Московский авиационный институт
(национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №9 по курсу «Дискретный анализ»

Студент: С. Ю. Свиридов
Группа: М8О-306Б-22
Дата:
Оценка:
Подпись:

Москва 2024

Графы

Задача: С. 5 Поиск кратчайшего пути между парой вершин алгоритмом Беллмана-Форда

Задан взвешенный неориентированный граф, состоящий из n вершин и m ребер. 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 ребер на этом пути. Обратите внимание, что в данном варианте веса ребер могут быть отрицательными, поскольку алгоритм умеет с ними работать. Граф не содержит петель, кратных ребер и циклов отрицательного веса.

Формат ввода

В первой строке заданы $1 \leq n \leq 3{*}10^5$ и $1 \leq m \leq 10^5$, $1 \leq start \leq n$, $1 \leq finish \leq n$. В следующих m строках записаны ребра. Каждая строка содержит три числа - номера вершин, соединенных ребром, и вес данного ребра. Вес ребра - целое число от -10^9 до 10^9.

Формат вывода

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

1 Описание

Алгоритм Беллмана-Форда позволяет искать кратчайший путь между парой вершин, при этом умеет работать с графами, содержащими отрицательные ребра. Алгоритм работает на принципе релаксации ребер, когда мы обновляем наименьшнее расстояние от заданной вершины до текущей по принципу: для ребра (u, v) с весом w - если переход через u дает более короткий путь к v от исходного узла (т.е., расстояние [v] $>$ расстояние [u] + w), мы обновляем расстояние [v] как расстояние [u] + w. Такой процесс повторяется V - 1 раз для всех ребер, где V - количество вершин в графе. Почему v - 1? потому что кратчайший путь между двумя вершинами может иметь не более (V – 1) ребер. Невозможно иметь простой путь с более чем (V – 1) ребрами (в противном случае он образовывал бы цикл). Следовательно, повторение процесса релаксации (V – 1) раз гарантирует, что все возможные пути между источником и любым другим узлом были пройдены. В противном случае, можно говорить о наличие в графе отрицательного цикла (это цикл в графике, сумма весов ребер которого отрицательна). При его наличии смысл поиска кратчайшего пути просто теряется, так как можно каждый раз сокращать его, проходясь по этому отрицательному циклу.

Реализация

Для описания графа опишем простую структуру $Edge$, у которой будут поля $to, from, wt$, которые будут значить номер вершины отправления, номер вершины прибытия и вес ребра; этого будет достаточно для решения задачи. В цикле while(true) будет для каждого ребра (u, v, w) проверять, можно ли уменьшить путь до вершины v, прокладывая его через вершину u. Цикл завершится, когда обновление расстояния не поступит. Далее просто из массива dist, который хранит в себе кратчайшие расстояния до всех вершин от исходной start, возвращаем значение по индексу $finish - 1$.

2 Исходный код

Приложен исходный код программы.

```
1 #include <vector>
2 #include <iostream>
3 #include <tuple>
4 #include <limits>
5
6 struct Edge {
7   int from;
8   int to;
```

```
  long long int wt;
};

long long int BellmanFord(int n, int start, int finish, std::vector<Edge> allEdges) {
  std::vector<long long int> dist(n, 1e18);
  dist[start] = 0;

  while (true) {
    bool flag = false;
    for (int i = 0; i < allEdges.size(); i++) {
      int u = allEdges[i].from - 1;
      int v = allEdges[i].to - 1;
      long long int w = allEdges[i].wt;

      if (dist[u] < 1e18 && dist[u] + w < dist[v]) {
        dist[v] = dist[u] + w;
        flag = true;
      }
    }

    if (!flag) {
      break;
    }
  }

  return dist[finish];
}

int main() {
  std::cin.tie(nullptr);
  std::ios_base::sync_with_stdio(false);

  int n = 0, m = 0, start = 0, finish = 0;

  std::cin >> n >> m >> start >> finish;
  std::vector<Edge> allEdges;

  int from = 0, to = 0;
  long long int len = 0;
  allEdges.resize(m);

  for (int i = 0; i < m; i++) {
    std::cin >> allEdges[i].from >> allEdges[i].to >> allEdges[i].wt;
  }

  long long int dist = BellmanFord(n, start - 1, finish - 1, allEdges);

  if (dist == 1e18) {
    std::cout << "No solution" << std::endl;
```

```
58    } else {
59      std::cout << dist << std::endl;
60    }
61  }
```

3 Консоль

```
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba9$ g++ main.cpp
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba9$ ./a.out
5 6 1 5
1 2 2
1 3 0
3 2 -1
2 4 1
3 4 4
4 5 5
5

potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba9$ ./a.out
3 2 1 3
1 2 5
2 3 6
11

potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba9$ ./a.out
3 1 1 3
2 3 14
No solution
```

Сложность алгоритма - $O(m * n)$

4 Тест производительности

В тестах проверим работу алгоритма с разным количеством вершин и ребер.

```
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba9$ ./a.out
300 300 1 300
No solution
Result time: 5 ms

potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba9$ g++ bench.cpp
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba9$ ./a.out
3000 3000 1 3000
No solution
Result time: 108 ms

potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba9$ g++ bench.cpp
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba9$ ./a.out
100000 100000 1 100000
No solution
Result time: 5418 ms

potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba9$ ./a.out
300000 300000 1 300000
No solution
Result time: 27197 ms
```

Как видно из результатов теста, время выполнения растет очень быстро, быстрее, чем растет линейная функция. Это доказывает высокую сложность $O(m * n)$ нашего алгоритма.

5 Выводы

В ходе выполнения данной лабораторной работы я познакомился с такой структрой данных как граф. Так же попробовал поискать кратчайшее расстояние между парой вершин методом Беллмана-Форда. В принципе, метод прост в понимании, однако в то же время позволяет работать с отрицательными весами ребер, в отличие от, например, алгоритма Дейкстры, что делает его универсальнее. К сожалению, метод имеет довольно высокую сложность, но зато достаточно прост в реализации и понятен в понимании.

Список литературы

[1] *Алгоритм Беллмана-Форда*
URL: https://translated.turbopages.org/proxy_u/en-ru.ru.b1853e60-6766c48a-9ba04d18

[2] *Графы: основы теории, алгоритмы поиска*
URL: https://nuancesprog.ru/p/9269/