Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету "Дискретный анализ" №9

Студент: Кострюков Е.С.

Преподаватель: Макаров Н.К.

Группа: М8О-307Б-22

Дата:

Оценка:

Подпись:

Москва 2024 г.

Оглавление

Цель работы

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.

Постановка задачи

Формат ввода

В первой строке заданы 1 <= *n* <= 2000 и 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 -10^9 до 10^9.

Формат вывода

Если граф содержит цикл отрицательного веса, следует вывести строку "*Negative cycle*" (без кавычек). В противном случае следует вывести 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.

Общий алгоритм решения

Алгоритм решения состоит из 5 шагов, каждый из которых вынесен в отдельную функцию:

1. Добавление фиктивной вершины (Шаг 1).

- Добавляется новая вершина *S* и рёбра от *S* ко всем вершинам исходного графа с весом 0. Это делается для удобства запуска алгоритма Беллмана-Форда, чтобы определить потенциалы (модифицированные значения) всех вершин.

2. Запуск Беллмана-Форда (Шаг 2).

- Основная цель этого шага — вычислить потенциалы *h(u)* для каждой вершины *u*, которые определяются как кратчайшие расстояния от *S* до *u*.
- Если обнаруживается отрицательный цикл — алгоритм завершается, так как вычислить кратчайшие пути в графе с отрицательными циклами невозможно.

3. Пересчёт весов рёбер (Шаг 3).

- Пересчитываем веса ребер по формуле: *w'(u, v) = w(u, v) + h(u) - h(v)*, где *w(u, v)* — исходный вес ребра, *h(u)* и *h(v)* — потенциалы начальной и конечной вершин соответственно.
- Этот шаг гарантирует, что все новые веса *w'(u, v)* будут неотрицательными. Это необходимо, чтобы корректно применять алгоритм Дейкстры.

4. Запуск Дейкстры для каждой вершины (Шаг 4).

- После пересчёта весов рёбер, запускаем алгоритм Дейкстры V раз (по одному разу для каждой вершины графа). Поскольку веса рёбер неотрицательные, Дейкстра работает корректно и эффективно.

5. Обратное преобразование весов (Шаг 5).

- Когда все кратчайшие пути подсчитаны с использованием новых весов $w'(u, v)$, необходимо преобразовать их обратно к исходным весам, чтобы восстановить правильные значения.
- Итоговая формула для длины пути между двумя вершинами u и v:

$$d(u, v) = d'(u, v) + h(v) - h(u),$$

где $d'(u, v)$ — кратчайший путь, рассчитанный на графе с пересчитанными весами, $h(u)$ и $h(v)$ — потенциалы вершин.

Вычисление сложности работы алгоритма:

1. Добавление фиктивной вершины (Шаг 1).

- Это занимает $O(n)$, так как добавляется одно ребро для каждой вершины графа.

Сложность шага: $O(n)$.

2. Алгоритм Беллмана-Форда (Шаг 2).

Алгоритм Беллмана-Форда работает за $O(V * E)$, где V — число вершин, а E — число рёбер. Учитывая, что была добавлена одна фиктивная вершина и n рёбер:

- $V = n + 1, E = m + n$.
- Сложность: $O((n + 1) * (m + n))$, что в асимптотике $O(n * m + n^2)$.

Сложность шага: $O(n * m + n^2)$.

3. Пересчёт весов рёбер (Шаг 3).

Пересчёт весов рёбер требует обхода всех рёбер и пересчёта их весов по формуле. Так как рёбер $m + n$, то сложность будет $O(m + n)$.

Сложность шага: $O(m + n)$.

4. Алгоритм Дейкстры для каждой вершины (Шаг 4).

Алгоритм Дейкстры работает за $O(E * \log V)$, если используется очередь с приоритетом. Для графа с n вершинами и m рёбрами:

- Каждая вершина запускает алгоритм Дейкстры: $O(n * (m * \log n))$.

Сложность шага: $O(n * m * \log n)$.

5. Обратное преобразование весов (Шаг 5).

Обратное преобразование выполняется для каждого элемента матрицы расстояний (всего n^2):

- Для каждого расстояния требуется одна операция: $O(n^2)$.

Сложность шага: $O(n^2)$.

Объединяя, получаем:

$$O(n + n * m + n^2 + m + n + n * m * \log n + n^2) = O(n * m * \log n + n^2 + n * m)$$

Ассимптотически доминирующий член:
*O(n * m * log n)*

Итоговая сложность:
*O(n * m * log n)*, где *n* - число вершин, *m* - число рёбер.

