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:	А. А. Почечура
Преподаватель:	А. А. Кухтичев
Группа:	М8О-306Б
Дата:	
Оценка:	
Подпись:	

Москва, 2023

Лабораторная работа №9

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.

1 Описание

Требуется реализовать улучшенный алгоритм Форда-Фалкерсона — алгоритм Эдмондса-Карпа. Идея заключается в том, что мы используем для поиска максимального потока обход в ширину вместо обхода в глубину. Это позволяет нам всегда найти путь от истока к стоку, состоящий из самого малого количества вершин, который может быть в текущем графе. После нахождения пути определяется минимальная пропускная способность среди всех ребёр и из всех их пропускной способности вычитается найденное значение. При вычитании из ребра пропускной способности в одну сторону, она увеличивается у этого же ребра в обратную сторону ровно на это же значение. Таким образом, по рёбрам может прокладываться путь в обе стороны, но суммарная пропускная способность у ребра в обе стороны всегда постоянна и задаётся в начале. Алгоритм прокладывает пути до тех пор, пока новые пути от истока к стоку нельзя будет найти.

2 Исходный код

Граф хранится как список смежности, так как в таком представлении можно быстро определить, есть ли путь из одной вершины в другую. В ячейках матрицы хранится пропускная способность ребра, ведущего из вершины i в вершину j. В функции Bfs происходит поиск пути, состоящий из минимального количества рёбер. Вектор $path$ для вершины i единственным образом определяет, из какой вершины в неё попали (номер вершины, откуда пришли в вершину i хранится в значении i-той ячейки), а в массиве $path\_weight$ содержится пропускная способность ребра, по которому попали в вершину i. Обход в ширину продолжается до тех пор, пока все возможные вершины не будут посещены, либо пока не достигнут сток. После этого, если сток всё таки достигнут, восстанавливается весь путь с помошью массива $path$, определяется минимальная пропускная способность пути с помощью массива $pass\_weight$ и вычитается из ребёр, по которым был проложен путь (рёбрам, направленным в обратную сторону, пропускная способность прибавляется). Далее возвращаем величину потока, который нам удалось провести из истока в сток и прибавляем к ответу. Когда функция вернёт значение -1, алгоритм заканчивает работу и выводится значение переменной ans.

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

int inf = 1e9 + 1;

int Bfs(vector<vector<int>>& edges) {
  queue<int> q;
  vector<int> path(edges[0].size(), -1); //  i-  ,        i
  vector<int> path_weight(edges[0].size()); //  i-  ,        i
  q.push(0);
  int vertex;
  while (!q.empty()) {
    vertex = q.front();
    if (vertex == edges[0].size() - 1) { //    .     (         )
      break;
    }
    q.pop();
    for (int i = 0;i < edges[vertex].size();i++) { //          vertex
      if (edges[vertex][i] != 0) { //    ,
        if (path[i] == -1) { //       ,
          q.push(i);
          path[i] = vertex;
          path_weight[i] = edges[vertex][i];
        }
```

```
      }
    }
  }
  if (q.empty()) { //     (    )
    return -1;
  }
  int tmp = inf;
  while (vertex != 0) { //
    tmp = min(tmp, path_weight[vertex]);
    vertex = path[vertex];
  }
  vertex = edges[0].size() - 1;
  while (vertex != 0) { //
    edges[path[vertex]][vertex] = edges[path[vertex]][vertex] - tmp;
    edges[vertex][path[vertex]] = edges[vertex][path[vertex]] + tmp;
    vertex = path[vertex];
  }
  return tmp;
}

int main() {
  int n, m;
  cin >> n >> m;
  vector<vector<int>> edges(n, vector<int>(n));
  int a, b;
  for (int i = 0;i < m;i++) {
    cin >> a >> b;
    a--;
    b--;
    cin >> edges[a][b]; //
  }
  long long ans = 0;
  long long tmp = Bfs(edges); //
  while (tmp != -1) {
    ans = ans + tmp;
    tmp = Bfs(edges); //  ,
  }
  cout << ans;
}
```

3 Консоль

```
root@DESKTOP-5HM2HTK:~# cat <test
5 6
1 2 4
1 3 3
```

```
1 4 1
2 5 3
3 5 3
4 5 10
root@DESKTOP-5HM2HTK:~# g++ lab9.cpp
root@DESKTOP-5HM2HTK:~# ./a.out <test
7
root@DESKTOP-5HM2HTK:~#
```

4 Тест производительности

```
root@DESKTOP-5HM2HTK:~# g++ generator.cpp
root@DESKTOP-5HM2HTK:~# ./a.out tests
root@DESKTOP-5HM2HTK:~# g++ benchmark.cpp
root@DESKTOP-5HM2HTK:~# ./a.out <tests/1.t
N is: 10000
M is: 100000
Edmonds-Karp algorithm time: 467ms
Ford-Fulkerson algoritm time: 1398ms
root@DESKTOP-5HM2HTK:~#
```

Как видно, алгоритм Эдмондса-Карпа работает примерно в три раза быстрее, чем алгоритм Форда-Фалкерсона. Это объясняется тем, то первым алгоритм реже использует уже пройденные рёбра, поэтому все остальные рёбра заполняются значительно быстрее, а значит поиск максимального потока занимает меньшее количество времени.

5 Выводы

Выполнив девятую лабораторную работу по курсу «Дискретный анализ», я попрактиковался в работе с графами и научился писать алгоритм Эдмондса-Карпа. Данный алгоритм было интересно осваивать и полезно знать для решения будущих задач.

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. (ISBN 5-8459-0857-4 (рус.)).*

[2] *Алгоритм Эдмондса — Карпа - Википедия.*
URL: `https://ru.wikipedia.org/wiki/Алгоритм_Эдмондса_-_Карпа` (дата обращения: 13.11.2022).