582. Kill Process

难度 中等 6 收藏 分享 切换为中文 通过次数 53 提交次数 106

关闭 ‹ 上一题解 1 / 4 下一题解 ›

杀死进程

力扣 (LeetCode) 发布于 17 天前 14 阅读 官方题解 Java 哈希表 深度优先搜索

方法 1：深度优先搜索 [Time Limit Exceeded]

算法

由于杀死一个进程会导致杀死它所有孩子进程，最简单的办法就是遍历 $ppid$ 数组并找出所有需要被杀死的孩子进程。对于每个被杀死的孩子进程，我们递归调用 `killProcess` 函数，将当前孩子作为被杀死的父亲进程，以找到该孩子进程的孩子进程。在每一次调用中，我们都遍历 $ppid$ 数组来找到当前进程的子进程。同时，在调用函数的过程中，我们将被杀死的进程放入列表 l 中，并最终返回这个列表。

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;
    }
}
```

复杂度分析

- 时间复杂度：$O(n^n)$。最坏情况下函数调用需要 $O(n^n)$ 的时间。
- 空间复杂度：$O(n)$。递归树的深度最多为 n。

方法 2：模拟树 [Accepted]

算法

我们可以把进程间的关系看做一棵树，我们构建这样一棵树：每个节点保存它自己的值和它直接孩子节点的信息。一旦构建好这棵树，我们可以直接删除要求的节点，然后递归杀死它的子进程，而不需要像前一种方法一样对每个进程都一遍又一遍地遍历 $ppid$ 数组。

为了实现这个方法，我们使用 $Node$ 类来表示树中的一个节点，每个节点表示一个进程。因此每个节点保存着它自己的值 ($Node.val$) 和它所有直接孩子进程的一个列表 ($Node.children$) 。我们遍历 pid 数组并为每一个元素建立一个节点。然后，我们遍历 $ppid$ 数组并将父节点和子节点通过父节点的 $Node.children$ 联系起来。通过这样的过程，我们把给定的进程构建成了一棵树的结构。

现在我们已经有了树结构，我们可以把要删掉的进程添加到 l 列表中去。我们可以直接获得每个节点在树中的的孩子进程，并添加到返回列表中，我们重复此过程，递归地得到所有的孩子进程。

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)));
            }
```

```java
                l.add(n.val);
                getAllChildren(n, l);
            }
        }
}
```

复杂度分析

- 时间复杂度：$O(n)$。我们需要遍历大小为n 的 $ppid$ 和 pid 数组各一遍。由于一个节点只会有一个父节点，所以函数 `getAllChildren` 最多会调用 n 次。
- 空间复杂度：$O(n)$。使用了大小为 n 的 map。

