Problem 582: Kill Process

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You have

n

processes forming a rooted tree structure. You are given two integer arrays

pid

and

ppid

, where

pid[i]

is the ID of the

i

th

process and

ppid[i]

is the ID of the

i

th

process's parent process.

Each process has only

one parent process

but may have multiple children processes. Only one process has

ppid[i] = 0

, which means this process has

no parent process

(the root of the tree).

When a process is

killed

, all of its children processes will also be killed.

Given an integer

kill

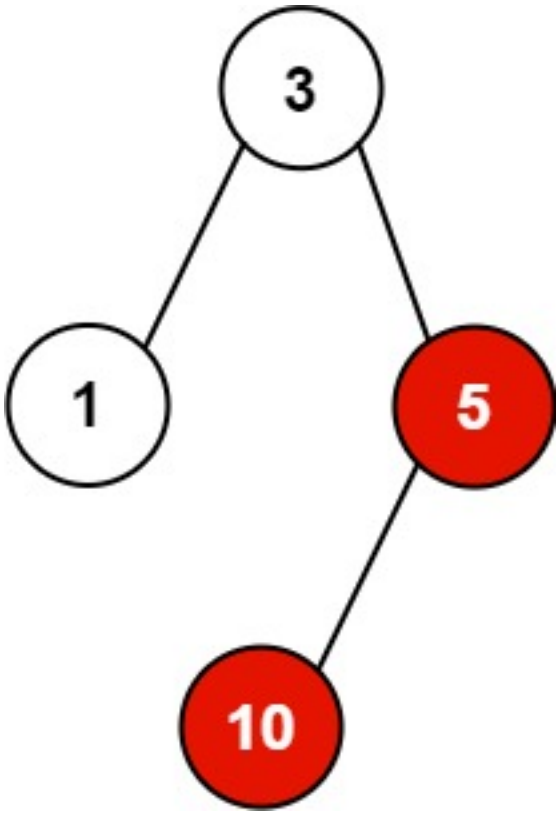
representing the ID of a process you want to kill, return

a list of the IDs of the processes that will be killed. You may return the answer in

any order

.

Example 1:



Input:

pid = [1,3,10,5], ppid = [3,0,5,3], kill = 5

Output:

[5,10]

Explanation:

The processes colored in red are the processes that should be killed.

Example 2:

Input:

pid = [1], ppid = [0], kill = 1

Output:

[1]

Constraints:

n == pid.length

n == ppid.length

1 <= n <= 5 * 10

4

1 <= pid[i] <= 5 * 10

4

0 <= ppid[i] <= 5 * 10

4

Only one process has no parent.

All the values of

pid

are

unique

.

kill

is

guaranteed

to be in

pid

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> killProcess(vector<int>& pid, vector<int>& ppid, int kill) {

}
};
```

Java:

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

}
}
```

Python3:

```
class Solution:
def killProcess(self, pid: List[int], ppid: List[int], kill: int) ->
List[int]:
```

Python:

```
class Solution(object):
def killProcess(self, pid, ppid, kill):
"""
:type pid: List[int]
:type ppid: List[int]
:type kill: int
```

```
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} pid
* @param {number[]} ppid
* @param {number} kill
* @return {number[]}
*/
var killProcess = function(pid, ppid, kill) {

};
```

TypeScript:

```
function killProcess(pid: number[], ppid: number[], kill: number): number[] {

};
```

C#:

```
public class Solution {
public IList<int> KillProcess(IList<int> pid, IList<int> ppid, int kill) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* killProcess(int* pid, int pidSize, int* ppid, int ppidSize, int kill,
int* returnSize) {

}
```

Go:

```
func killProcess(pid []int, ppid []int, kill int) []int {

}
```

Kotlin:

```
class Solution {
fun killProcess(pid: List<Int>, ppid: List<Int>, kill: Int): List<Int> {

}
}
```

