Форд-Беллман, Флойд

1 A. Кратчайшие пути

Разбор

В задаче необходимо посчитать расстояния от данной вершины s до остальных вершин. При этом стоит учитывать, что ответ может быть равен $+\infty$ в случае, когда вершина недостижима из s, и $-\infty$ в случае, когда на пути из s до вершины содержится цикл отрицательного веса.

Массив расстояний считается за $n-1$ итерацию алгоритма Форда-Беллмана, заодно находятся недостижимые вершины из s вершины. Начиная с n'ой итерации любая релаксация будет сигнализировать о наличии отрицательного цикла, потому обновленное расстояние можно смело приравнивать к $-\infty$.

Обратим внимание, что хватит $2n-1$ итераций алгоритма Форда-Беллмана - такого количества шагов достаточно, чтобы дойти из s до отрицательного цикла, пройти по нему и отправиться в нужную вершину (если, конечно, такое возможно).

Асимптотика решения: $O(|V||E|)$.

Решение

```
import math

n, m, s = map(int, input().split())
edges = [ tuple(map(int, input().split())) for _ in range(m)]

d = [math.inf]*(n+1)
d[s] = 0

for i in range(2*n-1):
    for u, v, w in edges:
        if d[v] > d[u]+w:
            d[v] = d[u]+w if i < n else -math.inf

for i in range(1, n+1):
    if d[i] == math.inf:
        print("*")
    elif d[i] == -math.inf:
        print("-")
    else:
        print(d[i])
```

2 B. Лабиринт знаний

Разбор

В этой задаче необходимо найти самый длинный путь между заданными вершинами. Эта задача эквивалентна поиску кратчайшего пути, достаточно изменить все знаки неравенств.

Кроме того, необходимо обработать случай положительного цикла. При этом требовалось проверить не сам факт наличия отрицательного цикла (он мог содержаться в изолированной компоненте), а наличие отрицательного цикла, который мог лежать на пути из 1 в n. Для этого сначала насчитаем массив can_finish – в нем будет храниться true для вершины, если до нее можно дойти из вершины 1 и из нее можно дойти до вершины n, это простая динамика на графе, которая решается одним запуском dfs, подробнее можно прочитать в коде.

Теперь достаточно запустить алгоритм Форда-Беллмана с n итерациями и проверить, что на последней итерации обновится значение любой вершины, у которой can_finish равен true. Если обновилось – можно

набрать неограниченное количество знаний, если не обновилось – мы знаем либо длину наибольшего пути, либо что пути не существует.

Асимптотика решения: $O(|V||E|)$

Решение

```
#include <iostream>
#include <vector>

using namespace std;

const int MAXN = 2e3 + 7;
const int MAXM = 1e4 + 7;
const int INF  = 1e9 + 7;

struct Edge {
    int a, b, cost;
} edges[MAXM];

vector<int> g[MAXN];

int d[MAXN];

int used[MAXN], can_finish[MAXN];
bool calc_can_finish(int v, int target) {
    if (used[v]) {
        return can_finish[v];
    }

    used[v] = true;
    if (v == target) {
        return can_finish[v] = true;
    }

    can_finish[v] = false;
    for (int i : g[v]) {
        can_finish[v] |= calc_can_finish(i, target);
    }
    return can_finish[v];
}

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

    for (int i = 0; i < m; ++i) {
        int a, b, cost;
        cin >> a >> b >> cost;
        g[a - 1].push_back(b - 1);
        edges[i] = {a - 1, b - 1, cost};
    }

    calc_can_finish(0, n - 1);

    fill(&d[0], &d[0] + MAXN, -INF);
    d[0] = 0;

    bool is_updated;
    for (int it = 0; it < n; ++it) {
        is_updated = false;
        for (auto& e : edges) {
            if (d[e.a] != -INF && d[e.a] + e.cost > d[e.b]) {
                d[e.b] = min(INF, d[e.a] + e.cost);
                if (can_finish[e.b]) {
                    is_updated = true;
                }
            }
        }
    }
```

```
    if (is_updated) {
        cout << ":)\n";
    } else if (d[n - 1] == -INF) {
        cout << ":(\n";
    } else {
        cout << d[n - 1] << '\n';
    }
}
```

3 C. Авиаперелеты

Разбор

Воспользуемся алгоритмом Форда-Беллмана для поиска кратчайших путей из S. Заметим, что после k шагов релаксации мы обязательно обработаем все кратчайшие пути длины не больше k: на k-м шаге мы попытаемся улучшить ответ для конца каждого ребра, поэтому если на $(k-1)$-м шагу мы нашли кратчайшие пути длины $\leq k-1$, мы найдем все и на k-м. Таким образом, если вместо классических $N-1$ итераций мы прорелаксируем K раз, то получим искомый ответ.

Важная деталь: в этой задаче нужно поддерживать два массива расстояний, один актуальный на предыдущем шагу и второй релаксируемый. Если все операции выполнять с одним массивом, то на одной итерации можно прорелаксировать два подряд идущих ребра $u \to v \to w$, за счет чего мы можем обновить кратчайший путь до w большей длины, хотя это должно было произойти на следующем шагу.

