Kill Process

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 1

```
public List<Integer> killProcess(List<Integer> pid, List<Integer> ppid, int kill) {

    // Store process tree as an adjacency list
    Map<Integer, List<Integer>> adjacencyLists = new HashMap<>();
    for (int i=0;i<ppid.size();i++) {
        adjacencyLists.putIfAbsent(ppid.get(i), new LinkedList<>());
        adjacencyLists.get(ppid.get(i)).add(pid.get(i));
    }

    // Kill all processes in the subtree rooted at process "kill"
    List<Integer> res = new LinkedList<>();
    Stack<Integer> stack = new Stack<>();
    stack.add(kill);
    while (!stack.isEmpty()) {
        int cur = stack.pop();
        res.add(cur);
        List<Integer> adjacencyList = adjacencyLists.get(cur);
        if (adjacencyList == null) continue;
        stack.addAll(adjacencyList);
    }
    return res;

}
```

written by compton_scatter original link here

Solution 2

```
public class Solution {
    public List<Integer> killProcess(List<Integer> pid, List<Integer> ppid, int kill
) {
        if (kill == 0) return pid;

        int n = pid.size();
        Map<Integer, Set<Integer>> tree = new HashMap<>();
        for (int i = 0; i < n; i++) {
            tree.put(pid.get(i), new HashSet<Integer>());
        }
        for (int i = 0; i < n; i++) {
            if (tree.containsKey(ppid.get(i))) {
                Set<Integer> children = tree.get(ppid.get(i));
                children.add(pid.get(i));
                tree.put(ppid.get(i), children);
            }
        }

        List<Integer> result = new ArrayList<>();
        traverse(tree, result, kill);

        return result;
    }

    private void traverse(Map<Integer, Set<Integer>> tree, List<Integer> result, int
pid) {
        result.add(pid);

        Set<Integer> children = tree.get(pid);
        for (Integer child : children) {
            traverse(tree, result, child);
        }
    }
}
```

written by shawngao original link here

Solution 3

DFS

```
class Solution {
public:
    vector<int> killProcess(vector<int>& pid, vector<int>& ppid, int kill) {
        vector<int> killed;
        map<int, set<int>> children;
        for (int i = 0; i < pid.size(); i++) {
            children[ppid[i]].insert(pid[i]);
        }
        killAll(kill, children, killed);
        return killed;
    }

private:
    void killAll(int pid, map<int, set<int>>& children, vector<int>& killed) {
        killed.push_back(pid);
        for (int child : children[pid]) {
            killAll(child, children, killed);
        }
    }
};
```

BFS

```
class Solution {
public:
    vector<int> killProcess(vector<int>& pid, vector<int>& ppid, int kill) {
        vector<int> killed;
        map<int, set<int>> children;
        for (int i = 0; i < pid.size(); i++) {
            children[ppid[i]].insert(pid[i]);
        }

        queue<int> q;
        q.push(kill);
        while (!q.empty()) {
            int p = q.front(); q.pop();
            killed.push_back(p);
            for (int child : children[p]) {
                q.push(child);
            }
        }

        return killed;
    }
};
```

written by alexander original link here