Swift:

```
class Solution {
func killProcess(_ pid: [Int], _ ppid: [Int], _ kill: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn kill_process(pid: Vec<i32>, ppid: Vec<i32>, kill: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} pid
# @param {Integer[]} ppid
# @param {Integer} kill
# @return {Integer[]}
def kill_process(pid, ppid, kill)

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer[] $pid
* @param Integer[] $ppid
* @param Integer $kill
* @return Integer[]
*/
function killProcess($pid, $ppid, $kill) {

}
}
```

Dart:

```
class Solution {
List<int> killProcess(List<int> pid, List<int> ppid, int kill) {

}
}
```

Scala:

```
object Solution {
def killProcess(pid: List[Int], ppid: List[Int], kill: Int): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec kill_process(pid :: [integer], ppid :: [integer], kill :: integer) ::
[integer]
def kill_process(pid, ppid, kill) do

end
end
```

Erlang:

```
-spec kill_process(Pid :: [integer()], Ppid :: [integer()], Kill ::
integer()) -> [integer()].
kill_process(Pid, Ppid, Kill) ->
.
```

Racket:

```
(define/contract (kill-process pid ppid kill)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Kill Process
* Difficulty: Medium
* Tags: array, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<int> killProcess(vector<int>& pid, vector<int>& ppid, int kill) {

}
};
```

Java Solution:

```
/**
* Problem: Kill Process
* Difficulty: Medium
* Tags: array, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
```

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

}
}
```

Python3 Solution:

```
"""
Problem: Kill Process
Difficulty: Medium
Tags: array, tree, hash, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def killProcess(self, pid: List[int], ppid: List[int], kill: int) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def killProcess(self, pid, ppid, kill):
"""
:type pid: List[int]
:type ppid: List[int]
:type kill: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Kill Process
* Difficulty: Medium
* Tags: array, tree, hash, search
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[]} pid
* @param {number[]} ppid
* @param {number} kill
* @return {number[]}
*/
var killProcess = function(pid, ppid, kill) {

};
```

TypeScript Solution:

```
/**
* Problem: Kill Process
* Difficulty: Medium
* Tags: array, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function killProcess(pid: number[], ppid: number[], kill: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Kill Process
* Difficulty: Medium
* Tags: array, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public IList<int> KillProcess(IList<int> pid, IList<int> ppid, int kill) {

}
}
```

C Solution:

```
/*
* Problem: Kill Process
* Difficulty: Medium
* Tags: array, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* killProcess(int* pid, int pidSize, int* ppid, int ppidSize, int kill,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Kill Process
// Difficulty: Medium
// Tags: array, tree, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func killProcess(pid []int, ppid []int, kill int) []int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun killProcess(pid: List<Int>, ppid: List<Int>, kill: Int): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func killProcess(_ pid: [Int], _ ppid: [Int], _ kill: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Kill Process
// Difficulty: Medium
// Tags: array, tree, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn kill_process(pid: Vec<i32>, ppid: Vec<i32>, kill: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} pid
# @param {Integer[]} ppid
# @param {Integer} kill
# @return {Integer[]}
def kill_process(pid, ppid, kill)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $pid
* @param Integer[] $ppid
* @param Integer $kill
* @return Integer[]
*/
function killProcess($pid, $ppid, $kill) {

}
}
```

Dart Solution:

```
class Solution {
List<int> killProcess(List<int> pid, List<int> ppid, int kill) {

}
}
```

Scala Solution:

```
object Solution {
def killProcess(pid: List[Int], ppid: List[Int], kill: Int): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec kill_process(pid :: [integer], ppid :: [integer], kill :: integer) ::
[integer]
def kill_process(pid, ppid, kill) do
```

```
end
end
```

Erlang Solution:

```
-spec kill_process(Pid :: [integer()], Ppid :: [integer()], Kill ::
integer()) -> [integer()].
kill_process(Pid, Ppid, Kill) ->
.
```

Racket Solution:

```
(define/contract (kill-process pid ppid kill)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer? (listof
exact-integer?))
)
```