Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №9 по курсу «Дискретный анализ»

Студентка: А. А. Довженко
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2018

Лабораторная работа №9

Задача: Разработать программу на языке C или C++, реализующую указанный алгоритм. Формат входных и выходных данных описан в варианте задания. Первый тест в проверяющей системе совпадает с примером.

Вариант 7: Поиск максимального потока алгоритмом Форда-Фалкерсона. 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 рёбер.

1 Описание

Задача состоит в том, чтобы отправить как можно больше груза из истока в сток, не превышая пропускной способности рёбер. Необходимо найти величину потока каждого ребра, которые удовлетворяют набору линейных ограничений и максимизируют величину потока. Задача о максимальном потоке сводится к задаче линейного программирования.

Идея алгоритма заключается в следующем. Изначально величине потока присваивается значение $0 : f(u, v) = 0$ для всех u, v из V. Затем величина потока итеративно увеличивается посредством поиска увеличивающего пути (путь от источника s к стоку t, вдоль которого можно послать ненулевой поток). Шаг алгоритма состоит в выборе пути из истока в сток в остаточной сети и увеличении потока вдоль него насколько возможно (на этом пути нас ограничевает ребро с наименьшей пропускной способностью в остаточной сети). Процесс повторяется, пока можно найти увеличивающий путь.

2 Исходный код

Изначально обнуляем все потоки так, что остаточная сеть совпадает с исходной сетью. В остаточной сети находим кратчайший путь из источника в сток. Если такого пути нет, останавливаемся. Пускаем через пройденный путь максимально возможный поток: ищем в нем ребро с минимальной пропускной способностью, для каждого ребра на найденном пути увеличиваем потока на это число, а в противоположном ему уменьшаем. Модифицируем осточную сеть. Для всех рёбер на найденном пути, а также для противоположных им рёбер, вычисляем новую пропускную спообность. Если она стала ненулевой, добавляем ребро к остаточной сети, а если обнулилась, стираем его.

```
1  #include <iostream>
2  #include <limits>
3  #include <vector>
4  #include <queue>
5  #include <algorithm>
6
7  bool BFS(std::vector<std::vector<int> > &graph, int start, int end, std::vector<int> &
       parent)
8  {
9      std::vector<bool> visited(graph.size());
10     std::queue <int> q;
11     q.push(start);
12     visited[start] = true;
13     parent[start] = -1;
14
15     while (!q.empty()) {
16         int u = q.front();
17         q.pop();
18         for (int v = 1; v < graph.size(); ++v) {
19             if (!visited[v] && graph[u][v]) {
20                 q.push(v);
21                 parent[v] = u;
22                 visited[v] = true;
23             }
24         }
25     }
26     return visited[end];
27 }
28
29 long long FordFulkerson(std::vector<std::vector<int> > &graph, int start, int end)
30 {
31     int u, v;
32     long long maxFlow = 0;
33     std::vector<std::vector<int> > resGraph = graph;
34     std::vector<int> parent(graph.size());
35
```

```
36      while (BFS(resGraph, start, end, parent)) {
37          int flow = std::numeric_limits<int>::max();
38          for (v = end; v != start; v = parent[v]) {
39              u = parent[v];
40              flow = std::min(flow, resGraph[u][v]);
41          }
42
43          for (v = end; v != start; v = parent[v]) {
44              u = parent[v];
45              resGraph[u][v] -= flow;
46              resGraph[v][u] += flow;
47          }
48
49          maxFlow += flow;
50      }
51
52      return maxFlow;
53  }
54
55  int main(void)
56  {
57      int n = 0, m = 0, from = 0, to = 0;
58      std::cin >> n >> m;
59      std::vector<std::vector<int> > graph;
60      graph.resize(n + 1);
61      for (int i = 0; i < n + 1; ++i) {
62          graph[i].resize(n + 1);
63      }
64      for (int i = 0; i < m; ++i) {
65          std::cin >> from >> to;
66          std::cin >> graph[from][to];
67      }
68
69      std::cout << FordFulkerson(graph, 1, n) << std::endl;
70      return 0;
71  }
```

