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: Л. Я. Вельтман
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2019

Лабораторная работа №9

Задача: Разработать программу на языке C или C++, реализующую указанныи алгоритм. Формат входных и выходных данных описан в варианте задания.

Вариант 5: Поиск кратчайшего пути между парои вершин алгоритмом Беллмана-Форда

Задан взвешенныи ориентированный граф, состоящий из n вершин и m рёбер. Вершины пронумерованы целыми числами от 1 до n. Необходимо найти длину кратчайшего пути из вершины с номером start в вершину с номером finish при помощи алгоритма Беллмана-Форда. Длина пути равна сумме весов рёбер на этом пути. Обратите внимание, что в данном варианте веса рёбер могут быть отрицательными, поскольку алгоритм умеет с ними работать. Граф не содержит петель, кратных рёбер и циклов отрицательного веса.

Входные данные: В первои строке заданы $1 \leqslant n \leqslant 10^5, 1 \leqslant m \leqslant 3 * 10^5$, $1 \leqslant start \leqslant n$ и $1 \leqslant finish \leqslant n$. В следующих m строках записаны рёбра. Каждая строка содержит три числа – номера вершин, соединенных ребром, и вес данного ребра. Вес ребра – целое число от -10^9 до 10^9.

Выходные данные: Необходимо вывести одно число – длину кратчаишего пути между указанными вершинами. Если пути между указанными вершинами не существует, следует вывести строку "No solution" (без кавычек).

1 Описание

Пусть дан ориентированный взвешенный граф G с n вершинами и m рёбрами, и указана некоторая вершина v. Требуется найти длины кратчайших путей от вершины v до всех остальных вершин.

Этот алгоритм применим также и к графам, содержащим рёбра отрицательного веса. Впрочем, если граф содержит отрицательный цикл, то, понятно, кратчайшего пути до некоторых вершин может не существовать (по причине того, что вес кратчайшего пути должен быть равен минус бесконечности). Но в условии задания сказано, что отрицательных циклов не предусмотрено, поэтому проверять наличие цикла отрицательного веса не потребуется.

2 Исходный код

Заведём массив расстояний dist[0 . . . n-1], который после отработки алгоритма будет содержать ответ на задачу. В начале работы мы заполняем его следующим образом: dist[v] = 0, а все остальные элементы dist[] равны бесконечности ∞.

Сам алгоритм Форда-Беллмана представляет из себя несколько фаз. На каждой фазе просматриваются все рёбра графа, и алгоритм пытается произвести релаксацию (relax, ослабление) вдоль каждого ребра (v,u) стоимости weight. Релаксация вдоль ребра — это попытка улучшить значение dist[u] значением dist[v] + weight. Фактически это значит, что мы пытаемся улучшить ответ для вершины u, пользуясь ребром (v,u) и текущим ответом для вершины v.

Для недостижимых вершин расстояние dist[] останется равным бесконечности ∞.

