Problem 2689: Extract Kth Character From The Rope Tree

Problem Information

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

Problem Description

You are given the

root

of a binary tree and an integer

k

. Besides the left and right children, every node of this tree has two other properties, a

string

node.val

containing only lowercase English letters (possibly empty) and a non-negative integer

node.len

. There are two types of nodes in this tree:

Leaf

: These nodes have no children,

node.len = 0

, and

node.val

is some

non-empty

string.

Internal

: These nodes have at least one child (also at most two children),

node.len > 0

, and

node.val

is an

empty

string.

The tree described above is called a

Rope

binary tree. Now we define

S[node]

recursively as follows:

If

node

is some leaf node,

S[node] = node.val

,

Otherwise if

node

is some internal node,

S[node] = concat(S[node.left], S[node.right])

and

S[node].length = node.len

.

Return

k-th character of the string

S[root]

.

Note:

If

s

and

p

are two strings,

concat(s, p)

is a string obtained by concatenating

p

to

s

. For example,

concat("ab", "zz") = "abzz"

.

Example 1:

Input:

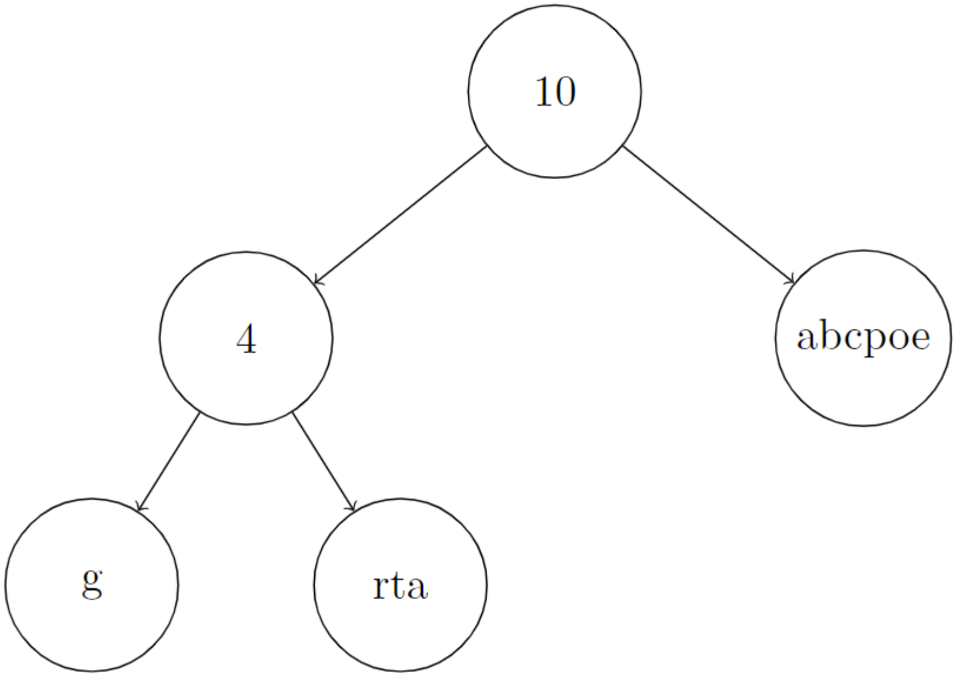
root = [10,4,"abcpoe","g","rta"], k = 6

Output:

"b"

Explanation:

In the picture below, we put an integer on internal nodes that represents node.len, and a string on leaf nodes that represents node.val. You can see that S[root] = concat(concat("g", "rta"), "abcpoe") = "grtaabcpoe". So S[root][5], which represents 6th character of it, is equal to "b".



