Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Дискретный анализ»

Студент: Е. А. Кондратьев
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19
Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №5

Задача: Вариант №6 Разработать программу на языке C или C++, реализующую указанный алгоритм согласно заданию:

Задан взвешенный ориентированный граф, состоящий из n вершин и m ребер. Вершины пронумерованы целыми числами от 1 до n. Необходимо найти длины кратчайших путей между всеми парами вершин при помощи алгоритма Джонсона. 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 и кратных ребер.

Формат входных данных:

В первой строке заданы 1 <= n <= 2000, 1 <= m <= 4000. В следующих m строках записаны ребра. Каждая строка содержит три числа – номера вершин, соединенных ребром, и вес данного ребра. Вес ребра – целое число от 109 до 109.

Формат результата:

Если граф содержит цикл отрицательного веса, следует вывести строку "Negative cycle"(без кавычек). В противном случае следует выести матрицу из n строк и n столбцов, где j-е число в i-й строке равно длине кратчайшего пути из вершины i в вершину j. Если такого пути не существует, на соответствующей позиции должно стоять слово "inf"(без кавычек). Элементы матрицы в одной строке разделяются пробелом.

1 Описание

Код

```
typedef std::vector<std::vector<int64_t>> TMatrix;

struct TEdge
{
    size_t from;
    size_t to;
    int64_t weigth;
};

struct TGraph
{
    size_t v, e;
    std::vector<TEdge> edges;
    TGraph() {}
    TGraph(size_t n, size_t m) : v(n), e(m) {}
};
```

```

bool operator<(TEdge const&, TEdge const&);
void Deikstra(TMatrix const&, size_t const&, TMatrix&, size_t const&);
bool BellmanFord(TGraph const&, size_t const&, TMatrix&);
bool Johnson(TGraph const&, TMatrix&);

bool operator<(TEdge const& p1, TEdge const& p2) { return p1.weigth > p2.weigth; }

void Deikstra(TMatrix const& gr, size_t const& node, TMatrix& dist, size_t const& n)
{
    dist[node][node] = 0;
    std::priority_queue<TEdge> pq;
    TEdge t = {node, 0, 0};
    pq.push(t);
    while(!pq.empty())
    {
        TEdge s = pq.top();
        pq.pop();
        for(size_t i = 0; i < n; ++i)
        {
            if(dist[node][i] - dist[node][s.from] > gr[s.from][i])
            {
                dist[node][i] = dist[node][s.from] + gr[s.from][i];
                TEdge p = { i, 0, dist[node][i] };
                pq.push(p);
            }
        }
    }
}

bool BellmanFord(TGraph const& gr, size_t const& node, TMatrix& dist)
{
    dist[node][node] = 0;
    for(size_t j = 0; j < gr.v - 1; ++j)
    {
        for(auto& i: gr.edges)
            if(dist[node][i.from] != INT64_MAX && dist[node][i.to] > dist[node][i.from]
                + i.weigth)
                dist[node][i.to] = (dist[node][i.from] + i.weigth);
    }
    for(auto& i: gr.edges)
```

```
        if(dist[node][i.from] != INT64_MAX && dist[node][i.to] > dist[node][i.from] + i
            .weigth)
            return false;
    return true;
}

bool Johnson(TGraph const& gr, TMatrix& dist)
{
    TGraph new_gr;
    new_gr.v = gr.v + 1;
    new_gr.e = gr.e + gr.v;
    new_gr.edges = gr.edges;
    for(size_t i = 0; i < gr.v; ++i)
        new_gr.edges.push_back(TEdge{gr.v, i, 0});
    TMatrix new_dist(1, std::vector<int64_t>(new_gr.v, 0));
    if(!BellmanFord(new_gr, 0, new_dist))
    {
        std::cout << "Negative cycle\n";
        return false;
    }
    TMatrix graph(gr.v, std::vector<int64_t>(gr.v, INT64_MAX));
    for(size_t i = 0; i < gr.v; ++i)
        graph[i][i] = 0;
    for(auto& i: gr.edges)
        graph[i.from][i.to] = i.weigth + new_dist[0][i.from] - new_dist[0][i.to];
    for(size_t i = 0; i < gr.v; ++i)
        Deikstra(graph, i, dist, gr.v);
    for(size_t i = 0; i < gr.v; ++i)
        for(size_t j = 0; j < gr.v; ++j)
            if(dist[i][j] != INT64_MAX)
                dist[i][j] = dist[i][j] + new_dist[0][j] - new_dist[0][i];
    return true;
}
```

```
int main()
{
    size_t n = 0;
    size_t m = 0;
    std::cin >> n >> m;
    TGraph gr(n, m);
    TMatrix dist(n, std::vector<int64_t>(n, INT64_MAX));
    for(size_t i = 0; i < n; ++i)
        dist[i][i] = 0;
    size_t from = 0;
    size_t to = 0;
    int64_t weight = 0;
    for(size_t k = 0; k < m; ++k)
    {
        std::cin >> from >> to >> weight;
        gr.edges.push_back(TEdge{from - 1, to - 1, weight});
    }
    if(Johnson(gr, dist))
    {
        for(size_t i = 0; i < n; ++i)
        {
            for(size_t j = 0; j < n; ++j)
            {
                if(dist[i][j] == INT64_MAX)
                    std::cout << "inf";
                else
```

```
27                  std::cout << dist[i][j];
28              if(j != n - 1)
29                  std::cout << ' ';
30          }
31          std::cout << "\n";
32      }
33  }
34  return 0;
35 }
```

2 Консоль

du@Du$ cat test.txt
5 4
1 2 -1
2 3 2
1 4 -5
3 1 1
du@Du$ ./solution<test.txt
0 -1 1 -5 inf
3 0 2 -2 inf
1 0 0 -4 inf
inf inf inf 0 inf
inf inf inf inf 0
Тест состоит из ввода графа и запуска алгоритма Джонсона 50000, 100000 и 200000 раз.
du@Du$ ./benchmark
Enter graph:
5 4
1 2 -1
2 3 2
1 4 -5
3 1 1
Enter test count:
50000
Time for Johnson on 50000 tests: 3 seconds
du@Du$ ./benchmark
Enter graph:
5 4
1 2 -1
2 3 2
1 4 -5
3 1 1
Enter test count:
100000
Time for Johnson on 100000 tests: 7 seconds
du@Du$ ./benchmark
Enter graph:
5 4
1 2 -1

2 3 2

1 4 -5

3 1 1

Enter test count:

200000

Time for Johnson on 200000 tests: 14 seconds

Алгоритм работает за O(n*m+n2*lnn), в нашем случае n =m -> O(n2*lnn), что лучше чем алгоритм Флойда-Уоршела, имеющего сложность O(n3) и явно лучше наивного перебора за O(n4).

3 Выводы

Выполнив девятую лабораторную работу по курсу «Дискретный анализ», я познакомился с алгоритмами на графах, такими как алгоритм Дейкстры, Беллмана-Форда и Джонсона, узнал в каких случаях нужно использовать алгоритм Джонсона и почему нельзя использовать Дейкстру(при наличии отрицательных ребер), для чего в алгоритме Джонсона нужен алгоритм Беллмана-Форда(для нахождения отрицательного цикла).

Список литературы

[1] *Поисковик - Google.*
URL: `https://www.google.com/`

[2] *Сайт с подробной документацией библиотек C++*
URL: `https://en.cppreference.com/`

[3] *Алгоритм Джонсона*
URL: `https://habr.com/ru/company/otus/blog/510942//`