Previous Next

582. Kill Process

May 13, 2017 | 18.6K views

Average Rating: 4.58 (12 votes)

Given **n** processes, each process has a unique **PID (process id)** and its **PPID (parent process id)**.

Each process only has one parent process, but may have one or more children processes. This is just like a tree structure. Only one process has PPID that is 0, which means this process has no parent process. All the PIDs will be distinct positive integers.

We use two list of integers to represent a list of processes, where the first list contains PID for each process and the second list contains the corresponding PPID.

Now given the two lists, and a PID representing a process you want to kill, return a list of PIDs of processes that will be killed in the end. You should assume that when a process is killed, all its children processes will be killed. No order is required for the final answer.

Example 1:

```
Input:
pid =  [1, 3, 10, 5]
ppid = [3, 0, 5, 3]
kill = 5
Output: [5,10]
Explanation:
           3
         /   \
        1     5
             /
            10
Kill 5 will also kill 10.
```

Note:

1. The given kill id is guaranteed to be one of the given PIDs.
2. n >= 1.

Solution

Approach #1 Depth First Search [Time Limit Exceeded]

Algorithm

Since killing a process leads to killing all its children processes, the simplest solution is to traverse over the $ppid$ array and find out all the children of the process to be killed. Further, for every child chosen to be killed we recursively make call to the `killProcess` function now treating this child as the new parent to be killed. In every such call, we again traverse over the $ppid$ array now considering the id of the child process, and continue in the same fashion. Further, at every step, for every process chosen to be killed, it is added to the list l that needs to be returned at the end.

Java

```java

public class Solution {

    public List < Integer > killProcess(List < Integer > pid, List < Integer > ppid, int kill) {
        List < Integer > l = new ArrayList < > ();
        if (kill == 0)
            return l;
        l.add(kill);
        for (int i = 0; i < ppid.size(); i++)
            if (ppid.get(i) == kill)
                l.addAll(killProcess(pid, ppid, pid.get(i)));
        return l;
    }
}

```

Complexity Analysis

- Time complexity : $O(n^n)$. $O(n^n)$ function calls will be made in the worst case
- Space complexity : $O(n)$. The depth of the recursion tree can go upto n.

Approach #2 Tree Simulation [Accepted]

Algorithm

We can view the given process relationships in the form of a tree. We can construct the tree in such a way that every node stores information about its own value as well as the list of all its direct children nodes. Thus, now, once the tree has been generated, we can simply start off by killing the required node, and recursively killing the children of each node encountered rather than traversing over the whole $ppid$ array for every node as done in the previous approach.

In order to implement this, we've made use of a $Node$ class which represents a node of a tree. Each node represents a process. Thus, every node stores its own value($Node.val$) and the list of all its direct children($Node.children$). We traverse over the whole pid array and create nodes for all of them. Then, we traverse over the $ppid$ array, and make the parent nodes out of them, and at the same time add all their direct children nodes in their $Node.children$ list. In this way, we convert the given process structure into a tree structure.

Now, that we've obtained the tree structure, we can add the node to be killed to the return list l. Now, we can directly obtain all the direct children of this node from the tree, and add its direct children to the return list. For every node added to the return list, we repeat the same process of obtaining the children recursively.

Java

```java
public class Solution {
    class Node {
        int val;
        List < Node > children = new ArrayList < > ();
    }
    public List < Integer > killProcess(List < Integer > pid, List < Integer > ppid, int kill) {
        HashMap < Integer, Node > map = new HashMap < > ();
        for (int id: pid) {
            Node node = new Node();
            node.val = id;
            map.put(id, node);
        }
        for (int i = 0; i < ppid.size(); i++) {
            if (ppid.get(i) > 0) {
                Node par = map.get(ppid.get(i));
                par.children.add(map.get(pid.get(i)));
            }
        }
        List < Integer > l = new ArrayList < > ();
        l.add(kill);
        getAllChildren(map.get(kill), l);
        return l;
    }
    public void getAllChildren(Node pn, List < Integer > l) {
        for (Node n: pn.children) {
            l.add(n.val);
            getAllChildren(n, l);
```

Complexity Analysis

- Time complexity : $O(n)$. We need to traverse over the $ppid$ and pid array of size n once. The `getAllChildren` function also takes atmost n time, since no node can be a child of two nodes.
- Space complexity : $O(n)$. map of size n is used.