Example 2:

Input:

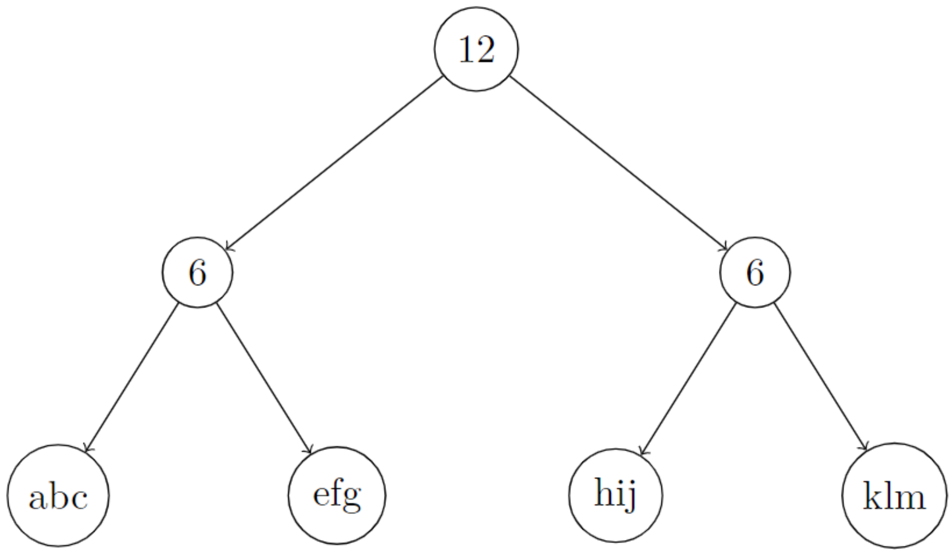
root = [12,6,6,"abc","efg","hij","klm"], k = 3

Output:

"c"

Explanation:

In the picture below, we put an integer on internal nodes that represents node.len, and a string on leaf nodes that represents node.val. You can see that S[root] = concat(concat("abc", "efg"), concat("hij", "klm")) = "abcefghijklm". So S[root][2], which represents the 3rd character of it, is equal to "c".



Example 3:

Input:

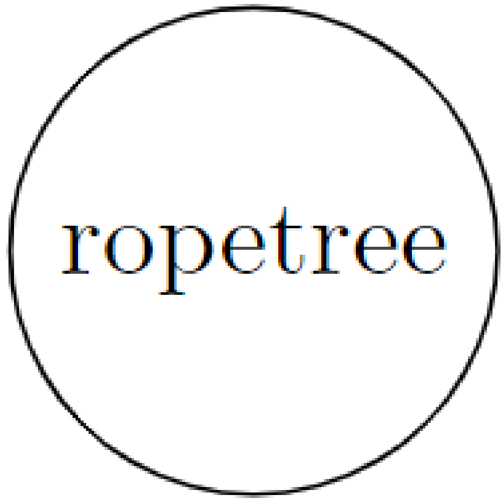
root = ["ropetree"], k = 8

Output:

"e"

Explanation:

In the picture below, we put an integer on internal nodes that represents node.len, and a string on leaf nodes that represents node.val. You can see that S[root] = "ropetree". So S[root][7], which represents 8th character of it, is equal to "e".



Constraints:

The number of nodes in the tree is in the range

[1, 10

3

]

node.val

contains only lowercase English letters

0 <= node.val.length <= 50

0 <= node.len <= 10

4

for leaf nodes,

node.len = 0

and

node.val

is non-empty

for internal nodes,

node.len > 0

and

node.val

is empty

1 <= k <= S[root].length

Code Snippets

C++:

```
/**
* Definition for a rope tree node.
* struct RopeTreeNode {
* int len;
* string val;
* RopeTreeNode *left;
* RopeTreeNode *right;
* RopeTreeNode() : len(0), val(""), left(nullptr), right(nullptr) {}
* RopeTreeNode(string s) : len(0), val(std::move(s)), left(nullptr),
right(nullptr) {}
* RopeTreeNode(int x) : len(x), val(""), left(nullptr), right(nullptr) {}
* RopeTreeNode(int x, RopeTreeNode *left, RopeTreeNode *right) : len(x),
val(""), left(left), right(right) {}
* };
*/
class Solution {
public:
char getKthCharacter(RopeTreeNode* root, int k) {
```

```
}
};
```

