Problem 863: All Nodes Distance K in Binary Tree

Problem Information

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

Problem Description

Given the

root

of a binary tree, the value of a target node

target

, and an integer

k

, return

an array of the values of all nodes that have a distance

k

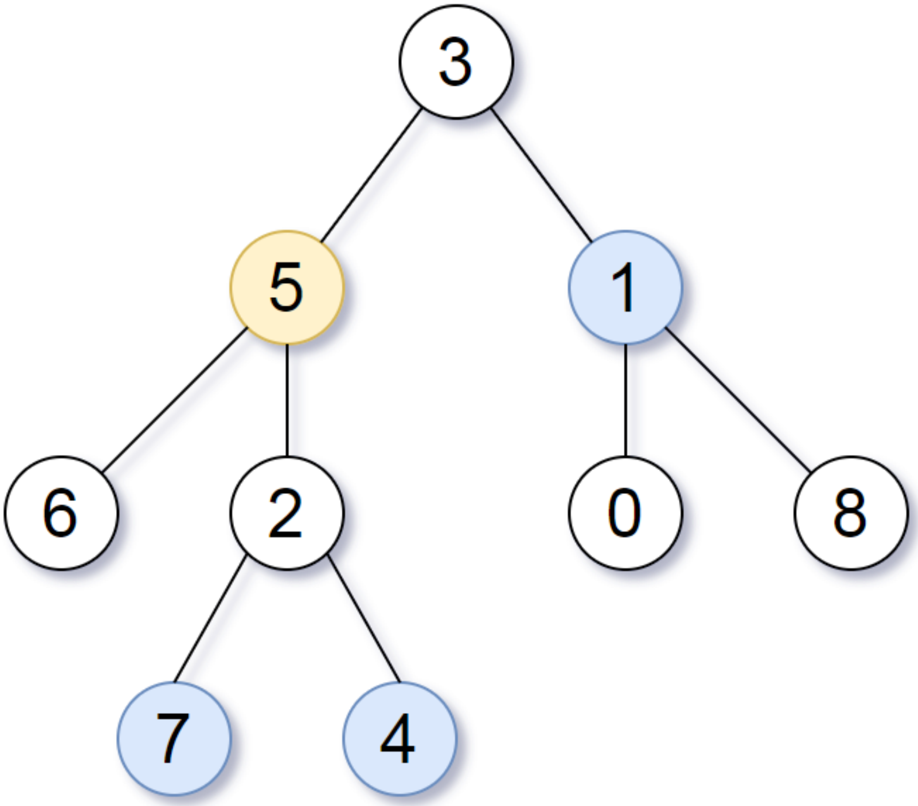
from the target node.

You can return the answer in

any order

.

Example 1:



Input:

root = [3,5,1,6,2,0,8,null,null,7,4], target = 5, k = 2

Output:

[7,4,1] Explanation: The nodes that are a distance 2 from the target node (with value 5) have values 7, 4, and 1.

Example 2:

Input:

root = [1], target = 1, k = 3

Output:

[]

Constraints:

The number of nodes in the tree is in the range

[1, 500]

.

0 <= Node.val <= 500

All the values

Node.val

are

unique

.

target

is the value of one of the nodes in the tree.

0 <= k <= 1000

Code Snippets

C++:

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

```
*/
class Solution {
public:
vector<int> distanceK(TreeNode* root, TreeNode* target, int k) {

}
};
```

Java:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode(int x) { val = x; }
* }
*/
class Solution {
public List<Integer> distanceK(TreeNode root, TreeNode target, int k) {

}
}
```

Python3:

```
# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, x):
# self.val = x
# self.left = None
# self.right = None

class Solution:
def distanceK(self, root: TreeNode, target: TreeNode, k: int) -> List[int]:
```

Python:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, x):
```

```
# self.val = x
# self.left = None
# self.right = None

class Solution(object):
def distanceK(self, root, target, k):
"""
:type root: TreeNode
:type target: TreeNode
:type k: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* Definition for a binary tree node.
* function TreeNode(val) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} root
* @param {TreeNode} target
* @param {number} k
* @return {number[]}
*/
var distanceK = function(root, target, k) {

};
```

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 distanceK(root: TreeNode | null, target: TreeNode | null, k:
number): number[] {

};
```

C#:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public int val;
* public TreeNode left;
* public TreeNode right;
* public TreeNode(int x) { val = x; }
* }
*/
public class Solution {
public IList<int> DistanceK(TreeNode root, TreeNode target, int k) {

}
}
```