Реализация

```
#include <iostream>
#include <vector>
#include <queue>
#include <limits>
#include <tuple>

using namespace std;

const long long INF = numeric_limits<long long>::max();

struct Edge {
    int u, v;
    long long weight;
};

// Шаг 1: Добавляем новую вершину S и рёбра от S ко всем вершинам исходного графа с весом 0
void addFictitiousVertex(vector<Edge> &edges, int n) {
    for (int i = 1; i <= n; ++i) {
        edges.push_back({0, i, 0});
    }
}

// Шаг 2: Запускаем алгоритм Беллмана-Форда, чтобы найти негативные циклы и вычислить потенциалы
bool bellmanFord(int n, const vector<Edge> &edges, vector<long long> &h) {
    h.assign(n + 1, INF);
    h[0] = 0;

    for (int i = 0; i < n; ++i) {
        bool updated = false;
        for (const auto &edge : edges) {
            if (h[edge.u] < INF && h[edge.u] + edge.weight < h[edge.v]) {
                h[edge.v] = h[edge.u] + edge.weight;
                updated = true;
            }
        }
        if (!updated) break;
    }

    for (const auto &edge : edges) {
        if (h[edge.u] < INF && h[edge.u] + edge.weight < h[edge.v]) {
            return false; // Нашли негативный цикл
        }
    }
```

```
    return true;
}

// Шаг 3: Пересчитываем веса ребер, используя потенциалы
void reweightEdges(vector<Edge> &edges, const vector<long long> &h) {
    for (auto &edge : edges) {
        edge.weight += h[edge.u] - h[edge.v];
    }
}

// Шаг 4: Запуск Дейкстры для каждой вершины
void dijkstra(int start, int n, const vector<vector<pair<int, long long>>> &adj, vector<long long> &dist) {
    dist.assign(n + 1, INF);
    dist[start] = 0;

    priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long long, int>>> pq;
    pq.emplace(0, start);

    while (!pq.empty()) {
        auto top = pq.top(); pq.pop();
        long long d = top.first;
        int u = top.second;

        if (d > dist[u]) continue;

        for (const auto &edge : adj[u]) {
            int v = edge.first;
            long long weight = edge.second;
            if (dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
                pq.emplace(dist[v], v);
            }
        }
    }
}

// Шаг 5: Обратное преобразование весов
void revertWeights(int n, const vector<vector<long long>> &dist, const vector<long long> &h,
vector<vector<string>> &result) {
    for (int u = 1; u <= n; ++u) {
        for (int v = 1; v <= n; ++v) {
            if (dist[u][v] < INF) {
                long long originalDistance = dist[u][v] - h[u] + h[v];
                result[u - 1][v - 1] = to_string(originalDistance);
            }
        }
    }
}

// Шаг 6: Алгоритм Джонсона (объединение всех шагов)
vector<vector<string>> johnsonAlgorithm(int n, const vector<Edge> &edges) {
    vector<Edge> modifiedEdges = edges;
    addFictitiousVertex(modifiedEdges, n);
```

```
    vector<long long> h;
    if (!bellmanFord(n, modifiedEdges, h)) {
        return { {"Negative cycle"} };
    }

    reweightEdges(modifiedEdges, h);

    vector<vector<pair<int, long long>>> adj(n + 1);
    for (const auto &edge : modifiedEdges) {
        if (edge.u != 0) {
            adj[edge.u].emplace_back(edge.v, edge.weight);
        }
    }

    vector<vector<long long>> distMatrix(n + 1, vector<long long>(n + 1, INF));
    for (int u = 1; u <= n; ++u) {
        dijkstra(u, n, adj, distMatrix[u]);
    }

    vector<vector<string>> result(n, vector<string>(n, "inf"));
    revertWeights(n, distMatrix, h, result);

    return result;
}

int main() {
    int n, m;
    cin >> n >> m;

    vector<Edge> edges;
    for (int i = 0; i < m; ++i) {
        int u, v;
        long long weight;
        cin >> u >> v >> weight;
        edges.push_back({u, v, weight});
    }

    auto result = johnsonAlgorithm(n, edges);

    if (result.size() == 1 && result[0][0] == "Negative cycle") {
        cout << "Negative cycle" << endl;
    } else {
        for (const auto &row : result) {
            for (size_t i = 0; i < row.size(); ++i) {
                if (i > 0) cout << " ";
                cout << row[i];
            }
            cout << endl;
        }
    }

    return 0;
}
```

Пример работы

Ввод:	**Вывод:**
5 4	0 -1 1 -5 inf
1 2 -1	3 0 2 -2 inf
2 3 2	1 0 0 -4 inf
1 4 -5	inf inf inf 0 inf
3 1 1	inf inf inf inf 0

Ввод:	**Вывод:**
3 3	0 -2 -3
1 2 -2	inf 0 -1
2 3 -1	inf inf 0
1 3 4	

Ввод:	**Вывод:**
4 5	Negative cycle
1 2 1	
2 3 -2	
3 4 2	
4 2 -1	
1 4 10	

Вывод

В ходе выполнения данной лабораторной работы я изучил алгоритм Джонсона для нахождения кратчайших путей между всеми парами вершин взвешенного ориентированного графа. Я разобрался в теоретических основах алгоритма, включая его ключевые этапы: добавление фиктивной вершины, использование алгоритма Беллмана-Форда для проверки отрицательных циклов и пересчёта весов рёбер, а также применение алгоритма Дейкстры для поиска кратчайших путей.

Я научился комбинировать несколько алгоритмов для решения сложных задач и понял, как пересчёт весов помогает устранить проблему отрицательных рёбер. Также я освоил подход к обработке графов с отрицательными весами рёбер и проверке на наличие циклов отрицательного веса.

Результатом работы стало понимание важности комбинированного подхода в задачах на графах, а также умение применять сложные алгоритмы для обработки больших объёмов данных.