Java:

```
/**
* Definition for a rope tree node.
* class RopeTreeNode {
* int len;
* String val;
* RopeTreeNode left;
* RopeTreeNode right;
* RopeTreeNode() {}
* RopeTreeNode(String val) {
* this.len = 0;
* this.val = val;
* }
* RopeTreeNode(int len) {
* this.len = len;
* this.val = "";
* }
* RopeTreeNode(int len, RopeTreeNode left, RopeTreeNode right) {
* this.len = len;
* this.val = "";
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public char getKthCharacter(RopeTreeNode root, int k) {

}
}
```

Python3:

```
# Definition for a rope tree node.
# class RopeTreeNode(object):
# def __init__(self, len=0, val="", left=None, right=None):
# self.len = len
# self.val = val
```

```
# self.left = left
# self.right = right
class Solution:
def getKthCharacter(self, root: Optional[object], k: int) -> str:
"""
:type root: Optional[RopeTreeNode]
"""
```

Python:

```
# Definition for a rope tree node.
# class RopeTreeNode(object):
# def __init__(self, len=0, val="", left=None, right=None):
# self.len = len
# self.val = val
# self.left = left
# self.right = right
class Solution(object):
def getKthCharacter(self, root, k):
"""
:type root: Optional[RopeTreeNode]
:type k: int
:rtype: str
"""
```

JavaScript:

```
/**
* Definition for a rope tree node.
* class RopeTreeNode {
* constructor(len, val, left, right) {
* this.len = (len===undefined ? 0 : len);
* this.val = (val===undefined ? "" : val);
* this.left = (left===undefined ? null : left);
* this.right = (right===undefined ? null : right);
* }
* }
*/
/**
* @param {RopeTreeNode} root
* @param {number} k
* @return {character}
```

```
*/
var getKthCharacter = function(root, k) {

};
```

C#:

```
/**
* Definition for a rope tree node.
* public class RopeTreeNode {
* public int len;
* public string val;
* public RopeTreeNode left;
* public RopeTreeNode right;
* public RopeTreeNode(int len=0, string val="", RopeTreeNode left=null,
RopeTreeNode right=null) {
* this.len = len;
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
public class Solution {
public char GetKthCharacter(RopeTreeNode root, int k) {

}
}
```

C:

```
/**
* Definition for a rope tree node. */
struct RopeTreeNode {
int len;
char* val;
struct RopeTreeNode* left;
struct RopeTreeNode* right;
};

/// DO NOT MODIFY THE CODE ABOVE
```

```
char getKthCharacter(struct RopeTreeNode* root, int k){

}
```

Go:

```
/**
* Definition for a rope tree node.
* type RopeTreeNode struct {
* len int
* val string
* left *RopeTreeNode
* right *RopeTreeNode
* }
*/
func getKthCharacter(root *TreeNode, k int) byte {

}
```

