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. М. Чистяков
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б-20
Дата:
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №9

Задача:

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.

Формат входных данных: В первой строке заданы $1 \leq n \leq 2000$ и $1 \leq m \leq 10000$. 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 результата: Необходимо вывести одно число – искомую величину максимального потока. Если пути из истока в сток не существует, данная величина равна нулю.

1 Описание

Алгоритм Форда-Фалкерсона состоит из поиска в ширину и изменение весов графа в соответствии с минимальным весом ребра в найденном пути.

Алгоритм крутит поиск в ширину пока не сможет найти путь из истока в сток графа (1,n). Далее, в двудольном графе меняет веса ребер. Каждый раз находя новый путь в графе, минимльный вес является довабочным потоком в графе и оно прибавляется к общему ответу, когда алгоритм не смодет найти путь, алгоритм прекращает работу и выходит.

Скорость работы алгоритма оценивается как $O(f * |E|)$, где f - кол-во найденных путей в графе, а $|E|$ - кол-во или мощность множества вершин графа. Это можно объяснить так: мы ищем путь в графе, далее для каждой вершины в пути, длина которого может быть максимум $|E|$, меняем веса за $O(1)$.

2 Исходный код

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

using namespace std;

bool BFS(vector<unordered_map<int,int>>& graph, int start, int end, vector<int>&
    parents) {
  parents.clear();
  parents.resize(graph.size(), -1);
  vector<bool> visited(graph.size());
  queue<int> q;
  q.push(start);
  visited[start] = true;

  while (!q.empty()) {
    int cur_vertex = q.front();
    q.pop();
    for (auto edge : graph[cur_vertex]) {
      int new_vertex = edge.first;
      int new_capacity = edge.second;
      if (!visited[new_vertex] && new_capacity) {
        visited[new_vertex] = true;
        q.push(new_vertex);
        parents[new_vertex] = cur_vertex;
        if (new_vertex == end) {
          return true;
        }
      }
    }
  }
  return false;
}

long long MaxFlow(vector<unordered_map<int,int>>& graph, int source, int sink) {
  long long resFlow = 0;
  vector<int> parents;
  while (BFS(graph, source, sink, parents)) {
    int flow = 1000000001;
    for (int i = sink; i != source; i = parents[i]) {
      if (i == -1) {
        throw runtime_error("Can't find a parent for not source node.");
      }
      if (graph[parents[i]][i] < flow) {
        flow = graph[parents[i]][i];
      }
```

```
47      }
48      resFlow += flow;
49
50      for (int i = sink; i != source; i = parents[i]) {
51        graph[parents[i]][i] -= flow;
52        graph[i][parents[i]] += flow;
53      }
54    }
55    return resFlow;
56  }
57
58  int main() {
59    ios_base::sync_with_stdio(false);
60    cin.tie(nullptr);
61    int n, m;
62    cin >> n >> m;
63    vector<unordered_map<int,int>> graph(n);
64    for (int i = 0; i < m; ++i) {
65      int from, to, capacity;
66      cin >> from >> to >> capacity;
67      --from;
68      --to;
69      graph[from][to] = capacity;
70    }
71    cout << MaxFlow(graph, 0, n - 1) << "\n";
72  }
```

3 Консоль

```
den@vbox:~/Документы/DA/lab9$ ./a.out
5 6
1 2 4
1 3 3
1 4 1
2 5 3
3 5 3
4 5 10
7
```

4 Тест производительности

В программе я использовал поиск в ширину для нахождения пути. Попробуем использовать поиск в глубину и сравним результаты. Для сравнения я подготовил четыре теста с полными графами, где количество вершин равно 10, 20 и 30.

```
den@vbox:~/Документы/DA/lab9$ ./bfs <test10
368639003
Time: 0.0002965 s
den@vbox:~/Документы/DA/lab9$ ./dfs <test10
368639003
Time: 0.0038175 s
den@vbox:~/Документы/DA/lab9$ ./bfs <test20
808017301
Time: 0.0005277 s
den@vbox:~/Документы/DA/lab9$ ./dfs <test20
808017301
Time: 0.0012016 s
den@vbox:~/Документы/DA/lab9$ ./bfs <test30
1252856464
Time: 0.0008176 s
den@vbox:~/Документы/DA/lab9$ ./dfs <test30
1252856464
Time: 1.36699 s
```

Иногда поиск в глубину значительно проигрывает по времени поиску в ширину. Дело в том, что поиск в ширину всегда находит кратчайший путь, на каждом шаге приближаясь к конечной точке, а поиск в глубину обходит вершины графа в случайном порядке и находит не самый оптимальный путь, совершая много лишних операций. Однако асимптотическая сложность этих алгоритмов одинакова: $O(m + n)$, где m - количество ребер графа, n - количество вершин графа.

5 Выводы

Выполнив девятую лабораторную работу по курсу «Дискретный анализ», я изучил способы представления и обработки графов на C++. Граф можно представить списком ребер, матрицей смежности или списком смежности. Изучен алгоритм Форда-Фалкерсона нахождения максимального потока. Этот алгоритм часто используется в оптимизационных задачах на практике, хотя его идея чрезмерно проста.

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.))

(дата обращения; 19.07.20).

[2] Дэн Гасфилд. *Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология.*, Пер. с англ. И. В. Романовского. — СПб.: Невский Диалект; БХВ-Петербург, 2003. — 654 с: ил.)

(дата обращения; 19.07.20).

[3] *Greedy_ algorithm* URL: https://en.wikipedia.org/wiki/Greedy_algorithm

(дата обращения; 19.07.20).