1993. Operations on Tree

Medium Topics Companies Hint

You are given a tree with `n` nodes numbered from `0` to `n - 1` in the form of a parent array `parent` where `parent[i]` is the parent of the

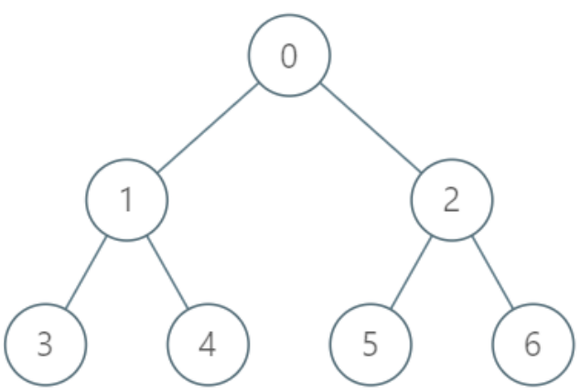
The data structure should support the following functions:

- **Lock: Locks** the given node for the given user and prevents other users from locking the same node. You may only lock a node using this
- **Unlock: Unlocks** the given node for the given user. You may only unlock a node using this function if it is currently locked by the same u
- **Upgrade: Locks** the given node for the given user and **unlocks** all of its descendants **regardless** of who locked it. You may only upgrade
  - The node is unlocked,
  - It has at least one locked descendant (by **any** user), and
  - It does not have any locked ancestors.

Implement the `LockingTree` class:

- `LockingTree(int[] parent)` initializes the data structure with the parent array.
- `lock(int num, int user)` returns `true` if it is possible for the user with id `user` to lock the node `num`, or `false` otherwise. If it is pos
- `unlock(int num, int user)` returns `true` if it is possible for the user with id `user` to unlock the node `num`, or `false` otherwise. If it i
- `upgrade(int num, int user)` returns `true` if it is possible for the user with id `user` to upgrade the node `num`, or `false` otherwise. If

Example 1:




```
Input
["LockingTree", "lock", "unlock", "unlock", "lock", "upgrade", "lock"]
[[[-1, 0, 0, 1, 1, 2, 2]], [2, 2], [2, 3], [2, 2], [4, 5], [0, 1], [0, 1]]
Output
[null, true, false, true, true, true, false]

Explanation
LockingTree lockingTree = new LockingTree([-1, 0, 0, 1, 1, 2, 2]);
lockingTree.lock(2, 2);    // return true because node 2 is unlocked.
                           // Node 2 will now be locked by user 2.
lockingTree.unlock(2, 3);  // return false because user 3 cannot unlock a node locked by user 2.
lockingTree.unlock(2, 2);  // return true because node 2 was previously locked by user 2.
                           // Node 2 will now be unlocked.
lockingTree.lock(4, 5);    // return true because node 4 is unlocked.
                           // Node 4 will now be locked by user 5.
lockingTree.upgrade(0, 1); // return true because node 0 is unlocked and has at least one locked des
                           // Node 0 will now be locked by user 1 and node 4 will now be unlocked.
lockingTree.lock(0, 1);    // return false because node 0 is already locked.
```

Constraints:

- `n == parent.length`
- `2 <= n <= 2000`