Problem 1110: Delete Nodes And Return Forest

Problem Information

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

Problem Description

Given the

root

of a binary tree, each node in the tree has a distinct value.

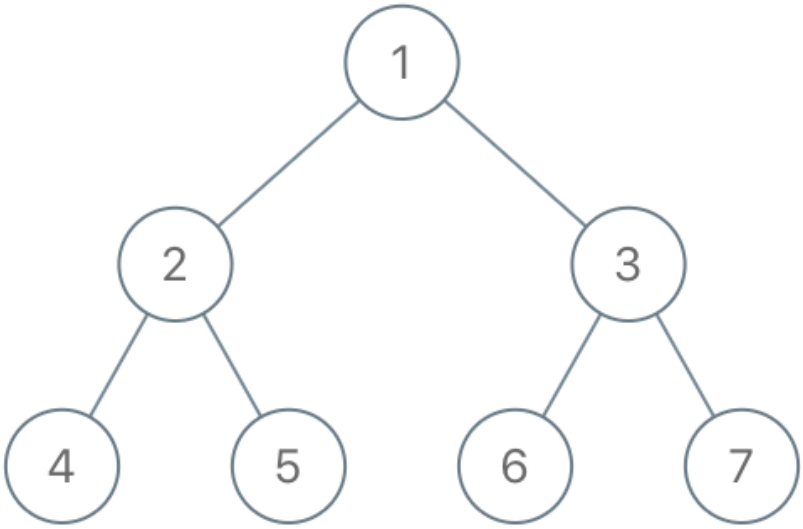
After deleting all nodes with a value in

to_delete

, we are left with a forest (a disjoint union of trees).

Return the roots of the trees in the remaining forest. You may return the result in any order.

Example 1:



Input:

root = [1,2,3,4,5,6,7], to_delete = [3,5]

Output:

[[1,2,null,4],[6],[7]]

Example 2:

Input:

root = [1,2,4,null,3], to_delete = [3]

Output:

[[1,2,4]]

Constraints:

The number of nodes in the given tree is at most

1000

.

Each node has a distinct value between

1

and

1000

.

to_delete.length <= 1000

to_delete

contains distinct values between

1

and

1000

.

Code Snippets

C++:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode() : val(0), left(nullptr), right(nullptr) {}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
```

```
* };
*/
class Solution {
public:
vector<TreeNode*> delNodes(TreeNode* root, vector<int>& to_delete) {

}
};
```

Java:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode() {}
* TreeNode(int val) { this.val = val; }
* TreeNode(int val, TreeNode left, TreeNode right) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public List<TreeNode> delNodes(TreeNode root, int[] to_delete) {

}
}
```

Python3:

```
# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution:
def delNodes(self, root: Optional[TreeNode], to_delete: List[int]) ->
```

```
List[TreeNode]:
```

Python:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution(object):
def delNodes(self, root, to_delete):
"""
:type root: TreeNode
:type to_delete: List[int]
:rtype: List[TreeNode]
"""
```

JavaScript:

```
/**
* Definition for a binary tree node.
* function TreeNode(val, left, right) {
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
*/
/**
* @param {TreeNode} root
* @param {number[]} to_delete
* @return {TreeNode[]}
*/
var delNodes = function(root, to_delete) {

};
```

TypeScript:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* val: number
```

```
* left: TreeNode | null
* right: TreeNode | null
* constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
{
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
* }
*/

function delNodes(root: TreeNode | null, to_delete: number[]): Array<TreeNode
| null> {

};
```

C#:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public int val;
* public TreeNode left;
* public TreeNode right;
* public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
public class Solution {
public IList<TreeNode> DelNodes(TreeNode root, int[] to_delete) {

}
}
```

C:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
```

```
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/


/**
* Note: The returned array must be malloced, assume caller calls free().
*/
struct TreeNode** delNodes(struct TreeNode* root, int* to_delete, int
to_deleteSize, int* returnSize){

}
```

Go:

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func delNodes(root *TreeNode, to_delete []int) []*TreeNode {

}
```

Kotlin:

```
/**
* Example:
* var ti = TreeNode(5)
* var v = ti.`val`
* Definition for a binary tree node.
* class TreeNode(var `val`: Int) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/
class Solution {
```

```
fun delNodes(root: TreeNode?, to_delete: IntArray): List<TreeNode?> {

}
}
```

Swift:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public var val: Int
* public var left: TreeNode?
* public var right: TreeNode?
* public init() { self.val = 0; self.left = nil; self.right = nil; }
* public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/
class Solution {
func delNodes(_ root: TreeNode?, _ to_delete: [Int]) -> [TreeNode?] {

}
}
```

Rust:

```
// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
// pub val: i32,
// pub left: Option<Rc<RefCell<TreeNode>>>,
// pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
// #[inline]
// pub fn new(val: i32) -> Self {
```

```
// TreeNode {
// val,
// left: None,
// right: None
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn del_nodes(root: Option<Rc<RefCell<TreeNode>>>, to_delete: Vec<i32>) ->
Vec<Option<Rc<RefCell<TreeNode>>>> {

}
}
```

Ruby:

```
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
# @val = val
# @left = left
# @right = right
# end
# end
# @param {TreeNode} root
# @param {Integer[]} to_delete
# @return {TreeNode[]}
def del_nodes(root, to_delete)

end
```

PHP:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* public $val = null;
* public $left = null;
* public $right = null;
```

```
* function __construct($val = 0, $left = null, $right = null) {
* $this->val = $val;
* $this->left = $left;
* $this->right = $right;
* }
* }
*/
class Solution {

