

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерных технологий

Продвинутые алгоритмы и структуры данных

Отчет о задаче № Н

Выполнила  
студентка

Ершова А. И.

Группа № Р4115

Преподаватель: Косяков Михаил Сергеевич

г. Санкт-Петербург

2024

## Условие задачи

# Н. Темпоральный катаклизм

Ограничение времени	1 секунда
Ограничение памяти	50.0 Мб
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Доктор Кто заметил, что каждое решение, принятое человечеством, создает новую временную ветвь, разделяя поток времени на бесчисленные альтернативные реальности. Эти ветви переплетаются в сложное древо, где каждая развилка - это новый вариант будущего.

Однако недавний темпоральный катаклизм нарушил естественный ход времени, и теперь все эти ветви начинают нестабильно взаимодействовать друг с другом, угрожая привести к разрушению временного континуума. Чтобы предотвратить распад реальности, Доктору Кто необходимо перемещаться между ключевыми временными точками и восстанавливать порядок.

Но есть одна загвоздка: заряд ТАРДИС - машины времени Доктора - ограничен, и каждый прыжок по временным линиям требует определённого количества энергии. Доктор должен оценить, хватит ли заряда для перемещения от одной точки к другой.

Помогите Доктору Кто справиться с последствиями темпорального катаклизма, путешествуя по ветвям времени и восстанавливая баланс в альтернативных реальностях!

## Формат ввода

На вход подаётся следующее:

Первой строкой задаётся число  $N$  - количество временных точек (вершин) в древе, такое что  $1 \leq N \leq 10^5$

Следующие  $N-1$  строк описывают связи (рёбра) между временными точками. Каждая строка содержит два числа  $u$  и  $v$ , которые обозначают соединение между временными точками  $u$  и  $v$ . Эти идентификаторы - натуральные числа от 0 до  $N-1$ , уникальные для каждой временной точки.

После этого задаётся число  $Q$  - количество запросов на перемещение, такое что  $Q \leq 10^6$ . Следующие  $Q$  строк содержат описание запросов. Каждый запрос представлен тремя числами: два идентификатора временных точек  $u$  и  $v$ , между которыми Доктор хочет переместиться, и величина заряда  $T$ , доступного для этого перемещения, т.е. максимально возможное расстояние на которое можно переместиться, измеряемое в числе пройденных ребер, данная величина не превосходит  $N$ .



## Формат вывода

На выходе для каждого запроса необходимо вывести одну строку:

"Yes", если Доктор может переместиться между двумя указанными временными точками при доступном заряде  $T$  (то есть если расстояние между точками меньше либо равно  $T$ ). "No", если заряд недостаточен для совершения перемещения (то есть если расстояние больше  $T$ ).

Каждый ответ выводится в отдельной строке, в том же порядке, в каком поступили запросы.

### Пример 1

Ввод 	Вывод 
7	Yes
0 1	No
1 2	No
1 4	
4 5	
4 6	
2 3	
3	
6 5 3	
3 5 2	
0 6 1	

## Пример 2

Ввод



Вывод



10

2 0

3 0

4 0

5 0

6 0

7 0

8 0

9 0

1 9

5

0 1 1

1 2 3

1 3 3

1 5 3

1 9 3

No

Yes

Yes

Yes

Yes

## Решение:

```
#include <fstream>
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

const int LOG = 17; // log2(MAXN) + 1

vector<vector<int>> tree;
vector<int> depth;
vector<vector<int>> binary_up;

void preparation_lca(const int n) {
    queue<int> q;
    depth[0] = 0; // Корень дерева
    q.push(0);

    while (!q.empty()) {
```

```

int node = q.front();
q.pop();
for (int neighbor : tree[node]) {
    if (depth[neighbor] == -1) {
        depth[neighbor] = depth[node] + 1;
        binary_up[neighbor][0] = node;
        q.push(neighbor);
    }
}
}

for (int j = 1; j < LOG; ++j) {
    for (int i = 0; i < n; ++i) {
        if (binary_up[i][j - 1] != -1) {
            binary_up[i][j] = binary_up[binary_up[i][j - 1]][j - 1];
        }
    }
}
}

string find_ans_LCA(int u, int v, int t) {
    if (depth[u] < depth[v]) {
        swap(u, v); // Пусть u глубже
    }

    int distance = 0;
    int diff = depth[u] - depth[v];

    for (int i = 0; i < LOG; ++i) {
        if (diff & (1 << i)) {
            u = binary_up[u][i];
            distance += (1 << i);
            if (distance > t) {
                return "No";
            }
        }
    }

    if (u == v) {
        return "Yes";
    }

    for (int i = LOG - 1; i >= 0; --i) {
        if (binary_up[u][i] != binary_up[v][i]) {
            u = binary_up[u][i];
            v = binary_up[v][i];
            distance += (1 << (i + 1));
            if (distance > t) {
                return "No";
            }
        }
    }
}

```

```

    }
}

distance += 2;
if (distance > t) {
    return "No";
}

return "Yes";
}

int main() {
    ifstream fin("../input.txt");
    int n; // кол-во вершин в дереве
    fin >> n;

    tree.resize(n);
    depth.resize(n, -1);
    binary_up.resize(n, vector<int>(LOG, -1));

    // for (int i = 0; i < n; ++i) {
    //     tree[i].resize(n, 0);
    //     for (int j = 0; j < LOG; ++j) {
    //         binary_up[i][j] = -1;
    //     }
    // }

    for (int i = 0; i < n - 1; ++i) {
        int u, v;
        fin >> u >> v;
        tree[u].push_back(v);
        tree[v].push_back(u);
    }

    preparation_lca(n);

    int q; // кол-во запросов
    fin >> q;

    for (int i = 0; i < q; ++i) {
        int u, v; // номера вершин
        int t; // величина доступного заряда для перемещения
        fin >> u >> v >> t;
        cout << find_ans_LCA(u, v, t) << '\n';
    }
    fin.close();
    return 0;
}

```

**Сложность:**  $O((n+q)\log n)$

**Объяснение решения:**

Для решения данной задачи используется метод двоичного подъема для нахождения наименьшего общего предка (LCA) двух вершин в дереве. Вначале проводится предобработка дерева, включающая вычисление глубин всех вершин относительно корня с помощью обхода в ширину (BFS), а также построение массива `binary_up`. Этот массив хранит информацию о том, на какую вершину можно попасть, поднявшись на  $2^j$  уровней вверх от текущей вершины. На первом этапе массив заполняется для  $j=0$ , где для каждой вершины указывается её непосредственный родитель. Затем с помощью рекуррентной формулы  $\text{binary\_up}[u][j] = \text{binary\_up}[\text{binary\_up}[u][j-1]][j-1]$  заполняются остальные уровни для  $j>0$ .

Для обработки запросов проверяется, можно ли пройти от вершины  $u$  к вершине  $v$  с доступным количеством шагов  $t$ . Сначала вершины  $u$  и  $v$  приводятся к одной глубине. Если текущая глубина  $u$  больше, чем у  $v$ , то  $u$  поднимается вверх до уровня  $v$  с использованием массива `binary_up`, уменьшая разницу в глубинах. После этого, если вершины совпадают, путь считается допустимым. Если нет, алгоритм начинает одновременно поднимать обе вершины вверх, начиная с самого большого шага ( $2^{(\text{LOG}-1)}$ ), пока их родители не совпадут. В конце LCA определяется как родитель обеих вершин, после чего расстояние между  $u$  и  $v$  суммируется с учетом всех шагов подъема и проверяется на соответствие  $t$ . Если длина пути превышает  $t$  на любом этапе, возвращается "No", иначе — "Yes".