Problem 1490: Clone N-ary Tree

Problem Information

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

Problem Description

Given a

root

of an N-ary tree, return a

deep copy

(clone) of the tree.

Each node in the n-ary tree contains a val (

int

) and a list (

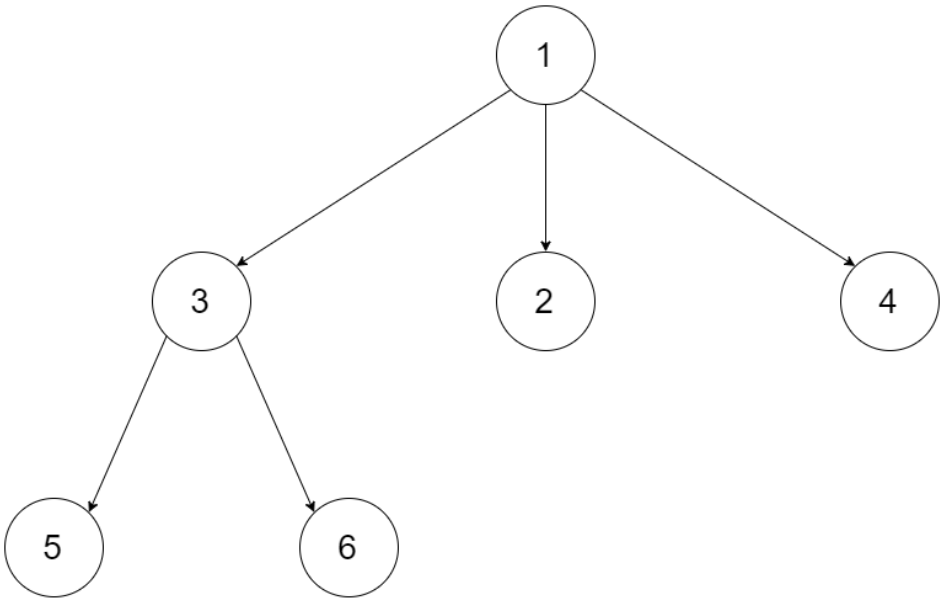
List[Node]

) of its children.

class Node { public int val; public List<Node> children; }

Nary-Tree input serialization is represented in their level order traversal, each group of children is separated by the null value (See examples).

Example 1:



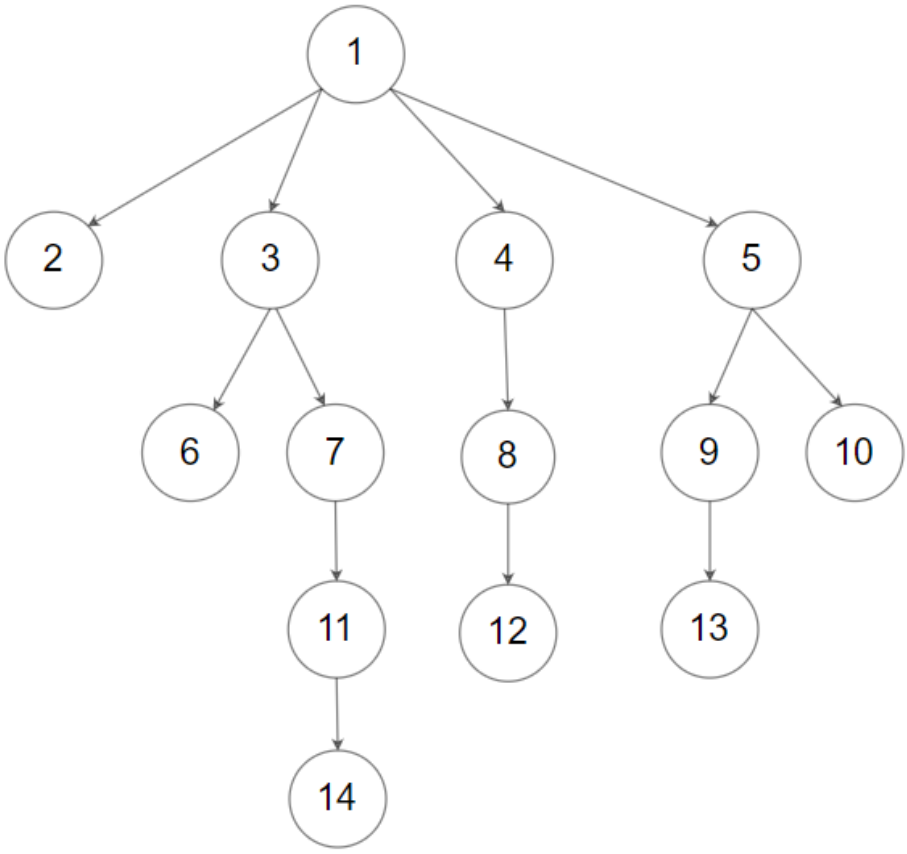
Input:

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

Output:

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

Example 2:



Input:

root = [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 depth of the n-ary tree is less than or equal to

1000

.

The total number of nodes is between

[0, 10

4

]

.

Follow up:

Can your solution work for the

graph problem

?

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* cloneTree(Node* root) {

}
};
```

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 cloneTree(Node root) {

}
}
```

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 cloneTree(self, root: '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 cloneTree(self, root):
"""
:type root: 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|null} node
* @return {_Node|null}
*/
var cloneTree = function(root) {
```

```
};
```

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 cloneTree(root: _Node | null): _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 CloneTree(Node root) {

}
}
```

Go:

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

func cloneTree(root *Node) *Node {

}
```

Kotlin:

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

class Solution {
fun cloneTree(root: 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 cloneTree(_ root: 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} root
# @return {Node}
def clone_tree(root)

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 $root
* @return Node
*/
function cloneTree($root) {

}
}
```

Scala:

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

object Solution {
def cloneTree(root: Node): Node = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Clone N-ary Tree
* Difficulty: Medium
* Tags: tree, graph, hash, search
```

```
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* 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* cloneTree(Node* root) {

}
};
```

Java Solution:

```
/**
* Problem: Clone N-ary Tree
* Difficulty: Medium
* Tags: tree, graph, hash, search
*
* Approach: DFS or BFS traversal
```

```
* Time Complexity: O(n) where n is number of nodes
* 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 cloneTree(Node root) {

}
}
```

Python3 Solution:

```
"""
Problem: Clone N-ary Tree
Difficulty: Medium
Tags: tree, graph, hash, search

Approach: DFS or BFS traversal
```

```
Time Complexity: O(n) where n is number of nodes
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 cloneTree(self, root: '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 cloneTree(self, root):
"""
:type root: Node
:rtype: Node
"""
```

JavaScript Solution:

```
/**
* Problem: Clone N-ary Tree
* Difficulty: Medium
* Tags: tree, graph, hash, search
```

```
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* 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|null} node
* @return {_Node|null}
*/
var cloneTree = function(root) {

};
```

TypeScript Solution:

```
/**
* Problem: Clone N-ary Tree
* Difficulty: Medium
* Tags: tree, graph, hash, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* 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 cloneTree(root: _Node | null): _Node | null {

};
```

C# Solution:

```
/*
* Problem: Clone N-ary Tree
* Difficulty: Medium
* Tags: tree, graph, hash, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* 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 CloneTree(Node root) {

}
}
```

Go Solution:

```
// Problem: Clone N-ary Tree
// Difficulty: Medium
// Tags: tree, graph, hash, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

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

func cloneTree(root *Node) *Node {

}
```

Kotlin Solution:

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

```
*/

class Solution {
fun cloneTree(root: 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 cloneTree(_ root: 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} root
# @return {Node}
```

```
def clone_tree(root)

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 $root
* @return Node
*/
function cloneTree($root) {

}
}
```

Scala Solution:

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

object Solution {
def cloneTree(root: Node): Node = {
```

```
}
}
```