Problem 1506: Find Root of N-Ary Tree

Problem Information

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

Problem Description

You are given all the nodes of an

N-ary tree

as an array of

Node

objects, where each node has a

unique value

.

Return

the

root

of the N-ary tree

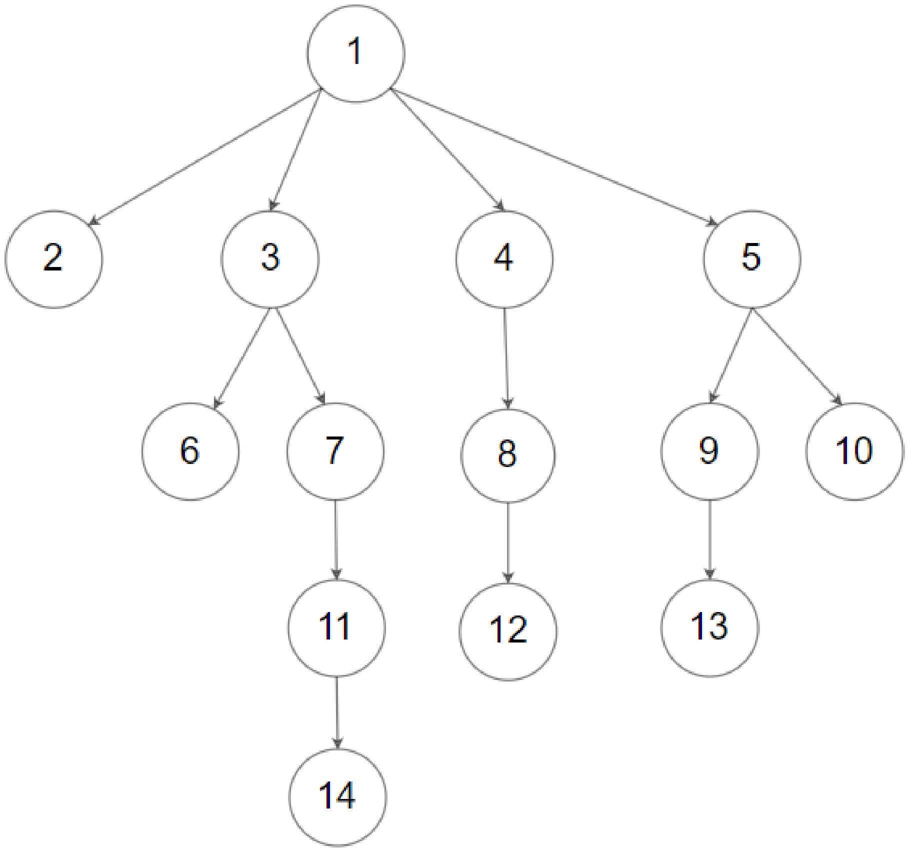
.

Custom testing:

An N-ary tree can be serialized as represented in its level order traversal where each group of children is separated by the

null

value (see examples).



For example, the above tree is serialized as

[1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null,13,null,null,14]

.

The testing will be done in the following way:

The

input data

should be provided as a serialization of the tree.

The driver code will construct the tree from the serialized input data and put each

Node

object into an array

in an arbitrary order

.

The driver code will pass the array to

findRoot

, and your function should find and return the root

Node

object in the array.

