# HoLD on

Zaky Zein

| Time limit | 2s |
|---|---|
| Memory limit | 256MB |

Diberikan sebuah graf tak berarah, terhubung, dan tidak memiliki siklus (tree) yang terdiri dari N buah simpul (nodes). Simpul-simpul tersebut diberi nomor unik dari 1 hingga N. Struktur pohon ini memiliki akar (root) yang ditetapkan pada Simpul 1.

Setiap simpul pada pohon memiliki sebuah fungsi state biner. Pada kondisi awal, seluruh simpul berada dalam State **Merah**.

Kamu diminta untuk memproses Q buah instruksi secara dinamis. Instruksi yang diberikan terdiri dari dua jenis operasi sebagai berikut:

- 0 i : Mengubah secara **inversi** state dari simpul i. Jika simpul i saat ini berada dalam State Merah, maka ubah menjadi State Hitam. Sebaliknya, jika simpul i berada dalam State Hitam, kembalikan menjadi State Merah.
- 1 v : Melakukan traversal dari root (Simpul 1) menuju simpul target v. Cari dan kembalikan indeks simpul pertama yang berada dalam State Hitam yang ditemui di sepanjang jalur penelusuran tersebut. Jika seluruh simpul dari Simpul 1 hingga Simpul v berada dalam State Merah, maka operasi ini harus mengembalikan nilai -1.

## Format Input

Baris pertama berisi dua buah bilangan bulat N dan Q (1 ≤ N, Q ≤ 10^5), berturut-turut menyatakan jumlah simpul pada pohon dan jumlah instruksi yang harus diproses.

N-1 baris berikutnya masing-masing baris berisi dua bilangan bulat u dan v (1 ≤ u, v ≤ N) yang merepresentasikan sebuah edge (sisi) tak berarah yang menghubungkan simpul u dan simpul v.

Q baris terakhir masing-masing baris berisi satu instruksi dalam format "0 i" atau "1 v" (1 ≤ i, v ≤ N).

## Format Output

Untuk setiap instruksi jenis "1 v", cetak satu baris berisi satu bilangan bulat yang menyatakan hasil kueri penelusuran (indeks simpul Hitam pertama yang ditemui, atau -1 jika jalur bersih dari simpul berwarna Hitam).

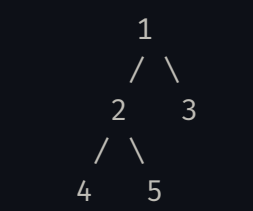
## Input

```
5 8
1 2
1 3
2 4
2 5
0 3
0 4
1 4
1 3
1 1
0 3
1 3
1 4
```

## Output

```
4
3
-1
-1
4
```

# Penjelasan




Op 1: 0 3 → toggle simpul 3, merah → hitam
State: {3}

Op 2: 0 4 → toggle simpul 4, merah → hitam
State: {3, 4}

Op 3: 1 4 → cari simpul hitam pertama di path 1 → 2 → 4
simpul 3 memang hitam, tapi simpul 3 ada di cabang lain (anak dari simpul 1 via sisi kanan).
simpul 3 tidak dilalui path 1 → 2 → 4, jadi tidak dihitung.
simpul 4 hitam dan ada di path → jawaban 4

Op 4: 1 3 → cari simpul hitam pertama di path 1 → 3
simpul 4 hitam, tapi ada di cabang kiri (via simpul 2), tidak dilalui path ini.
simpul 3 hitam dan ada di path → jawaban 3

Op 5: 1 1 → cari simpul hitam pertama di path 1 (hanya root)
Path hanya terdiri dari simpul 1 saja. simpul 1 merah → jawaban -1

Op 6: 0 3 → toggle simpul 3, hitam → merah
State: {4}

Op 7: 1 3 → cari simpul hitam pertama di path 1 → 3
simpul 3 sekarang merah. simpul 1 merah. Tidak ada simpul hitam di path ini → jawaban -1

Op 8: 1 4 → cari simpul hitam pertama di path 1 → 2 → 4
simpul 4 hitam dan ada di path → jawaban 4

# HoLD on

| Time limit | 2s |
|---|---|
| Memory limit | 256MB |

You are given an undirected, connected, and acyclic graph (a tree) consisting of N nodes, uniquely numbered from 1 to N. The tree is rooted at Node 1.

Each node in the tree has a binary state. Initially, all nodes are in the **Red State**.

You are required to process Q dynamic queries. There are two types of operations:

- **0 i**: Toggle the state of node i. If node i is currently in the Red State, change it to the Black State. Conversely, if it is in the Black State, change it back to the Red State.
- **1 v**: Traverse the path from the root (Node 1) to the target node v. Find and return the index of the **first node in the Black State** encountered along this path. If all nodes from Node 1 to Node v are in the Red State, return -1.

## Input Format

The first line contains two integers, N and Q  (1 ≤ N, Q ≤  10^5), representing the number of nodes in the tree and the number of queries to process, respectively.

Each of the next N-1 lines contains two integers, u and v (1 ≤ u, v ≤ N), representing an undirected edge connecting node u and node v.

Each of the last Q lines contains a query in the format “0 i” or “1 v” (1 ≤ i, v ≤ N).

## Output Format

For each query of type 1 v, print a single integer on a new line representing the result of the path traversal (the index of the first Black node encountered, or -1 if the path contains only Red nodes).

## Input

```
5 8
1 2
1 3
2 4
2 5
0 3
0 4
1 4
1 3
1 1
0 3
1 3
1 4
```

## Output

```
4
3
-1
-1
4
```

# Explanation


Op 1: 0 3 → Toggle node 3 (Red → Black)
Current State: {3}

Op 2: 0 4 → Toggle node 4 (Red → Black)
Current State: {3, 4}

Op 3: 1 4 → Find the first Black node on the path 1 → 2 → 4
Node 3 is indeed Black, but it is located in a different branch (a child of node 1 via the right side).
Since node 3 is not visited along the path 1 → 2 → 4, it is ignored.
Node 4 is Black and lies on the path → Output: 4

Op 4: 1 3 → Find the first Black node on the path 1 → 3
Node 4 is Black, but it is located in the left branch (via node 2) and is not visited on this path.
Node 3 is Black and lies on the path → Output: 3

Op 5: 1 1 → Find the first Black node on the path 1 (root only)
The path consists only of node 1. Node 1 is Red → Output: -1

Op 6: 0 3 → Toggle node 3 (Black → Red)
Current State: {4}

Op 7: 1 3 → Find the first Black node on the path 1 → 3
Node 3 is now Red. Node 1 is Red. There are no Black nodes on this path → Output: -1

Op 8: 1 4 → Find the first Black node on the path 1 → 2 → 4
Node 4 is Black and lies on the path → Output: 4