Problem 1600: Throne Inheritance

Problem Information

Difficulty: Medium
Acceptance Rate: 66.50%
Paid Only: No
Tags: Hash Table, Tree, Depth-First Search, Design

Problem Description

A kingdom consists of a king, his children, his grandchildren, and so on. Every once in a while, someone in the family dies or a child is born.

The kingdom has a well-defined order of inheritance that consists of the king as the first member. Let's define the recursive function `Successor(x, curOrder)`, which given a person `x` and the inheritance order so far, returns who should be the next person after `x` in the order of inheritance.

Successor(x, curOrder): if x has no children or all of x's children are in curOrder: if x is the king return null else return Successor(x's parent, curOrder) else return x's oldest child who's not in curOrder

For example, assume we have a kingdom that consists of the king, his children Alice and Bob (Alice is older than Bob), and finally Alice's son Jack.

1. In the beginning, `curOrder` will be `["king"]`. 2. Calling `Successor(king, curOrder)` will return Alice, so we append to `curOrder` to get `["king", "Alice"]`. 3. Calling `Successor(Alice, curOrder)` will return Jack, so we append to `curOrder` to get `["king", "Alice", "Jack"]`. 4. Calling `Successor(Jack, curOrder)` will return Bob, so we append to `curOrder` to get `["king", "Alice", "Jack", "Bob"]`. 5. Calling `Successor(Bob, curOrder)` will return `null`. Thus the order of inheritance will be `["king", "Alice", "Jack", "Bob"]`.

Using the above function, we can always obtain a unique order of inheritance.

Implement the `ThroneInheritance` class:

* `ThroneInheritance(string kingName)` Initializes an object of the `ThroneInheritance` class. The name of the king is given as part of the constructor. * `void birth(string parentName, string childName)` Indicates that `parentName` gave birth to `childName`. * `void death(string name)` Indicates the death of `name`. The death of the person doesn't affect the `Successor` function nor the current inheritance order. You can treat it as just marking the person as dead. * `string[] getInheritanceOrder()` Returns a list representing the current order of inheritance **excluding** dead people.

Example 1:

Input ["ThroneInheritance", "birth", "birth", "birth", "birth", "birth", "birth", "getInheritanceOrder", "death", "getInheritanceOrder"] [["king"], ["king", "andy"], ["king", "bob"], ["king", "catherine"], ["andy", "matthew"], ["bob", "alex"], ["bob", "asha"], [null], ["bob"], [null]] **Output** [null, null, null, null, null, null, null, ["king", "andy", "matthew", "bob", "alex", "asha", "catherine"], null, ["king", "andy", "matthew", "alex", "asha", "catherine"]] **Explanation** ThroneInheritance t= new ThroneInheritance("king"); // order: **king** t.birth("king", "andy"); // order: king > **andy** t.birth("king", "bob"); // order: king > andy > **bob** t.birth("king", "catherine"); // order: king > andy > bob > **catherine** t.birth("andy", "matthew"); // order: king > andy > **matthew** > bob > catherine t.birth("bob", "alex"); // order: king > andy > matthew > bob > **alex** > catherine t.birth("bob", "asha"); // order: king > andy > matthew > bob > alex > **asha** > catherine t.getInheritanceOrder(); // return ["king", "andy", "matthew", "bob", "alex", "asha", "catherine"] t.death("bob"); // order: king > andy > matthew > **~~bob~~** > alex > asha > catherine t.getInheritanceOrder(); // return ["king", "andy", "matthew", "alex", "asha", "catherine"]

Constraints:

* `1 <= kingName.length, parentName.length, childName.length, name.length <= 15` * `kingName`, `parentName`, `childName`, and `name` consist of lowercase English letters only. * All arguments `childName` and `kingName` are **distinct**. * All `name` arguments of `death` will be passed to either the constructor or as `childName` to `birth` first. * For each call to `birth(parentName, childName)`, it is guaranteed that `parentName` is alive. * At most `105` calls will be made to `birth` and `death`. * At most `10` calls will be made to `getInheritanceOrder`.

Code Snippets

C++:

```
class ThroneInheritance {
public:
ThroneInheritance(string kingName) {

}

void birth(string parentName, string childName) {

}

void death(string name) {

}

vector<string> getInheritanceOrder() {

}
};

/**
* Your ThroneInheritance object will be instantiated and called as such:
* ThroneInheritance* obj = new ThroneInheritance(kingName);
* obj->birth(parentName,childName);
* obj->death(name);
* vector<string> param_3 = obj->getInheritanceOrder();
*/
```

Java:

```
class ThroneInheritance {

public ThroneInheritance(String kingName) {

}

public void birth(String parentName, String childName) {

}

public void death(String name) {

}
```

```
public List<String> getInheritanceOrder() {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* ThroneInheritance obj = new ThroneInheritance(kingName);
* obj.birth(parentName,childName);
* obj.death(name);
* List<String> param_3 = obj.getInheritanceOrder();
*/
```

Python3:

```
class ThroneInheritance:

def __init__(self, kingName: str):


def birth(self, parentName: str, childName: str) -> None:


def death(self, name: str) -> None:


def getInheritanceOrder(self) -> List[str]:



# Your ThroneInheritance object will be instantiated and called as such:
# obj = ThroneInheritance(kingName)
# obj.birth(parentName,childName)
# obj.death(name)
# param_3 = obj.getInheritanceOrder()
```