Проверка "if (dist[edge[j].from] $<$ INF)" нужна, только если граф содержит рёбра отрицательного веса: без такой проверки бы происходили релаксации из вершин, до которых пути ещё не нашли, и появлялись бы некорректные расстояния вида $\infty - 1$, $\infty - 2$, и т.д.

```
1  #include <iostream>
2  #include <vector>
3  #include <limits>
4
5  const long INF = std::numeric_limits<long>::max();
6
7  struct Edge {
8      unsigned long from, to;
9      long cost;
10 };
11
12 void BellmanFord(const std::vector<Edge>& , const unsigned long&, const unsigned long
       &, const unsigned long&, const unsigned long&);
13
14
15 int main(int argc, const char * argv[]) {
16     std::ios_base::sync_with_stdio(false);
17
18     unsigned long n, m, start, finish, u, v;
19     long weight;
20     std::cin >> n >> m >> start >> finish;
21     --start; --finish;
22     std::vector<Edge> edge(m);
23
24     for (unsigned long i = 0; i < m; ++i) {
25         std::cin >> v >> u >> weight;
26         --v; --u;
27         edge[i].from = v;
28         edge[i].to = u;
```

```
29          edge[i].cost = weight;
30      }
31
32      BellmanFord(edge, n, m, start, finish);
33
34      return 0;
35  }
36
37  void BellmanFord(const std::vector<Edge>& edge, const unsigned long& n, const unsigned
        long& m,const unsigned long& start,const unsigned long& finish) {
38      std::vector<long> dist(n, INF);
39      dist[start] = 0;
40      for (unsigned long i = 0; i < n; ++i) {
41          bool relaxation = false;
42          for (unsigned long j = 0; j < m; ++j) {
43              if (dist[edge[j].from] < INF) {
44                  if (dist[edge[j].to] > dist[edge[j].from] + edge[j].cost) {
45                      dist[edge[j].to] = dist[edge[j].from] + edge[j].cost;
46                      relaxation = true;
47                  }
48              }
49          }
50          if (!relaxation) break;
51      }
52
53      if (dist[finish] == INF) std::cout << "No solution" << std::endl;
54      else std::cout << dist[finish] << std::endl;
55  }
```

3 Консоль

```
MacBook-Pro-Lina:desktop linuxoid$ g++ -std=c++11 da9.cpp -o da9
MacBook-Pro-Lina:desktop linuxoid$ cat test
5 6 1 5
1 2 3
1 3 6
3 2 10
4 2 7
3 4 4
4 5 2
MacBook-Pro-Lina:desktop linuxoid$ ./da9 <test
12
```

4 Тест производительности

Тест производительности представляет собой сравнение с тем же алгоритмом Беллмана-Форда, но граф в нем задается матрицей весов. На вход каждому из алгоритмов подаются файлы с 10(test10), 20(test20) и 30(test30) вершинами.

```
MacBook-Pro-Lina:desktop linuxoid$ ./da9 <test10
14
BellmanFord O(n*m)
3.5e-05 sec
MacBook-Pro-Lina:desktop linuxoid$ ./bfn <test10
14
BellmanFord O(n^3)
5.1e-05 sec
MacBook-Pro-Lina:desktop linuxoid$ ./da9 <test20
27
BellmanFord O(n*m)
3.7e-05 sec
MacBook-Pro-Lina:desktop linuxoid$ ./bfn <test20
27
BellmanFord O(n^3)
0.000147 sec
MacBook-Pro-Lina:desktop linuxoid$ ./da9 <test30
18
BellmanFord O(n*m)
4.3e-05 sec
MacBook-Pro-Lina:desktop linuxoid$ ./bfn <test30
18
BellmanFord O(n^3)
0.000121 sec
```

Сразу заметна разница между алгоритмами. Дело в том, что граф, заданный матрицей весов имеет асимптотику $O(n^3)$, где n - количество вершин. В реализации алгоритма лежит 3 вложенных цикла, каждый из которых просматривает n вершин. В моей реализации граф задан списком ребер, работает алгоритм за $O(n * m)$.

5 Выводы

Хотелось бы отметить, что хоть алгоритм работает за O(|V|*|E|), где V - количество вершин, E - количество ребер, проигрывая асимптотике алгоритма Дейкстры (|E|+|V|*ln(|V|), однако он обладает важной особенностью, а именно его применимость к графам с произвольными, в том числе отрицательными, весами. Также интересен тот факт, что алгоритм Беллмана-Форда обладает значительным ресурсом параллелизма. Во-первых, поиск кратчайших путей от различных вершин может производиться независимо для каждой из вершин. Во-вторых, поиск кратчайших путей от фиксированной вершины u также может выполняться параллельно: инициализация начальных путей требует |V| параллельных операций, релаксация каждого ребра требует O(|E|) параллельных операций. Таким образом, при наличии O(|E|) процессоров алгоритм завершит работу максимум за |V| шагов.

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.))