The driver code will take the returned

Node

object and serialize it. If the serialized value and the input data are the

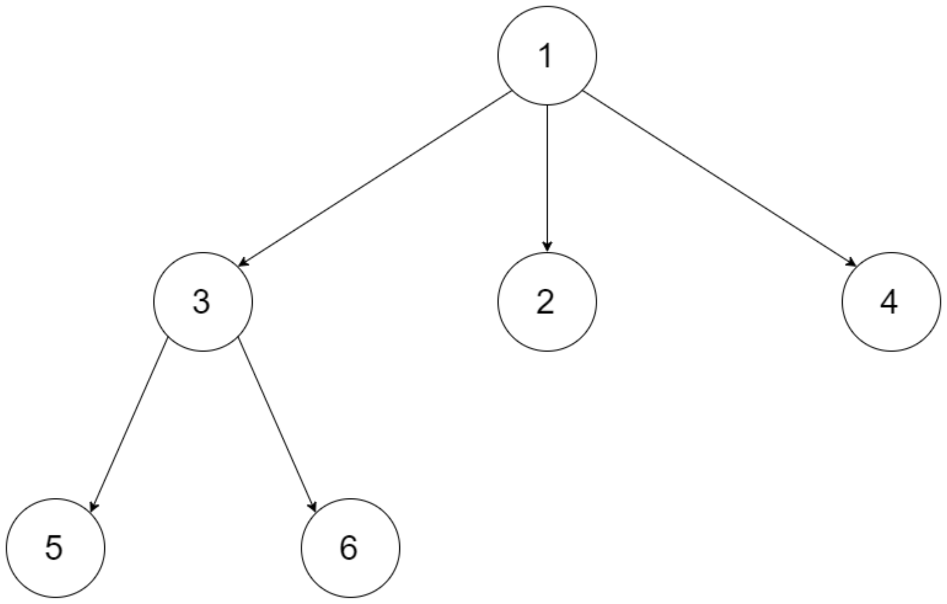
same

, the test

passes

.

Example 1:



Input:

tree = [1,null,3,2,4,null,5,6]

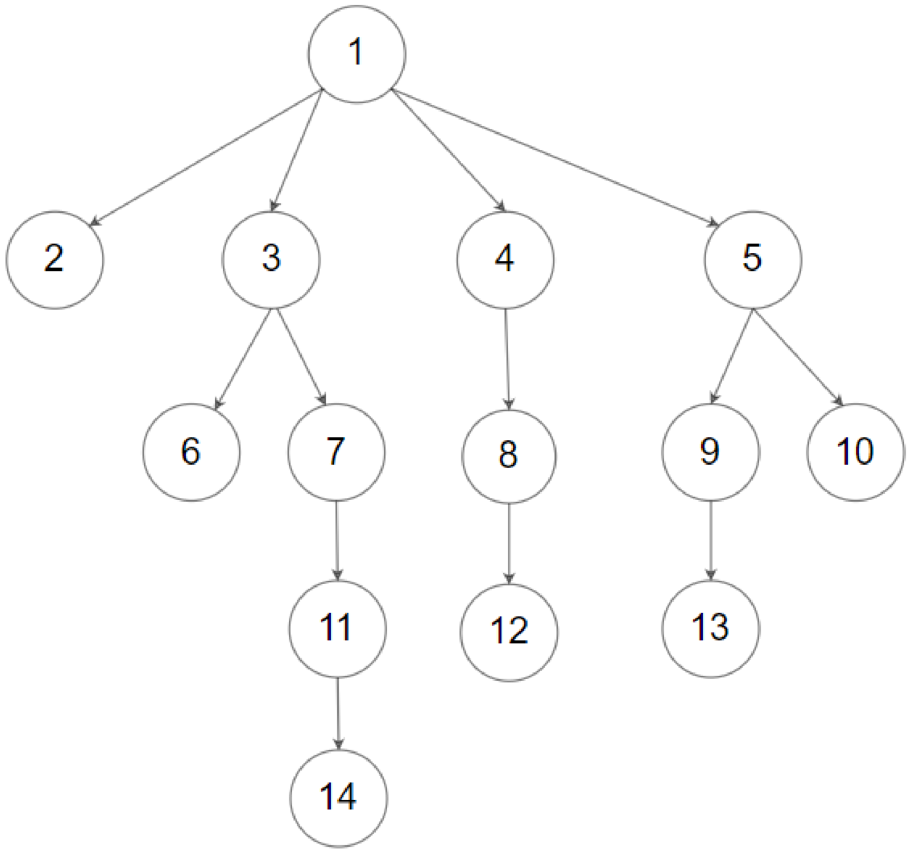
Output:

[1,null,3,2,4,null,5,6]

Explanation:

The tree from the input data is shown above. The driver code creates the tree and gives findRoot the Node objects in an arbitrary order. For example, the passed array could be [Node(5),Node(4),Node(3),Node(6),Node(2),Node(1)] or [Node(2),Node(6),Node(1),Node(3),Node(5),Node(4)]. The findRoot function should return the root Node(1), and the driver code will serialize it and compare with the input data. The input data and serialized Node(1) are the same, so the test passes.