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().
*/
int* distanceK(struct TreeNode* root, struct TreeNode* target, int k, int*
returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
/**
* Definition for a binary tree node.
* class TreeNode(var `val`: Int = 0) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/
class Solution {
fun distanceK(root: TreeNode?, target: TreeNode?, k: Int): List<Int> {

}
}
```

Swift:

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

}
}
```

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 distance_k(root: Option<Rc<RefCell<TreeNode>>>, target:
Option<Rc<RefCell<TreeNode>>>, k: i32) -> Vec<i32> {

}
}
```

Ruby:

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

end
```

PHP:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* public $val = null;
* public $left = null;
* public $right = null;
* function __construct($value) { $this->val = $value; }
* }
*/
class Solution {
/**
* @param TreeNode $root
* @param TreeNode $target
* @param Integer $k
* @return Integer[]
*/
function distanceK($root, $target, $k) {

}
}
```

Scala:

```
/**
* Definition for a binary tree node.
* class TreeNode(var _value: Int) {
* var value: Int = _value
* var left: TreeNode = null
* var right: TreeNode = null
* }
*/
object Solution {
def distanceK(root: TreeNode, target: TreeNode, k: Int): List[Int] = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: All Nodes Distance K in Binary Tree
* 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(int x) : val(x), left(NULL), right(NULL) {}
* };
*/
class Solution {
public:
```

```
vector<int> distanceK(TreeNode* root, TreeNode* target, int k) {

}
};
```

Java Solution:

```
/**
* Problem: All Nodes Distance K in Binary Tree
* 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(int x) { val = x; }
* }
*/
class Solution {
public List<Integer> distanceK(TreeNode root, TreeNode target, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: All Nodes Distance K in Binary Tree
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, x):
# self.val = x
# self.left = None
# self.right = None

class Solution:
def distanceK(self, root: TreeNode, target: TreeNode, k: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, x):
# self.val = x
# self.left = None
# self.right = None

class Solution(object):
def distanceK(self, root, target, k):
"""
:type root: TreeNode
:type target: TreeNode
:type k: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: All Nodes Distance K in Binary Tree
* 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) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} root
* @param {TreeNode} target
* @param {number} k
* @return {number[]}
*/
var distanceK = function(root, target, k) {

};
```

TypeScript Solution:

```
/**
* Problem: All Nodes Distance K in Binary Tree
* 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 distanceK(root: TreeNode | null, target: TreeNode | null, k:
number): number[] {

};
```

C# Solution:

```
/*
* Problem: All Nodes Distance K in Binary Tree
* 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 x) { val = x; }
* }
*/
public class Solution {
public IList<int> DistanceK(TreeNode root, TreeNode target, int k) {

}
}
```

C Solution:

```
/*
* Problem: All Nodes Distance K in Binary Tree
* 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().
*/
int* distanceK(struct TreeNode* root, struct TreeNode* target, int k, int*
returnSize) {

}
```

Go Solution:

```
// Problem: All Nodes Distance K in Binary Tree
// 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 distanceK(root *TreeNode, target *TreeNode, k int) []int {

}
```

Kotlin Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode(var `val`: Int = 0) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/
class Solution {
fun distanceK(root: TreeNode?, target: TreeNode?, k: Int): List<Int> {

}
}
```

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(_ val: Int) {
* self.val = val
* self.left = nil
* self.right = nil
* }
* }
*/
class Solution {
func distanceK(_ root: TreeNode?, _ target: TreeNode?, _ k: Int) -> [Int] {
}
}
```

Rust Solution:

```
// Problem: All Nodes Distance K in Binary Tree
// 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 distance_k(root: Option<Rc<RefCell<TreeNode>>>, target:
Option<Rc<RefCell<TreeNode>>>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* public $val = null;
* public $left = null;
* public $right = null;
* function __construct($value) { $this->val = $value; }
* }
*/
class Solution {
/**
* @param TreeNode $root
* @param TreeNode $target
* @param Integer $k
* @return Integer[]
*/
function distanceK($root, $target, $k) {

}
}
```

Scala Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode(var _value: Int) {
* var value: Int = _value
* var left: TreeNode = null
* var right: TreeNode = null
* }
*/
object Solution {
def distanceK(root: TreeNode, target: TreeNode, k: Int): List[Int] = {

}
}
```