3 Консоль

```
karma@karma:~/mai_study/DA/lab9$ make
g++ -std=c++11 -O3 -pedantic -lm -o da9 main.cpp
karma@karma:~/mai_study/DA/lab9$ cat test
5 6
1 2 4
1 3 3
1 4 1
2 5 3
3 5 3
4 5 10
karma@karma:~/mai_study/DA/lab9$ cat test | ./da9
7
```

4 Тест производительности

Тест производительности представляет собой сравнение с тем же алгоритмом Форда-Фалкерсона, в котором используется поиск в глубину вместо поиска в ширину.

Генерирую 4 файла, содержащие полные графы из 10 (input10), 20 (input20), 30 (input30) вершин.

```
karma@karma:~/mai_study/DA/lab9$ cat input10 | ./da9
Ford-Fulkerson with BFS time: 1.6e-05 sec.
karma@karma:~/mai_study/DA/lab9$ cat input10 | ./dfs
Ford-Fulkerson with DFS time: 0.000208 sec.

karma@karma:~/mai_study/DA/lab9$ cat input20 | ./da9
Ford-Fulkerson with BFS time: 0.000415 sec.
karma@karma:~/mai_study/DA/lab9$ cat input20 | ./dfs
Ford-Fulkerson with DFS time: 2.79952 sec.

karma@karma:~/mai_study/DA/lab9$ cat input30 | ./da9
Ford-Fulkerson with BFS time: 0.000686 sec.
karma@karma:~/mai_study/DA/lab9$ cat input30 | ./dfs
Ford-Fulkerson with DFS time: 159.879 sec.
```

Как видим, разница между алгоритмами ощутима. Дело в том, что при поиске в глубину пути из источника в сток выбираются неоптимально, и количество итераций по рёбрам становится слишком большим, в то время как поиск в ширину ищет путь с наименьшим числом рёбер. Сложность алгоритма Форда-Фалкерсона $O(E|f^*|)$, где $|f^*|$ – максимальный поток в графе. Сложность алгоритма, использующего поиск в ширину, – $O(VE^2)$. Существует еще более быстрый алгоритм решения этой задачи, использующий проталкивание предпотока, который решает её за время $O(V^3)$

5 Выводы

Задача о максимальном потоке является классической и имеет множество применений. Она является обобщением большого класса задач, сводящихся к ней. Выбор алгоритма зачастую зависит от имеющихся сведений о структуре транспортной сети. Поэтому необходимо грамотно выбирать стратегию, наиболее эффективную в данной практической ситуации. Для обширного класса комбинаторных алгоритмов эти задачи представляют собой водораздел – между необходимостью изучать эффективные алгоритмы решения конкретных задач и изучением обобщенных моделей решения. Чем выше уровень общности модели, тем больше задач она охватывает, а это повышает полезность алгоритма, который может решить любую задачу, сводимую к этой модели. Разработка такого алгоритма может оказаться сложным, а то и невозможным делом. Даже при отсутствии алгоритма с гарантированно приемлемой производительностью обычно существует алгоритм, который хорошо работает для специальных классов задач, которые нас интересуют. Отдельные аналитические выкладки зачастую дают очень грубые оценки производительности, но нередко имеются убедительные эмпирические исследования. Все же не следует увлекаться слишком обобщенными моделями, которые тратят кучу времени на решение тех задач, для которых эффективно работают более специализированные модели. Интересно, что никакой алгоритм с использованием расширения путей не может иметь линейную производительность в худшем случае.

Список литературы

[1] Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. *Алгоритмы: построение и анализ, 3-е изд.* — Издательский дом «Вильямс», 2013. Перевод с английского: ООО «И. Д. Вильямс». — 1328 с. (ISBN 978-5-8459-1794-2 (рус.))

[2] С. Дасгупта, Х. Пападимитриу, У. Вазирани. *Алгоритмы* — Издательство «МЦНМО», 2014. Перевод с английского: А. С. Куликова под редакцией А. Шеня. — 319 с. (ISBN 978-5-4439-0236-4 (рус.))