличивающего пути.

Оценим время работы полученного алгоритма. Как было доказано в [1], общее число увлечений потока в алгоритме Эдмондса-Карпа, составляет $O(mn)$, при этом в ходе каждого увеличения происходит поиск увеличивающего пути за время работы поиска в ширину – $O(m + n)$. Тогда можно сделать вывод, что алгоритм Эдмондса-Карпа работает за время $O(mn(m + n))$ или $O(m^2n)$ если $m >> n$.

2 Исходный код

Ниже приведён исходный код программы, в котором производится чтение входного графа, выполнение алгоритма Эдмондса-Карпа и вывод ответа – значения максимального потока:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>
const int MAX_EDGE_COST = 1000000000;
long long EdmondsKarp(const std::vector<std::vector<int> > & edges, std::vector<std::vector<int> > &cF) {
    long long answer = 0;
    bool ok = true;
    int n = edges.size()-1;
    while (ok) {
        ok = false;
        std::queue<int> q;
        std::vector<int> visited(n+1);
        q.push(1);
        while (!q.empty()) {
            int t = q.front();
            q.pop();
            if (t == n) {
                int mini = MAX_EDGE_COST;
                int pos = n;
                while (pos != 1) {
                    mini = std::min(mini, cF[visited[pos]][pos]);
                    pos = visited[pos];
                }
                pos = n;
                while (pos != 1) {
                    cF[visited[pos]][pos] -= mini;
                    cF[pos][visited[pos]] += mini;
                    pos = visited[pos];
                }
                answer += mini;
                ok = true;
                break;
            }
            for (int i = 0; i < edges[t].size(); ++i) {
                if (cF[t][edges[t][i]] != 0 && visited[edges[t][i]] == 0) {
                    visited[edges[t][i]] = t;
                    q.push(edges[t][i]);
                }
            }
        }
    }
```

```
    return answer;
}
int main() {
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<int> > edges(n+1);
    std::vector<std::vector<int> > cF(n+1, std::vector<int>(n+1));
    for (int i = 0; i < m; ++i) {
        int v, u, cost;
        std::cin >> v >> u >> cost;
        edges[v].push_back(u);
        edges[u].push_back(v);
        cF[v][u] = cost;
    }
    long long answer = EdmondsKarp(edges, cF);
    std::cout << answer << std::endl;
    return 0;
}
```

3 Консоль

```
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab9$ g++ main.cpp -o main
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab9$ cat test.txt
5 6
1 2 4
1 3 3
1 4 1
2 5 3
3 5 3
4 5 10
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab9$ ./main <test.txt
7
```

4 Тест производительности

Тест производительности представляет из себя сравнение времени работы алгоритма Эдмондса-Карпа с помощью поиска в ширину и алгоритма Форда-Фалкерсона с использованием поиска в глубину соответственно. Все тесты производились на полных графах с числом рёбер $m = n^2$, веса рёбер определялись случайно и имели значения от 1 до 10^9.

Число вершин полного графа n	Время работы алгоритма Эдмондса-Карпа	Время работы алгоритма Форда-Фалкерсона на dfs
5	53 мкс.	38 мкс.
10	220 мкс.	316 мкс.
20	1806 мкс.	10082 мкс.
40	30940 мкс.	246109 мкс.
80	123838 мкс.	4256952 мкс.

Как видно, время работы алгоритма Эдмондса-Карпа заметно быстрее алгоритма Форда-Фалкерсона с использованием поиска в глубину. Это связано с тем, что в ходе выбора увеличивающего пути алгоритм Форд-Фалкерсона может выбрать неоптимальный, который не является кратчайшим и поэтому может содержать использовавшиеся ранее обратные ребра с низкой стоимостью, отчего насыщение потока будет происходить очень медленно. Как сказано в [1], выбор кратчайшего увеличивающего пути всегда будет являться асимптотически оптимальнее.

5 Выводы

Выполнив девятую лабораторную работу по курсу «Дискретный анализ», я познакомился с алгоритмом Форда-Фалкерсона для поиска максимального потока в графе, в частности с одним из методов его реализации посредством поиска в ширину – алгоритмом Эдмондса-Карпа. Основную трудность для меня составило изменение значений остаточных пропускных способностей после нахождения увеличивающего пути – я не изменял значение для обратных рёбер в пути.

Стоит отметить, что данный алгоритм имеет множество различных практических применений. Например, с его помощью можно определить количество поставок вещества по какой-то системе трубопроводов или же использовать для решения задач в сфере логистики о размерах поставок продукта потребителю.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с.