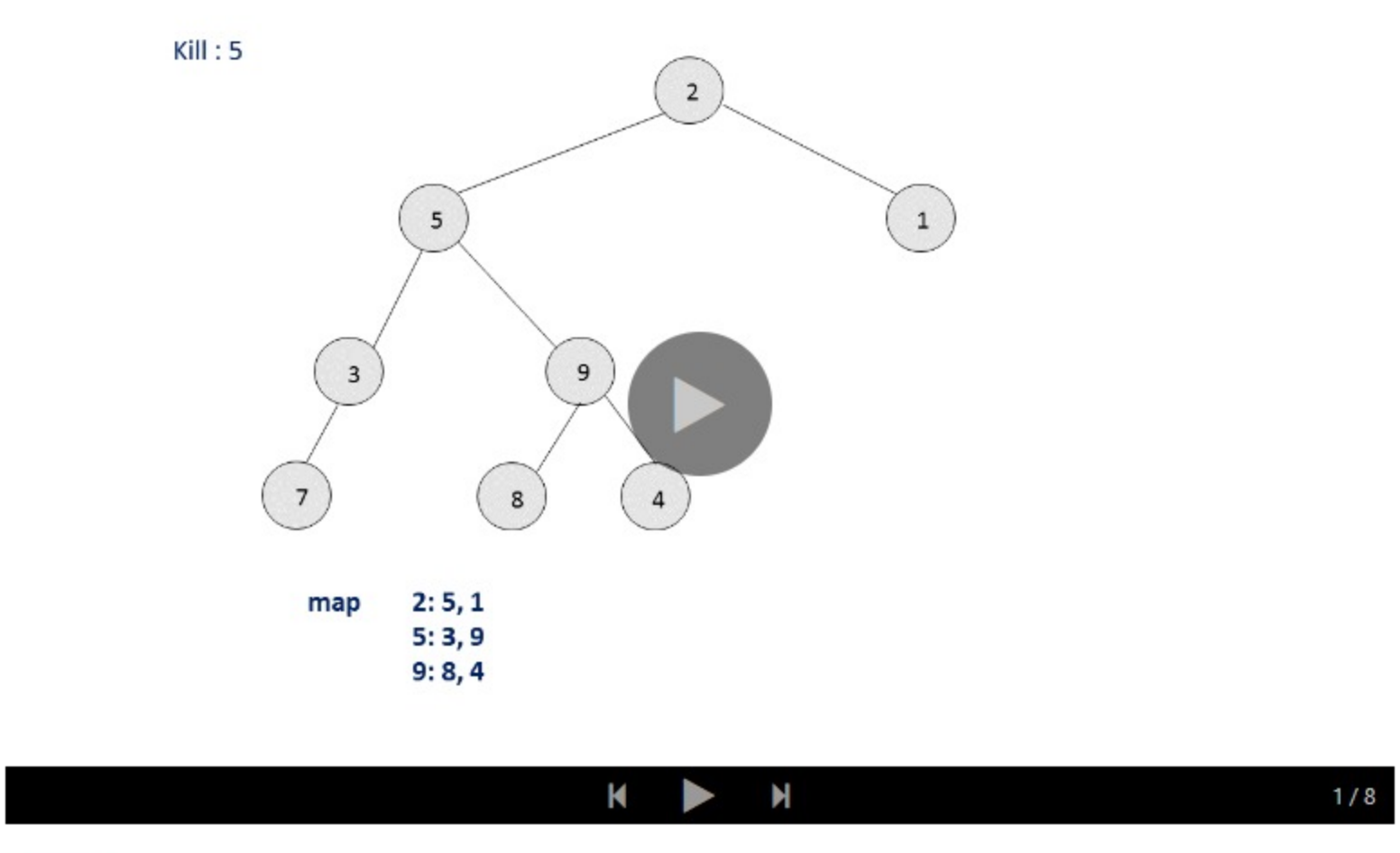
Approach #3 HashMap + Depth First Search [Accepted]

Algorithm

Instead of making the tree structure, we can directly make use of a data structure which stores a particular process value and the list of its direct children. For this, in the current implementation, we make use of a hashmap map, which stores the data in the form $parent : [listofallitsdirectchildren]$.

Thus, now, by traversing just once over the $ppid$ array, and adding the corresponding pid values to the children list at the same time, we can obtain a better structure storing the parent-children relationship.

Again, similar to the previous approach, now we can add the process to be killed to the return list, and keep on adding its children to the return list in a recursive manner by obtaining the child information from the structure created previously.