/**
* @param TreeNode $root
* @param Integer[] $to_delete
* @return TreeNode[]
*/
function delNodes($root, $to_delete) {

}
}
```

Scala:

```
/**
* Definition for a binary tree node.
* class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
* var value: Int = _value
* var left: TreeNode = _left
* var right: TreeNode = _right
* }
*/
object Solution {
def delNodes(root: TreeNode, to_delete: Array[Int]): List[TreeNode] = {

}
}
```

Racket:

```
; Definition for a binary tree node.
#|

; val : integer?
```

```
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))

|#

(define/contract (del-nodes root to_delete)
(-> (or/c tree-node? #f) (listof exact-integer?) (listof (or/c tree-node?
#f)))

)
```

Solutions

C++ Solution:

```
/*
* Problem: Delete Nodes And Return Forest
* 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
*/

/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode() : val(0), left(nullptr), right(nullptr) {}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
```

```
right(right) {}
* };
*/
class Solution {
public:
vector<TreeNode*> delNodes(TreeNode* root, vector<int>& to_delete) {

}
};
```

Java Solution:

```
/**
* Problem: Delete Nodes And Return Forest
* 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
*/

/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode() {}
* TreeNode(int val) { this.val = val; }
* TreeNode(int val, TreeNode left, TreeNode right) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public List<TreeNode> delNodes(TreeNode root, int[] to_delete) {

}
}
```

Python3 Solution:

```
"""
Problem: Delete Nodes And Return Forest
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
"""

# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution:
def delNodes(self, root: Optional[TreeNode], to_delete: List[int]) ->
List[TreeNode]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution(object):
def delNodes(self, root, to_delete):
"""
:type root: TreeNode
:type to_delete: List[int]
:rtype: List[TreeNode]
"""
```

JavaScript Solution:

```
/**
* Problem: Delete Nodes And Return Forest
* 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
*/

/**
* Definition for a binary tree node.
* function TreeNode(val, left, right) {
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
*/
/**
* @param {TreeNode} root
* @param {number[]} to_delete
* @return {TreeNode[]}
*/
var delNodes = function(root, to_delete) {

};
```

TypeScript Solution:

```
/**
* Problem: Delete Nodes And Return Forest
* 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
*/

/**
* Definition for a binary tree node.
* class TreeNode {
```

```
* val: number
* left: TreeNode | null
* right: TreeNode | null
* constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
{
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
* }
*/

function delNodes(root: TreeNode | null, to_delete: number[]): Array<TreeNode
| null> {

};
```

C# Solution:

```
/*
* Problem: Delete Nodes And Return Forest
* 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
*/

/**
* Definition for a binary tree node.
* public class TreeNode {
* public int val;
* public TreeNode left;
* public TreeNode right;
* public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
```

```
*/
public class Solution {
public IList<TreeNode> DelNodes(TreeNode root, int[] to_delete) {

}
}
```

C Solution:

```
/*
* Problem: Delete Nodes And Return Forest
* 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
*/

/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/



/**
* Note: The returned array must be malloced, assume caller calls free().
*/
struct TreeNode** delNodes(struct TreeNode* root, int* to_delete, int
to_deleteSize, int* returnSize){

}
```

Go Solution:

```
// Problem: Delete Nodes And Return Forest
// 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

/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func delNodes(root *TreeNode, to_delete []int) []*TreeNode {

}
```

Kotlin Solution:

```
/**
* Example:
* var ti = TreeNode(5)
* var v = ti.`val`
* Definition for a binary tree node.
* class TreeNode(var `val`: Int) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/
class Solution {
fun delNodes(root: TreeNode?, to_delete: IntArray): List<TreeNode?> {

}
}
```

Swift Solution:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
```

```
* public var val: Int
* public var left: TreeNode?
* public var right: TreeNode?
* public init() { self.val = 0; self.left = nil; self.right = nil; }
* public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/
class Solution {
func delNodes(_ root: TreeNode?, _ to_delete: [Int]) -> [TreeNode?] {

}
}
```

Rust Solution:

```
// Problem: Delete Nodes And Return Forest
// 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

// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
// pub val: i32,
// pub left: Option<Rc<RefCell<TreeNode>>>,
// pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
// #[inline]
// pub fn new(val: i32) -> Self {
// TreeNode {
```

```
// val,
// left: None,
// right: None
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn del_nodes(root: Option<Rc<RefCell<TreeNode>>>, to_delete: Vec<i32>) ->
Vec<Option<Rc<RefCell<TreeNode>>>> {

}
}
```

Ruby Solution:

```
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
# @val = val
# @left = left
# @right = right
# end
# end
# @param {TreeNode} root
# @param {Integer[]} to_delete
# @return {TreeNode[]}
def del_nodes(root, to_delete)

end
```

PHP Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* public $val = null;
* public $left = null;
* public $right = null;
```

```
* function __construct($val = 0, $left = null, $right = null) {
* $this->val = $val;
* $this->left = $left;
* $this->right = $right;
* }
* }
*/
class Solution {

/**
* @param TreeNode $root
* @param Integer[] $to_delete
* @return TreeNode[]
*/
function delNodes($root, $to_delete) {

}
}
```

Scala Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
* var value: Int = _value
* var left: TreeNode = _left
* var right: TreeNode = _right
* }
*/
object Solution {
def delNodes(root: TreeNode, to_delete: Array[Int]): List[TreeNode] = {

}
}
```

Racket Solution:

```
; Definition for a binary tree node.
#|
```

```
; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))

|#

(define/contract (del-nodes root to_delete)
(-> (or/c tree-node? #f) (listof exact-integer?) (listof (or/c tree-node?
#f)))

)
```