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. А. Тарпанов
Преподаватель: Н. С. Капралов
Группа: М8О-307Б-19
Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №9

Задача:
Разработать программу на языке C или C++, реализующую указанный алгоритм согласно заданию:

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 Дейкстры. Длина пути равна сумме весов ребер на этом пути. Граф не содержит петель и кратных ребер.

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

Задано целое число n. Необходимо найти количество натуральных (без нуля) чисел, которые меньше n по значению и меньше n лексикографически (если сравнивать два числа как строки), а так же делятся на m без остатка.

Формат входных данных

В первой строке заданы $1 \leq n \leq 10^5$, $1 \leq m \leq 10^5$, $1 \leq start \leq n$ и $1 \leq finish \leq n$. В следующих m строках записаны ребра. Каждая строка содержит три числа – номера вершин, соединенных ребром, и вес данного ребра. Вес ребра – целое число от 0 до 10^9.

Формат результата
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 Дейкстры является улучшенной версией алгоритма Форда-Беллмана, но он требует отсутствия рёбер отрицательного веса в графе. Такое требование обусловлено тем, что на каждой фазе ищется вершина с минимальным значение $d[u]$ - то есть пройденным путём. Требуется, чтобы пройденный путь не уменьшался.

Идея алгоритма - на каждой фазе просматривать не все рёбра, а только те, которые исходят из вершины u с минимальным значением $d[u]$. После рассмотрения вершина u помечается и затем никогда больше не рассматривается, так как при выборе этой вершины было взято минимальное $d[u]$ и меньше получено быть не может.

Простая реализация - на каждой фазе искать $d[u]$ за линейное время, тогда сложность алгоритма $O(n^2 + m)$. На каждой фазе ищем по всем d (это даёт n^2) и просмотр всех ребёр графа суммарно (даёт $+m$).

Продвинутая реализация использует std::priority_queue (можно и std::set) и позволяет находить минимальное $d[u]$ за логарифмическое время. Тогда нужно просто помещать в очередь новую вершину, если на текущей фазе удалось улучшить для неё результат. Сложность $O(m * logn)$, так как будут просмотреты все рёбра, но в очереди может быть больше, чем n вершин из-за плотности графа (на каждой фазе улучшается результат, поэтому может получится, что мы храним одну и ту же вершину с разными значениями d).

2 Исходный код

В файле main.cpp приводтся полное решение задачи.

Листинг main.cpp

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

#define ll unsigned long long

const ll INF = INT64_MAX;

struct TItem {
    ll weight;
    int id;

    friend bool operator < (const TItem & lhs, const TItem & rhs) {
        if (lhs.weight != rhs.weight) {
            return lhs.weight > rhs.weight;
        } else {
            return lhs.id < rhs.id;
        }
    }
};

int main() {
    int n, m, start, finish, u, v;
    ll weight;
    std::cin >> n >> m >> start >> finish;
    start--, finish--;
    std::vector<std::vector<std::pair<int, ll>>> graph(n);
    for (int i = 0; i < m; ++i) {
        std::cin >> u >> v >> weight;
        u--, v--;
        graph[u].push_back(std::make_pair(v, weight));
        graph[v].push_back(std::make_pair(u, weight));
    }
    std::vector<ll> d(n, INF);
    std::priority_queue<TItem> pq;
    d[start] = 0;
    std::vector<bool> marks(n);
    pq.push({0, start});
    while (!pq.empty()) {
        TItem curr = pq.top();
        pq.pop();
        u = curr.id;
```

```
44          if (u == finish) {
45              break;
46          }
47          if (!marks[u]) {
48              for (size_t i = 0; i < graph[u].size(); ++i) {
49                  v = graph[u][i].first, weight = graph[u][i].second;
50                  if (d[u] + weight < d[v]) {
51                      d[v] = d[u] + weight;
52                      pq.push({d[v],v});
53                  }
54              }
55              marks[u] = true;
56          }
57      }
58      if (d[finish] == INF) {
59          std::cout << "No solution\n";
60      } else {
61          std::cout << d[finish] << '\n';
62      }
63  }
```

3 Консоль

```
kng@kng-Legion-Y540-15IRH:~/CLionProjects/DA9/cmake-build-debug ./DA9
5 6 1 5
1 2 2
1 3 0
3 2 10
4 2 1
3 4 4
4 5 5
8
```

4 Тест производительности

Для изучения производетельности, сравним время работы программы на тестах разных размеров с $m = 10^3, 10^5$

```
1. $10^3$
/home/kng/CLionProjects/DA9/cmake-build-debug/DA9
No solution
10215ms
Process finished with exit code 0
2.$10^5$
/home/kng/CLionProjects/DA9/cmake-build-debug/DA9
No solution
328485ms
Process finished with exit code 0
```

5 Выводы

Во время выполнения работы, я изучил и реализовал алгоритм Дейкстры. Первая реализация работала за $O(n^2+m)$, и не зашла на чекер. Исходя из таких ограничений, мною было решено написать реализацию через std::priority_queue, сложность работы которой составляет $O(m * logn)$. Такая реализация работает значительно быстрее (сокращение с квадрата до логарифма), но является более сложной для понимания.