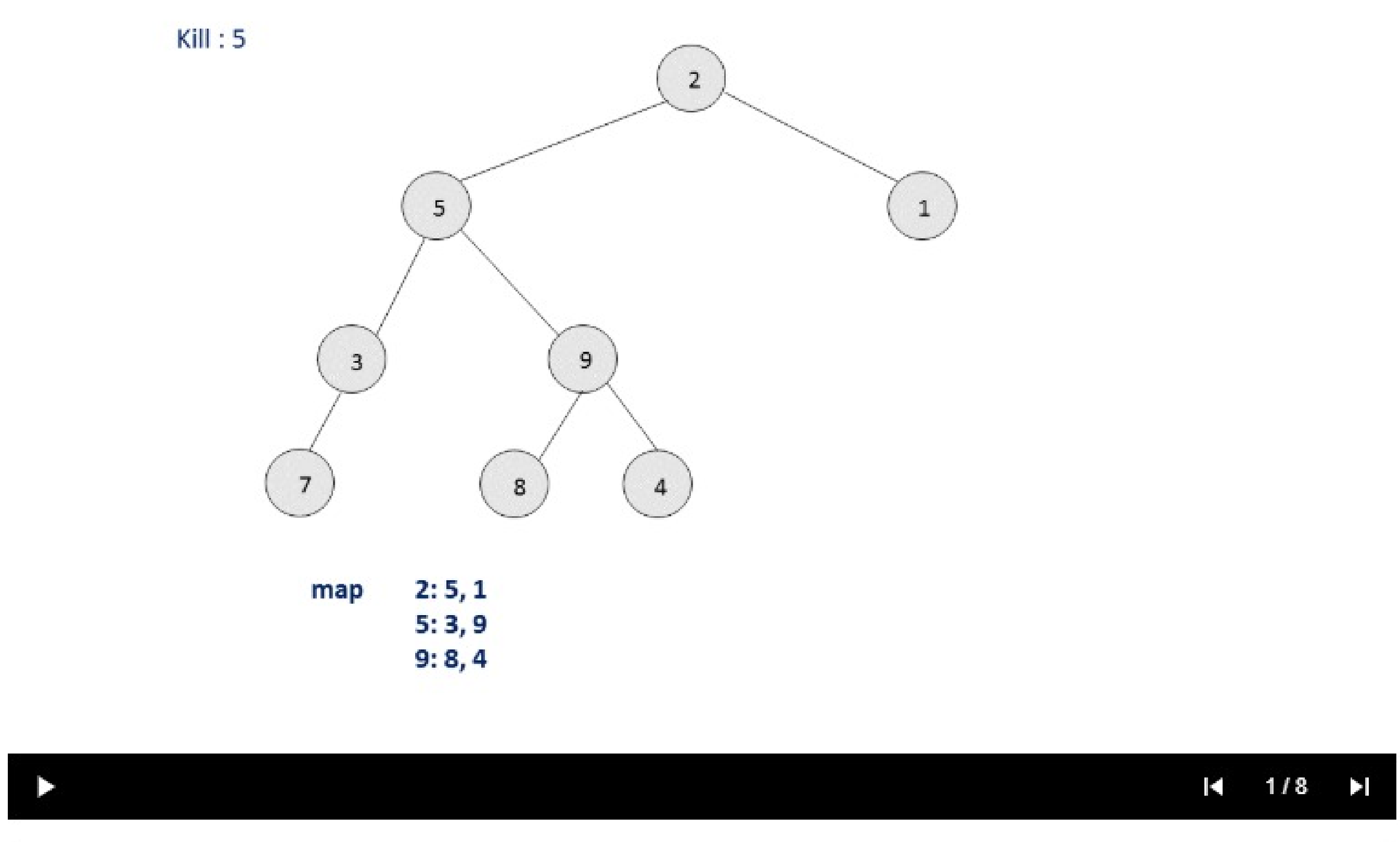
方法 3：哈希表 + 深度优先搜索 [Accepted]

算法

与其构建一棵树的结构，我们可以直接使用一个数据结构，保存特定进程的值和它的子节点。我们使用哈希表 map ，它以 $parent : [listofallitsdirectchildren]$ 的形式保存了数据。