Example 2:



Input:

tree = [1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null,13,null,null,14]

Output:

[1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null,13,null,null,14]

Constraints:

The total number of nodes is between

[1, 5 * 10

4

]

.

Each node has a

unique

value.

Follow up:

Could you solve this problem in constant space complexity with a linear time algorithm?

Code Snippets

C++:

```
/*
// Definition for a Node.
class Node {
public:
int val;
vector<Node*> children;

Node() {}

Node(int _val) {
val = _val;
}

Node(int _val, vector<Node*> _children) {
val = _val;
children = _children;
}
};
*/

class Solution {
public:
Node* findRoot(vector<Node*> tree) {

}
};
```

Java:

```
/*
// Definition for a Node.
class Node {
public int val;
public List<Node> children;


public Node() {
children = new ArrayList<Node>();
}

public Node(int _val) {
val = _val;
children = new ArrayList<Node>();
}

public Node(int _val,ArrayList<Node> _children) {
val = _val;
children = _children;
}
};
*/

class Solution {
public Node findRoot(List<Node> tree) {

}
}
```

Python3:

```
"""
# Definition for a Node.
class Node:
def __init__(self, val: Optional[int] = None, children:
Optional[List['Node']] = None):
self.val = val
self.children = children if children is not None else []
"""
```

```

class Solution:
def findRoot(self, tree: List['Node']) -> 'Node':
```

Python:

```
"""
# Definition for a Node.
class Node(object):
def __init__(self, val=None, children=None):
self.val = val
self.children = children if children is not None else []
"""

class Solution(object):
def findRoot(self, tree):
"""
:type tree: List['Node']
:rtype: 'Node'
"""
```

JavaScript:

```
/**
* // Definition for a _Node.
* function _Node(val, children) {
* this.val = val === undefined ? 0 : val;
* this.children = children === undefined ? [] : children;
* };
*/

/**
* @param {_Node[]} tree
* @return {_Node}
*/
var findRoot = function(tree) {

};
```

TypeScript:

```
/**
* Definition for _Node.
* class _Node {
* val: number
* children: _Node[]
*
* constructor(val?: number, children?: _Node[]) {
* this.val = (val===undefined ? 0 : val)
* this.children = (children===undefined ? [] : children)
* }
* }
*/


function findRoot(tree: _Node[]): _Node | null {

};
```

C#:

```
/*
// Definition for a Node.
public class Node {
public int val;
public IList<Node> children;

public Node() {
val = 0;
children = new List<Node>();
}

public Node(int _val) {
val = _val;
children = new List<Node>();
}

public Node(int _val, List<Node> _children) {
val = _val;
children = _children;
}
}
*/

public class Solution {
```

```
public Node FindRoot(List<Node> tree) {

}
}
```

Go:

```
/**
* Definition for a Node.
* type Node struct {
* Val int
* Children []*Node
* }
*/

func findRoot(tree []*Node) *Node {

}
```

Kotlin:

```
/**
* Definition for a Node.
* class Node(var `val`: Int) {
* var children: List<Node?> = listOf()
* }
*/

class Solution {
fun findRoot(tree: List<Node>): Node? {

}
}
```

Swift:

```
/**
* Definition for a Node.
* public class Node {
* public var val: Int
* public var children: [Node]
* public init(_ val: Int) {
```

```
* self.val = val
* self.children = []
* }
* }
*/

class Solution {
func findRoot(_ tree: [Node]) -> Node? {

}
}
```

Ruby:

```
# Definition for a Node.
# class Node
# attr_accessor :val, :children
# def initialize(val=0, children=[])
# @val = val
# @children = children
# end
# end

# @param {Node[]} tree
# @return {Node}
def find_root(tree)

end
```

PHP:

```
/**
* Definition for a Node.
* class Node {
* public $val = null;
* public $children = null;
* function __construct($val = 0) {
* $this->val = $val;
* $this->children = array();
* }
* }
*/
```

```
class Solution {
/**
* @param Node[] $tree
* @return Node
*/
function findRoot($tree) {

}
}
```