Swift:

```
/**
* Definition for a rope tree node.
* public class RopeTreeNode {
* var len: Int
* var val: String
* var left: RopeTreeNode?
* var right: RopeTreeNode?
* init(len: Int = 0, val: String = "", left: RopeTreeNode? = nil, right:
RopeTreeNode? = nil) {
* self.len = len
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/
class Solution {
func getKthCharacter(_ root: RopeTreeNode?, _ k: Int) -> Character {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Extract Kth Character From The Rope Tree
* Difficulty: Easy
* Tags: string, tree, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a rope tree node.
* struct RopeTreeNode {
* int len;
* string val;
* RopeTreeNode *left;
* RopeTreeNode *right;
* RopeTreeNode() : len(0), val(""), left(nullptr), right(nullptr) {
// TODO: Implement optimized solution
return 0;
}
* RopeTreeNode(string s) : len(0), val(std::move(s)), left(nullptr),
right(nullptr) {
// TODO: Implement optimized solution
return 0;
}
* RopeTreeNode(int x) : len(x), val(""), left(nullptr), right(nullptr) {
// TODO: Implement optimized solution
return 0;
}
* RopeTreeNode(int x, RopeTreeNode *left, RopeTreeNode *right) : len(x),
val(""), left(left), right(right) {
// TODO: Implement optimized solution
return 0;
}
* };
*/
```

```
class Solution {
public:
char getKthCharacter(RopeTreeNode* root, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Extract Kth Character From The Rope Tree
* Difficulty: Easy
* Tags: string, tree, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a rope tree node.
* class RopeTreeNode {
* int len;
* String val;
* RopeTreeNode left;
* RopeTreeNode right;
* RopeTreeNode() {
// TODO: Implement optimized solution
return 0;
}
* RopeTreeNode(String val) {
* this.len = 0;
* this.val = val;
* }
* RopeTreeNode(int len) {
* this.len = len;
* this.val = "";
* }
* RopeTreeNode(int len, RopeTreeNode left, RopeTreeNode right) {
* this.len = len;
* this.val = "";
```

```
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public char getKthCharacter(RopeTreeNode root, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Extract Kth Character From The Rope Tree
Difficulty: Easy
Tags: string, tree, search

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

# Definition for a rope tree node.
# class RopeTreeNode(object):
# def __init__(self, len=0, val="", left=None, right=None):
# self.len = len
# self.val = val
# self.left = left
# self.right = right
class Solution:
def getKthCharacter(self, root: Optional[object], k: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# Definition for a rope tree node.
# class RopeTreeNode(object):
# def __init__(self, len=0, val="", left=None, right=None):
# self.len = len
```

```
# self.val = val
# self.left = left
# self.right = right
class Solution(object):
def getKthCharacter(self, root, k):
"""
:type root: Optional[RopeTreeNode]
:type k: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Extract Kth Character From The Rope Tree
* Difficulty: Easy
* Tags: string, tree, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a rope tree node.
* class RopeTreeNode {
* constructor(len, val, left, right) {
* this.len = (len===undefined ? 0 : len);
* this.val = (val===undefined ? "" : val);
* this.left = (left===undefined ? null : left);
* this.right = (right===undefined ? null : right);
* }
* }
*/
/**
* @param {RopeTreeNode} root
* @param {number} k
* @return {character}
*/
var getKthCharacter = function(root, k) {
```

```
};
```

C# Solution:

```
/*
* Problem: Extract Kth Character From The Rope Tree
* Difficulty: Easy
* Tags: string, tree, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a rope tree node.
* public class RopeTreeNode {
* public int len;
* public string val;
* public RopeTreeNode left;
* public RopeTreeNode right;
* public RopeTreeNode(int len=0, string val="", RopeTreeNode left=null,
RopeTreeNode right=null) {
* this.len = len;
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
public class Solution {
public char GetKthCharacter(RopeTreeNode root, int k) {

}
}
```

C Solution:

```
/*
* Problem: Extract Kth Character From The Rope Tree
* Difficulty: Easy
```

```
* Tags: string, tree, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a rope tree node. */
struct RopeTreeNode {
int len;
char* val;
struct RopeTreeNode* left;
struct RopeTreeNode* right;
};

/// DO NOT MODIFY THE CODE ABOVE

char getKthCharacter(struct RopeTreeNode* root, int k){

}
```

Go Solution:

```
// Problem: Extract Kth Character From The Rope Tree
// Difficulty: Easy
// Tags: string, tree, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

/**
* Definition for a rope tree node.
* type RopeTreeNode struct {
* len int
* val string
* left *RopeTreeNode
* right *RopeTreeNode
* }
*/
```

```
func getKthCharacter(root *TreeNode, k int) byte {

}
```

Swift Solution:

```
/**
* Definition for a rope tree node.
* public class RopeTreeNode {
* var len: Int
* var val: String
* var left: RopeTreeNode?
* var right: RopeTreeNode?
* init(len: Int = 0, val: String = "", left: RopeTreeNode? = nil, right:
RopeTreeNode? = nil) {
* self.len = len
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/
class Solution {
func getKthCharacter(_ root: RopeTreeNode?, _ k: Int) -> Character {

}
}
```