Java

```java
public class Solution {
    public List < Integer > killProcess(List < Integer > pid, List < Integer > ppid, int kill) {
        HashMap < Integer, List < Integer >> map = new HashMap < > ();
        for (int i = 0; i < ppid.size(); i++) {
            if (ppid.get(i) > 0) {
                List < Integer > l = map.getOrDefault(ppid.get(i), new ArrayList < Integer > ());
                l.add(pid.get(i));
                map.put(ppid.get(i), l);
            }
        }
        List < Integer > l = new ArrayList < > ();
        l.add(kill);
        getAllChildren(map, l, kill);
        return l;
    }
    public void getAllChildren(HashMap < Integer, List < Integer >> map, List < Integer > l, int kill) {
        if (map.containsKey(kill))
            for (int id: map.get(kill)) {
                l.add(id);
                getAllChildren(map, l, id);
            }
    }
}

```

Complexity Analysis

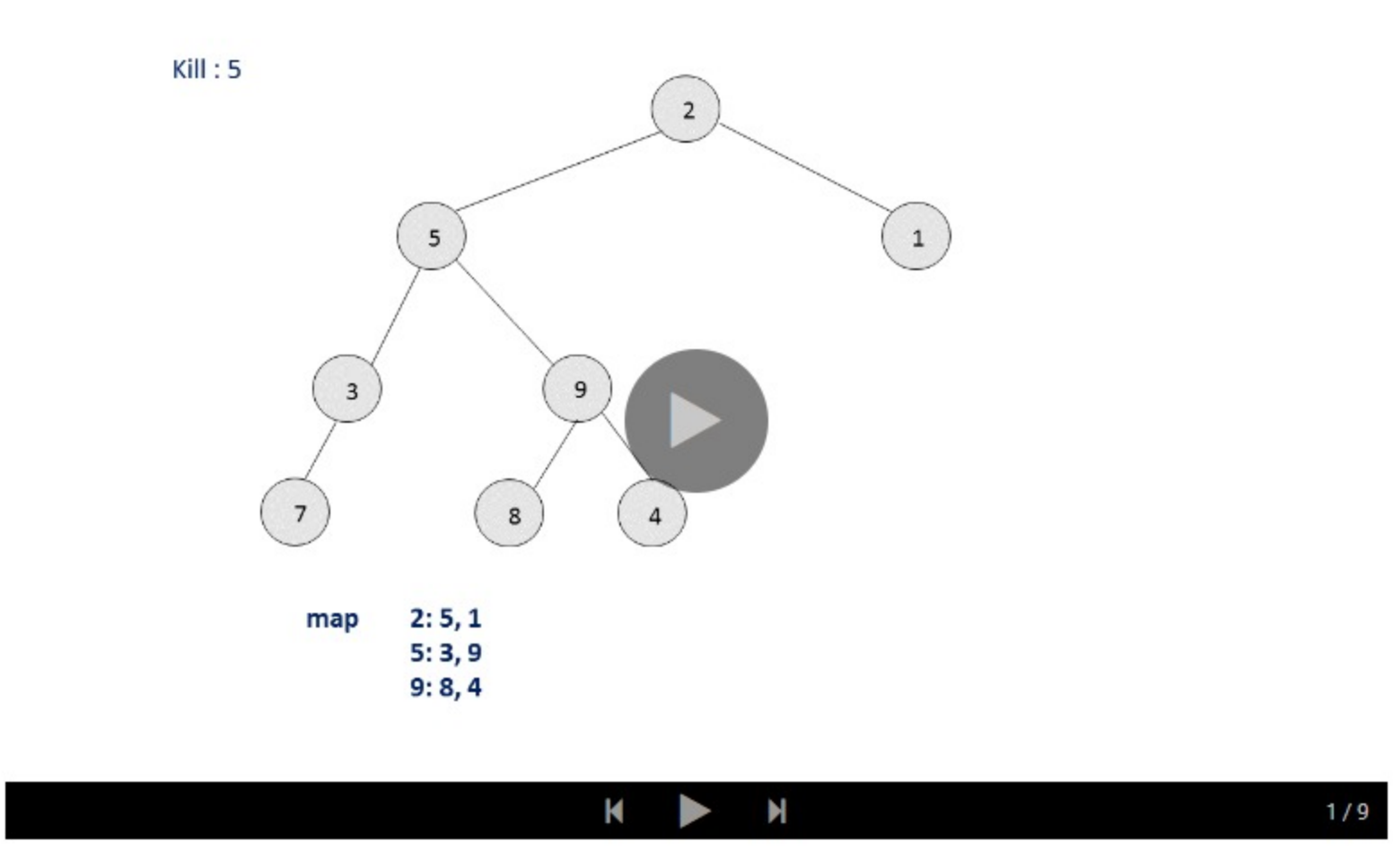
- Time complexity : $O(n)$. We need to traverse over the $ppid$ array of size n once. The `getAllChildren` function also takes atmost n time, since no node can be a child of two nodes.
- Space complexity : $O(n)$. map of size n is used.