Scala:

```
/**
* Definition for a Node.
* class Node(var _value: Int) {
* var value: Int = _value
* var children: List[Node] = List()
* }
*/

object Solution {
def findRoot(tree: List[Node]): Node = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Find Root of N-Ary 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 Node.
class Node {
public:
int val;
vector<Node*> children;

Node() {}

Node(int _val) {
val = _val;
}

Node(int _val, vector<Node*> _children) {
val = _val;
children = _children;
}
};
*/

class Solution {
public:
Node* findRoot(vector<Node*> tree) {

}
};
```

Java Solution:

```
/**
* Problem: Find Root of N-Ary 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 Node.
class Node {
public int val;
public List<Node> children;


public Node() {
children = new ArrayList<Node>();
}

public Node(int _val) {
val = _val;
children = new ArrayList<Node>();
}

public Node(int _val,ArrayList<Node> _children) {
val = _val;
children = _children;
}
};
*/

class Solution {
public Node findRoot(List<Node> tree) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Root of N-Ary 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 Node.
class Node:
def __init__(self, val: Optional[int] = None, children:
Optional[List['Node']] = None):
self.val = val
self.children = children if children is not None else []
"""

class Solution:
def findRoot(self, tree: List['Node']) -> 'Node':
# TODO: Implement optimized solution
pass
```

Python Solution:

```
"""
# Definition for a Node.
class Node(object):
def __init__(self, val=None, children=None):
self.val = val
self.children = children if children is not None else []
"""

class Solution(object):
def findRoot(self, tree):
"""
:type tree: List['Node']
:rtype: 'Node'
"""
```

JavaScript Solution:

```
/**
* Problem: Find Root of N-Ary 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 _Node.
* function _Node(val, children) {
* this.val = val === undefined ? 0 : val;
* this.children = children === undefined ? [] : children;
* };
*/

/**
* @param {_Node[]} tree
* @return {_Node}
*/
var findRoot = function(tree) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Root of N-Ary 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 _Node.
* class _Node {
* val: number
* children: _Node[]
*
* constructor(val?: number, children?: _Node[]) {
* this.val = (val===undefined ? 0 : val)
* this.children = (children===undefined ? [] : children)
* }
* }
*/
```

```
function findRoot(tree: _Node[]): _Node | null {

};
```

C# Solution:

```
/*
* Problem: Find Root of N-Ary 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 Node.
public class Node {
public int val;
public IList<Node> children;

public Node() {
val = 0;
children = new List<Node>();
}

public Node(int _val) {
val = _val;
children = new List<Node>();
}

public Node(int _val, List<Node> _children) {
val = _val;
children = _children;
}
}
*/
```

```
public class Solution {
public Node FindRoot(List<Node> tree) {

}
}
```

Go Solution:

```
// Problem: Find Root of N-Ary 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 Node.
* type Node struct {
* Val int
* Children []*Node
* }
*/

func findRoot(tree []*Node) *Node {

}
```

Kotlin Solution:

```
/**
* Definition for a Node.
* class Node(var `val`: Int) {
* var children: List<Node?> = listOf()
* }
*/

class Solution {
fun findRoot(tree: List<Node>): Node? {

}
}
```

Swift Solution:

```
/**
* Definition for a Node.
* public class Node {
* public var val: Int
* public var children: [Node]
* public init(_ val: Int) {
* self.val = val
* self.children = []
* }
* }
*/

class Solution {
func findRoot(_ tree: [Node]) -> Node? {

}
}
```

Ruby Solution:

```
# Definition for a Node.
# class Node
# attr_accessor :val, :children
# def initialize(val=0, children=[])
# @val = val
# @children = children
# end
# end

# @param {Node[]} tree
# @return {Node}
def find_root(tree)

end
```

PHP Solution:

```
/**
* Definition for a Node.
* class Node {
* public $val = null;
* public $children = null;
* function __construct($val = 0) {
* $this->val = $val;
* $this->children = array();
* }
* }
*/

class Solution {
/**
* @param Node[] $tree
* @return Node
*/
function findRoot($tree) {

}
}
```

Scala Solution:

```
/**
* Definition for a Node.
* class Node(var _value: Int) {
* var value: Int = _value
* var children: List[Node] = List()
* }
*/

object Solution {
def findRoot(tree: List[Node]): Node = {

}
}
```