Асимптотика решения: $O(KM)$

Решение

```
#include <algorithm>
#include <iostream>
#include <vector>

using namespace std;

struct edge {
    int u, v, p;
};

int main() {
    int n, m, k, s, f;
    cin >> n >> m >> k >> s >> f;
    vector<edge> g(m);
    for (int i = 0; i < m; i++) {
        cin >> g[i].u >> g[i].v >> g[i].p;
    }
    vector<int> d0(n + 1, 1e9), d1(n + 1, 1e9);
    d0[s] = d1[s] = 0;
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < m; j++) {
            if (d0[g[j].u] < 1e9) {
                d1[g[j].v] = min(d1[g[j].v], d0[g[j].u] + g[j].p);
            }
        }
        for (int j = 1; j <= n; j++) {
            d0[j] = d1[j];
        }
    }
    cout << (d0[f] == 1e9 ? -1 : d0[f]) << '\n';
    return 0;
}
```

4 D. Цикл

Разбор

Добавим фиктивную вершину 0, и проведём из неё ориентированные рёбра во все другие вершины веса 0. Рассмотрим кратчайшие пути от вершины 0, до всех других вершин. Если в графе нет циклов отрицательного веса, то эти пути конечны, и количество ребер в них не больше $n - 1$. Если же в графе есть цикл отрицательного веса, то кратчайших путей не существует, так как можно бесконечно ходить по циклу отрицательного веса и уменьшать расстояние.

Запустим n раз алгоритм итерацию алгоритма Форда-Беллмана. После этого запустим ещё одну итерацию, и посмотрим, изменились ли расстояния до вершин. Если изменились, то цикл есть, иначе - нет.

Также будем дополнительно поддерживать для каждой вершины предыдущую вершину в кратчайшем пути, от вершины 0. Это позволяет восстановить сами пути до любой вершины, но поскольку у нас любой кратчайший путь будет проходить через цикл отрицательного веса, то мы сможем восстановить цикл, идя назад по этим пометкам из любой вершины.

Ассимптотика решения: $O(n^3)$

Решение

```
NON_EXIST=100000

n = int(input())

edges = []

for i in range(n):
    to = list(map(int, input().split()))
    for j in range(n):
        if to[j] != NON_EXIST:
            edges.append((i + 1, j + 1, to[j]))
    edges.append((0, i + 1, 0))

dist = [int(1e10) for _ in range(n + 1)]
prev = [-1 for _ in range(n + 1)]
dist[0] = 0

def iteration():
    for edge in edges:
        if dist[edge[0]] + edge[2] < dist[edge[1]]:
            dist[edge[1]] = dist[edge[0]] + edge[2]
            prev[edge[1]] = edge[0]

for _ in range(n):
    iteration()

old_dist = dist.copy()
iteration()

ans = False
changed = -1
for i in range(n + 1):
    if dist[i] != old_dist[i]:
        ans = True
        changed = i
        break

if not ans:
    print("NO")
else:
    print("YES")
    used = [False for _ in range(n + 1)]
    current = changed
    stack = []
    while not used[current]:
        used[current] = True
        stack.append(current)
```

```
        current = prev[current]

    cycle = [current]
    while stack[-1] != current:
        cycle.append(stack[-1])
        stack.pop(-1)
    cycle.append(current)
    print(len(cycle))
    print(*cycle)
```

5 E. Полнейший Флойд

Разбор

В этой задаче надо было написать алгоритм Флойда, но вовремя отловить наличие цикла отрицательного веса, чтобы не получить переполнение + надо было правильно обработать петли. Подробнее можно прочитать здесь

Решение

```
#include <bits/stdc++.h>
using namespace std;

void relax(long long& a, long long b) {
    a = min(a, b);
}

void Print(vector<int>& cycle) {
    cout << "LOOP\n";
    cout << cycle.size() << "\n";
    for (auto& v : cycle) {
        cout << v << " ";
    }
    cout << "\n";
}

long long INF;

long long add(long long a, long long b) {
    if (a == INF || b == INF) {
        return INF;
    }
    return a + b;
}

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

    int n, m;
    cin >> n >> m;

    INF = n * 1e5;

    vector<vector<long long>> d(n, vector<long long> (n, INF));
    vector<vector<int>> next(n, vector<int> (n));

    for (int i = 0; i < n; ++i) {
        d[i][i] = 0;
        next[i][i] = -1;
    }

    while (m --> 0) {
        int from, to, weight;
        cin >> from >> to >> weight;
        --from;--to;
```

```
            relax(d[from][to], weight);
            next[from][to] = to;
        }

        for (int k = 0; k < n; ++k) {
            for (int i = 0; i < n; ++i) {
                for (int j = 0; j < n; ++j) {
                    if (i == j && j == k) {
                        continue;
                    }

                    if (d[i][j] > add(d[i][k], d[k][j])) {
                        d[i][j] = add(d[i][k], d[k][j]);
                        next[i][j] = next[i][k];
                    }
                }
            }

            for (int i = 0; i < n; ++i) {
                if (d[i][i] < 0) {
                    int cur = i;
                    vector<int> cycle;
                    do {
                        cycle.push_back(cur + 1);
                        cur = next[cur][i];
                    } while (cur != i);

                    Print(cycle);
                    return 0;
                }
            }
        }

        cout << "NO LOOP\n";
        for (auto& row : d) {
            for (auto& v : row) {
                if (v == INF) {
                    cout << "INF ";
                } else {
                    cout << v << " ";
                }
            }
            cout << "\n";
        }
    }
```

