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: Т. А. Габдуллин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2019

Лабораторная работа №9

Задача: Разработать программу на языке C или C++, реализующую указанный алгоритм обхода графов.

Вариант: 4. Поиск кратчайшего пути между парой вершин алгоритмом Дейкстры

Описание: 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.

Входные данные: В первой строке заданы $1 \leq n, m \leq 105$, $1 \leq start, finish \leq n$. 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 109.

Выходные данные: Необходимо вывести одно число – длину кратчайшего пути между указанными вершинами. Если пути между указанными вершинами не существует, следует вывести строку *No solution* .

1 Метод решения

Здесь описывается алгоритм, который предложил голландский исследователь Дейкстра в 1959 г.

Заведём массив $d[]$, в котором для каждой вершины v будем хранить текущую длину $d[v]$ кратчайшего пути из s в v. Изначально $d[s] = 0$, а для всех остальных вершин эта длина равна бесконечности (при реализации на компьютере обычно в качестве бесконечности выбирают просто достаточно большое число, заведомо большее возможной длины пути):

$d[v] = \infty, v \neq s$

Кроме того, для каждой вершины v будем хранить, помечена она ещё или нет, т.е. заведём булевский массив u[]. Изначально все вершины не помечены, т.е.

$u[v] = \text{false}$

Сам алгоритм Дейкстры состоит из n итераций. На очередной итерации выбирается вершина v с наименьшей величиной d[v] среди ещё не помеченных, т.е.:

$d[v] = \min_{p:\ u[p]=\text{false}} d[p]$

(Понятно, что на первой итерации выбрана будет стартовая вершина s.)

Выбранная таким образом вершина v отмечается помеченной. Далее, на текущей итерации, из вершины v производятся релаксации: просматриваются все рёбра (v, to), исходящие из вершины v, и для каждой такой вершины to алгоритм пытается улучшить значение $d[to]$. Пусть длина текущего ребра равна len, тогда в виде кода релаксация выглядит как:

$d[to] = \min(d[to], d[v] + \text{len})$

На этом текущая итерация заканчивается, алгоритм переходит к следующей итерации (снова выбирается вершина с наименьшей величиной d, из неё производятся релаксации, и т.д.). При этом в конце концов, после n итераций, все вершины графа станут помеченными, и алгоритм свою работу завершает. Утверждается, что найденные значения $d[v]$ и есть искомые длины кратчайших путей из s в v.

Стоит заметить, что, если не все вершины графа достижимы из вершины s, то значения $d[v]$ для них так и останутся бесконечными. Понятно, что несколько последних итераций алгоритма будут как раз выбирать эти вершины, но никакой полезной работы производить эти итерации не будут (поскольку бесконечное расстояние не сможет прорелаксировать другие, даже тоже бесконечные расстояния). Поэтому алгоритм можно сразу останавливать, как только в качестве выбранной вершины берётся вершина с бесконечным расстоянием.

Восстановление путей. Разумеется, обычно нужно знать не только длины кратчайших путей, но и получить сами пути. Покажем, как сохранить информацию, достаточную для последующего восстановления кратчайшего пути из s до любой вершины.

Для этого достаточно так называемого массива предков: массива $p[]$, в котором для каждой вершины $v \neq s$ хранится номер вершины $p[v]$, являющейся предпоследней в кратчайшем пути до вершиныv. Здесь используется тот факт, что если мы возьмём кратчайший путь до какой-то вершины v, а затем удалим из этого пути последнюю вершину, то получится путь, оканчивающийся некоторой вершиной $p[v]$, и этот путь будет кратчайшим для вершины $p[v]$. Итак, если мы будем обладать этим массивом предков, то кратчайший путь можно будет восстановить по нему, просто каждый раз беря предка от текущей вершины, пока мы не придём в стартовую вершину s — так мы получим искомый кратчайший путь, но записанный в обратном порядке. Итак, кратчайший путь P до вершины v равен:

