Алгоритм решения задачи "Метки рёбер"

Антон Дроздовский

1 Условие задачи

Вам дано дерево порядка n. Также известен набор троек (u_i, r_i, v_i). Необходимо расставить на каждом ребре метку 0 или 1, чтобы в результате для каждой тройки (u_i, v_i, r_i) из набора выполнялось свойство: r_i равно остатку от деления на 2 суммы меток рёбер вдоль (u_i, v_i)-цепи.

Подсчитайте, сколько существует способов расставить метки с соблюдением описанных условий. Поскольку это число может оказаться большим, найдите остаток от деления этого числа на 1 000 000 007.

Input

В первой строке записано число n вершин дерева ($1 \leq n \leq 200000$). В последующих $n - 1$ строках записано по два числа — номера концов очередного ребра. Далее в отдельной строке следует число k троек ($0 \leq k \leq 200000$). Затем в каждой из k строк записано по три числа — u_i, v_i и r_i ($1 \leq u_i, v_i \leq n, 0 \leq r_i < 2$).

Output

Выведите искомый остаток от деления на 1 000 000 007 числа способов расставить метки на рёбрах дерева.

Examples

labels.in	labels.out
4 1 2 2 3 3 4 2 1 3 1 2 4 1	2
3 1 2 1 3 3 1 2 1 1 3 1 2 4 1	0

2 Описание алгоритма

Для решения задачи, можно использовать DSU с поддержкой паритета. Изначально каждая вершина является своим собственным представителем, имеет нулевой ранг и паритет.

Для каждого запроса пытаемся объединить множества, к которым принадлежат вершины, с учётом паритета:

- Вершины в одном множестве. Если XOR паритетов вершин не совпадает с паритетом запроса, то задачу невозможно решить.
- Вершины в разных множествах. Объединяем множества и обновляем паритет с учётом запроса.

Метки на рёбрах внутри одной компоненты связности должны быть согласованы, а между компонентами можно выбирать метки независимо. Отсюда число возможных способов расставить метки на рёбрах равно $2^{components-1}$.

3 Решение задачи на C++

```cpp
#define _CRT_SECURE_NO_WARNINGS

#include <algorithm>
#include <cstdio>
#include <ios>
#include <iostream>
#include <tuple>
#include <utility>
#include <vector>

const int MOD = 1000000007;

struct DSU {
    std::vector<int> parent, rank, parity;
    int components;

    DSU(int n) : parent(n), rank(n, 0), parity(n, 0) {
        for (int i = 0; i < n; ++i) {
            parent[i] = i;
            components = n;
        }
    }

    int find(int x) {
        if (parent[x] != x) {
            int px = find(parent[x]);
            parity[x] ^= parity[parent[x]];
            parent[x] = px;
```

```
        }
        return parent[x];
    }

    bool unionSets(int x, int y, int r) {
        int px = find(x);
        int py = find(y);
        if (px == py) {
            return (parity[x] ^ parity[y]) == r;
        }
        if (rank[px] > rank[py]) {
            std::swap(px, py);
            std::swap(x, y);
        }
        parent[px] = py;
        parity[px] = parity[x] ^ parity[y] ^ r;
        --components;
        if (rank[px] == rank[py]) {
            ++rank[py];
        }
        return true;
    }

    int getComponents() const {
        return components;
    }
};

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

    std::freopen("labels.in", "r", stdin);
    std::freopen("labels.out", "w", stdout);

    int n;
    std::cin >> n;

    std::vector<std::pair<int, int>> edges(n - 1);
    for (auto& edge : edges) {
        std::cin >> edge.first >> edge.second;
        --edge.first;
        --edge.second;
    }

    int k;
    std::cin >> k;
    std::vector<std::tuple<int, int, int>> queries(k);
    for (auto& query : queries) {
        int u, v, r;
```

```
        std::cin >> u >> v >> r;
        queries.push_back({ u - 1, v - 1, r });
    }

    DSU dsu(n);

    for (const auto& [u, v, r] : queries) {
        if (!dsu.unionSets(u, v, r)) {
            std::cout << 0 << '\n';
            return 0;
        }
    }

    long long result = 1;
    for (int i = 0; i < dsu.getComponents() - 1; ++i) {
        result = (result * 2) % MOD;
    }

    std::cout << result << '\n';
    return 0;
}
```

4 Оценка временной сложности

Инициализация DSU и обработка запросов — $O(n + k \cdot \alpha(n))$

5 Оценка сложности по памяти

Для хранения DSU и запросов требуется $O(n + k)$ по памяти