6 F. Странствующий торговец

Разбор

В этой задаче необходимо найти максимальной эффективный цикл, то есть цикл, у которого отношение $\varepsilon = \frac{benefit}{length}$, где $benefit = \sum s_i - b_i$ — разница между стоимостью продажи и стомостью покупки какого-то товара на цикле (заметим, что покупок одного и тоже товара на разных участках может быть несколько), а $length$ — длина самого цикла. Давайте попробуем научиться проверять есть ли цикл с эффективностью **хотя бы** ε, то есть $\frac{benefit}{length} \geqslant \varepsilon \iff benefit \geqslant length \cdot \varepsilon \iff \varepsilon \cdot length - benefit \leqslant 0$. Если мы научимся решать такую задачу, то далее можно применить двоичный поиск по ответу.

Заметим, что нам нужно найти отрицательный цикл на каком-то графе. Для построения графа мысленно разобъем наш цикл на кусочки вида $v_1 \to v_2, v_2 \to v_3, \ldots, v_l \to v_1$, где на каждом кусочке мы везли какой-то товара (либо ехали пустыми). На каждом кусочке мы легко умеем считать его $benefit$ и $length$. Для подсчета $benefit$ на нужно просто выбрать оптимальный (с максимальной разницей цен закупки и продажи) товара, а для подсчета $length$ нужно воспользовать алгоритмом Флойда (так как нас интересует максимально быстро добраться из точки А в точку Б).

Теперь реализуем саму проверку. Для этого нам нужно построить граф, где мы между любымии двумя

вершинами проводим ребро $\varepsilon \cdot length - benefit$. Каждое ребро означает кусочек цикла, следовательно, нам остается найти цикл из кусочков отрицательного веса. Это можно сделать еще одним запуском алгоритма Флойда.

Итого: делаем предподсчет длины и выгоды между двумя любыми вершинами за $\mathcal{O}(n^3+n^2k)$ и дальше делаем $\log(answer)$ проверок за $\mathcal{O}(n^3)$.

Решение

```
#include <bits/stdc++.h>

template<typename T> void domax(T &a, T b) { if (b > a) a = b; }
template<typename T> void domin(T &a, T b) { if (b < a) a = b; }

const long long inf = 3e18, MaxT = 10ll*1000*1000;
const int MaxN = 105, MaxK = 1005;

int n, m, K;
long long b[MaxN][MaxK], s[MaxN][MaxK], dist[MaxN][MaxN], best[MaxN][MaxN], cost[MaxN][MaxN];

bool can(long long eff) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dist[i][j] < inf) {
                cost[i][j] = dist[i][j] * eff - best[i][j];
            } else {
                cost[i][j] = inf;
            }
        }
    }
    for (int m = 0; m < n; m++) {
        for (int s = 0; s < n; s++) {
            for (int e = 0; e < n; e++) {
                domin(cost[s][e], cost[s][m] + cost[m][e]);
                if (cost[s][e] < -inf) {
                    cost[s][e] = -inf;
                }
            }
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (cost[i][j] + cost[j][i] <= 0) {
                return true;
            }
        }
    }
    return false;
}

int main() {
    scanf("%d%d%d", &n, &m, &K);
    for (int i = 0; i < n; i++) {
        for (int k = 0; k < K; k++) {
            scanf("%lld%lld", b[i]+k, s[i]+k);
            if (b[i][k] == -1) b[i][k] = inf;
            if (s[i][k] == -1) s[i][k] = -inf;
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            dist[i][j] = inf;
        }
    }
    for (int i = 0; i < m; i++) {
        int v, w; long long t;
        scanf("%d%d%lld", &v, &w, &t);
        v--; w--;
        dist[v][w] = t;
```

```
        assert(t <= 10ll*1000*1000);
    }

    for (int m = 0; m < n; m++) {
        for (int s = 0; s < n; s++) {
            for (int e = 0; e < n; e++) {
                domin(dist[s][e], dist[s][m] + dist[m][e]);
            }
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            best[i][j] = 0; // carrying no item
            for (int k = 0; k < K; k++) {
                domax(best[i][j], s[j][k] - b[i][k]);
            }
        }
    }

    long long lo = 1ll, hi = 1000ll * 1000 * 1000;
    long long ans = 0ll;
    while (lo <= hi) {
        long long mid = (lo + hi) / 2;
        if (can(mid)) {
            lo = mid + 1;
            ans = mid;
        } else {
            hi = mid - 1;
        }
    }
    assert(!can(ans+1));

    printf("%lld\n", ans);
}
```