$P = (s, \ldots, p[p[p[v]]], p[p[v]], p[v], v)$

Осталось понять, как строить этот массив предков. Однако это делается очень просто: при каждой успешной релаксации, т.е. когда из выбранной вершины v происходит улучшение расстояния до некоторой вершины to, мы записываем, что предком вершины to является вершина v:

$p[to] = v$

2 Реализация:

Итак, алгоритм Дейкстры представляет собой n итераций, на каждой из которых выбирается непомеченная вершина с наименьшей величиной d[v], эта вершина помечается, и затем просматриваются все рёбра, исходящие из данной вершины, и вдоль каждого ребра делается попытка улучшить значение d[] на другом конце ребра.

Время работы алгоритма складывается из:

n раз поиск вершины с наименьшей величиной $d[v]$ среди всех непомеченных вершин, т.е. среди $O(n)$ вершин m раз производится попытка релаксаций При простейшей реализации этих операций на поиск вершины будет затрачиваться $O(n)$ операций, а на одну релаксацию — $O(1)$ операций, и итоговая асимптотика алгоритма составляет:

$O(n^2 + m)$

Алгоритм взят с сайта *e-maxx.ru*.

3 Листинг

```
#include <bits/stdc++.h>
using namespace std;
struct TGraph{
    vector<vector< pair<int,int> > > top;
    int count_top;
    int count_Side;
};

long long int Deikstra(TGraph &graph, int start, int finish) {
    vector<long long int> count(graph.count_top);
    priority_queue<pair<long long int, int> , vector<pair<long long int, int> >,
        greater<pair<long long int, int> > > que;

    for(int i = 0; i < graph.count_top; ++i) {
        count[i] = LLONG_MAX;
    }
    count[start] = 0;
    que.push(make_pair(0, start));

    while(!que.empty()) {
        int current = que.top().second;
        que.pop();
        vector<pair<int, int> >::iterator i;
        for (i = graph.top[current].begin(); i != graph.top[current].end(); ++i) {
            int tmp = i->first;
            int weight = i->second;

            if (count[tmp] > count[current] + weight) {
                count[tmp] = count[current] + weight;
                que.push(make_pair(count[tmp], tmp));
            }
        }
    }
    if(count[finish] == LLONG_MAX) {
        return -1;
    }
    return count[finish];
}

void Create_Graph(TGraph &graph, int &start, int &finish) {
    int top1;
    int top2;
    int weight;
    cin >> graph.count_top >> graph.count_Side >> start >> finish;
    graph.top.resize(graph.count_top);
    while(cin >> top1 >> top2 >> weight) {
        graph.top[top1 - 1].push_back(make_pair(top2 - 1, weight));
```

```
47          graph.top[top2 - 1].push_back(make_pair(top1 - 1, weight));
48      }
49  }
50
51  int main() {
52      int start;
53      int finish;
54      long long int result;
55      TGraph graph;
56      Create_Graph(graph, start, finish);
57      result = graph.count_top > 0 ? Deikstra(graph, start - 1, finish - 1) : -1;
58      if(result < 0) {
59          cout << "No solution" << endl;
60      } else {
61          cout << result << endl;
62      }
63      return 0;
64  }
```

4 Выводы

Задачи на нахождение кратчайших путей от заданной вершины до всех остальных вершин являются отдельным классом задачи на графы, в конкретной лабораторной работе был разобрал и реализован *Алгоритм Дейкстры*)
Задачи на графы довольно часто встречаются в различных областях, например таких, как олимпиадные контесты *opencup* и *codeforces*, в связи с этим, нужно понимать и уметь реализовывать такие обязательным алгоритмы, как *Алгоритм Джонсона, Дейкстры, Форда - Фалкерсона, Беллмана - Форда, Куна, обход в ширину, глубину* и другие.

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.))

[2] Edsger Dijkstra. A note on two problems in connexion with graphs

[3] *Нахождение кратчайших путей от заданной вершины до всех остальных вершин алгоритмом Дейкстры*
URL:`http://www.e-maxx-ru.1gb.ru/algo/dijkstra`