现在遍历一遍 $ppid$ 数组，同时把相应的 pid 值添加到孩子列表，我们可以获得一个更好的保存了 父-子 进程关系的结构。

与前一种方法类似的，我们现在可以把被杀死的进程添加到返回列表中并通过哈希表递归获得孩子节点的信息。



Java

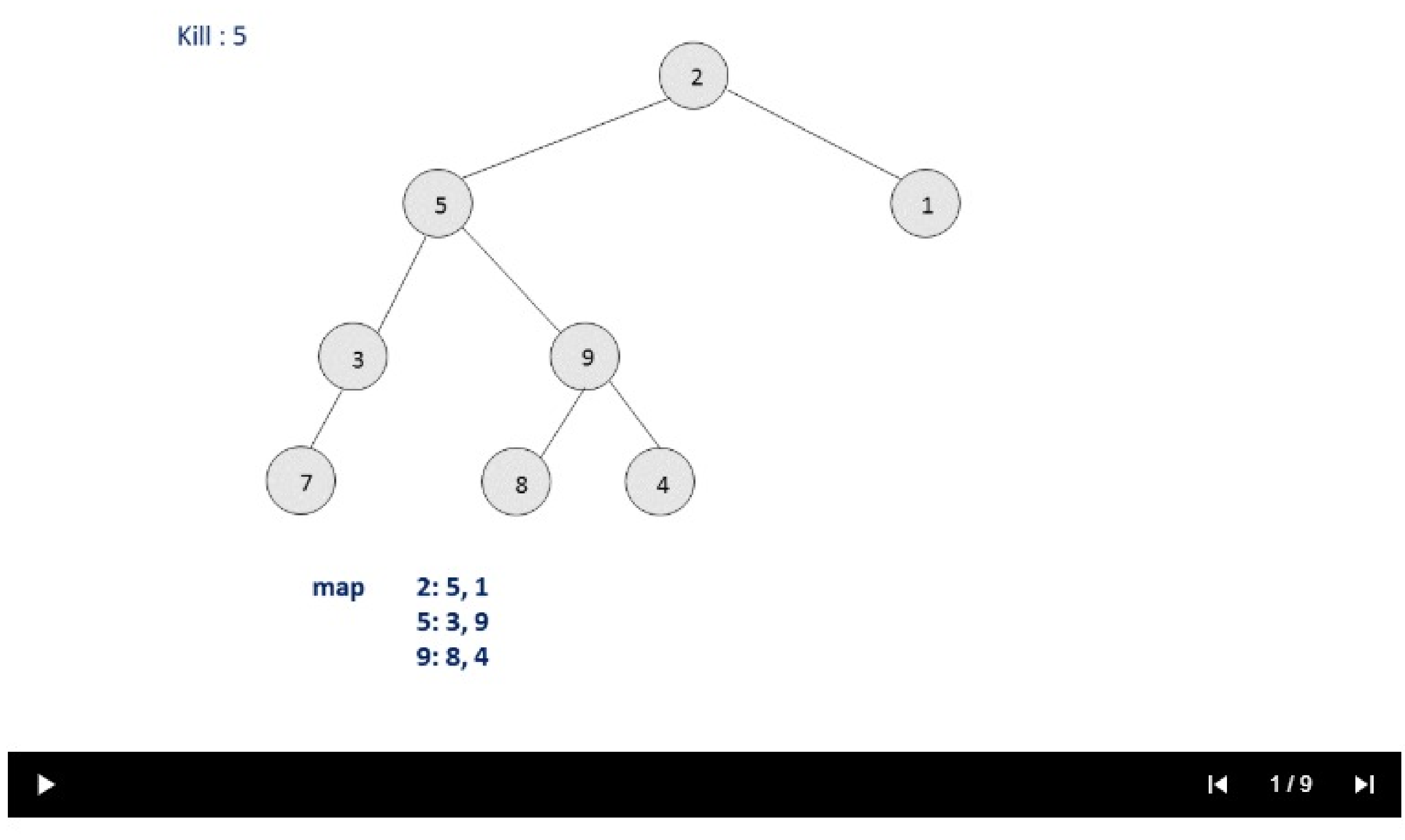
```java
public class Solution {
    public List < Integer > killProcess(List < Integer > pid, List < Integer > ppid, int kill) {
        HashMap < Integer, List < Integer >> map = new HashMap < > ();
        for (int i = 0; i < ppid.size(); i++) {
            if (ppid.get(i) > 0) {
                List < 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);
            }
    }
```

方法 4：哈希表 + 宽度优先搜索 [Accepted]

算法

我们也可以利用宽度优先搜索来获得某个节点的所有孩子信息 （包括直接的和间接的）。首先我们还是按照方法 3 中 $(process : [listofallitsdirectchildren]$ 的结构建立好数据结构。

为了获得某个要被杀死的特定父进程的所有子进程，我们使用宽度优先搜索。我们将要被杀死的进程添加到 $queue$ 中，然后我们从 $queue$ 的头部移出一个元素并把它添加到最终返回列表，然后我们把这个节点的所有直接子进程添加到 $queue$ 中。重复此过程直到 $queue$ 为空。



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;
    }
}
```

复杂度分析

0 条评论 ›_ 最热

编辑 预览 评论

暂无评论