Approach #4 HashMap + Breadth First Search [Accepted]:

Algorithm

We can also make use of Breadth First Search to obtain all the children(direct+indirect) of a particular node, once the data structure of the form $(process : [listofallitsdirectchildren]$ has been obtained. The process of obtaining the data structure is the same as in the previous approach.

In order to obtain all the child processes to be killed for a particular parent chosen to be killed, we can make use of Breadth First Search. For this, we add the node to be killed to a $queue$. Then, we remove an element from the front of the $queue$ and add it to the return list. Further, for every element removed from the front of the queue, we add all its direct children(obtained from the data structure created) to the end of the queue. We keep on doing so till the queue becomes empty.



Java

```java
public class Solution {

    public List < Integer > killProcess(List < Integer > pid, List < Integer > ppid, int kill) {
        HashMap < Integer, List < Integer >> map = new HashMap < > ();
        for (int i = 0; i < ppid.size(); i++) {
            if (ppid.get(i) > 0) {
                List < Integer > l = map.getOrDefault(ppid.get(i), new ArrayList < Integer > ());
                l.add(pid.get(i));
                map.put(ppid.get(i), l);
            }
        }
        Queue < Integer > queue = new LinkedList < > ();
        List < Integer > l = new ArrayList < > ();
        queue.add(kill);
        while (!queue.isEmpty()) {
            int r = queue.remove();
            l.add(r);
            if (map.containsKey(r))
                for (int id: map.get(r))
                    queue.add(id);
        }
        return l;
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$. We need to traverse over the $ppid$ array of size n once. Also, atmost n additions/removals are done from the $queue$.
- Space complexity : $O(n)$. map of size n is used.

Analysis written by: @vinod23

Rate this article:

Previous Next

Comments: 10

Sort By

Type comment here... (Markdown is supported)

Preview Post

laotzu ★7 September 14, 2017 11:40 AM

I think the time complexity for first approach is n^2, because there is no repetition nodes which are already explored during the search process.

4 Share Reply

SHOW 2 REPLIES

slz250 ★201 November 5, 2018 4:28 AM

can someone explain why the brute force approach (#1) runtime is O(n^n)?

3 Share Reply

SHOW 3 REPLIES

Mshubhi ★2 July 18, 2018 1:33 AM

```
List < Integer > l = new ArrayList < > ();
        if (kill == 0)
            return l;
        l.add(kill);
```

Read More

1 Share Reply

SHOW 1 REPLY

cleverKnight ★54 August 16, 2017 9:23 AM

I feel like you should also mention that a simple topological sort from the 'kill' node generates the answer - particularly Approach #3 does basically just that.

1 Share Reply

nicksg3395 ★12 September 23, 2019 3:43 PM

Should be an easy problem

1 Share Reply

anupHack ★4 June 20, 2017 6:36 AM

I was also thinking if it can be done like, find the node that needs to be killed. Then mark that with respect to its parent as null, (e.g. in this 5 is the node to be killed so mark 2's left to be null) but save that tree in a temporary node, say temp. After that just traverse using one of the traversal and add the nodes to be printed in an array or list and print them. How about this?

0 Share Reply

vinod23 ★425 June 1, 2017 3:38 PM

@mrvon @aprilyin Sorry there was some problem. We have fixed it. Thanks

0 Share Reply

mrvon ★9 June 1, 2017 1:59 PM

It seems mismatching answer and problem.

0 Share Reply

blairYY ★2 June 1, 2017 6:04 AM

The question is "Now given the two lists, and a PID representing a process you want to kill, return a list of PIDs of processes that will be killed in the end. You should assume that when a process is killed, all its children processes will be killed. No order is required for the final answer." Are you sure the signature "public int findLHS(int[] nums)" is good for this question?

0 Share Reply

abhilash13 ★1 April 3, 2020 4:18 PM

This code doesn't kill pid 0 with children.

0 Share Reply