Problem 429: N-ary Tree Level Order Traversal

Problem Information

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

Problem Description

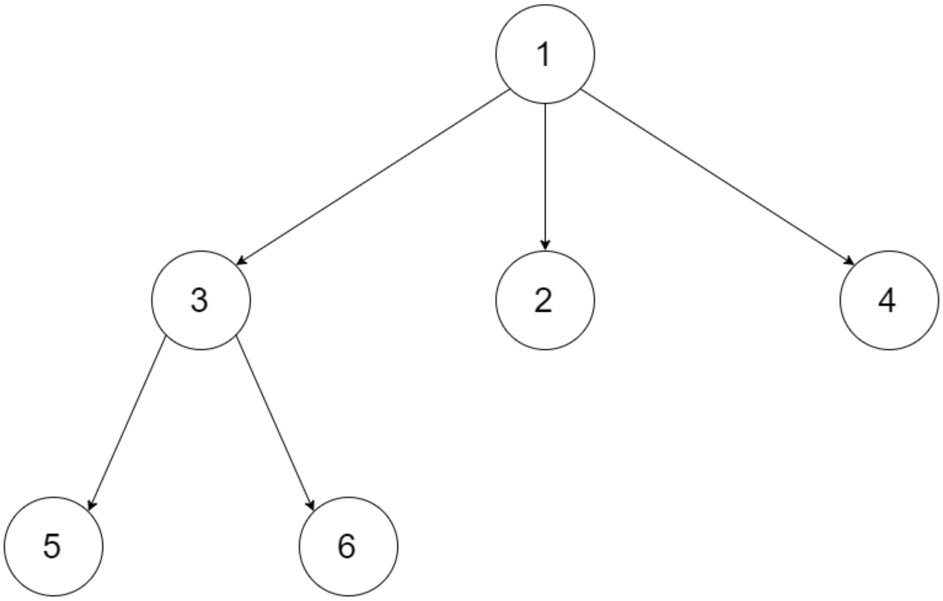
Given an n-ary tree, return the

level order

traversal of its nodes' values.

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:



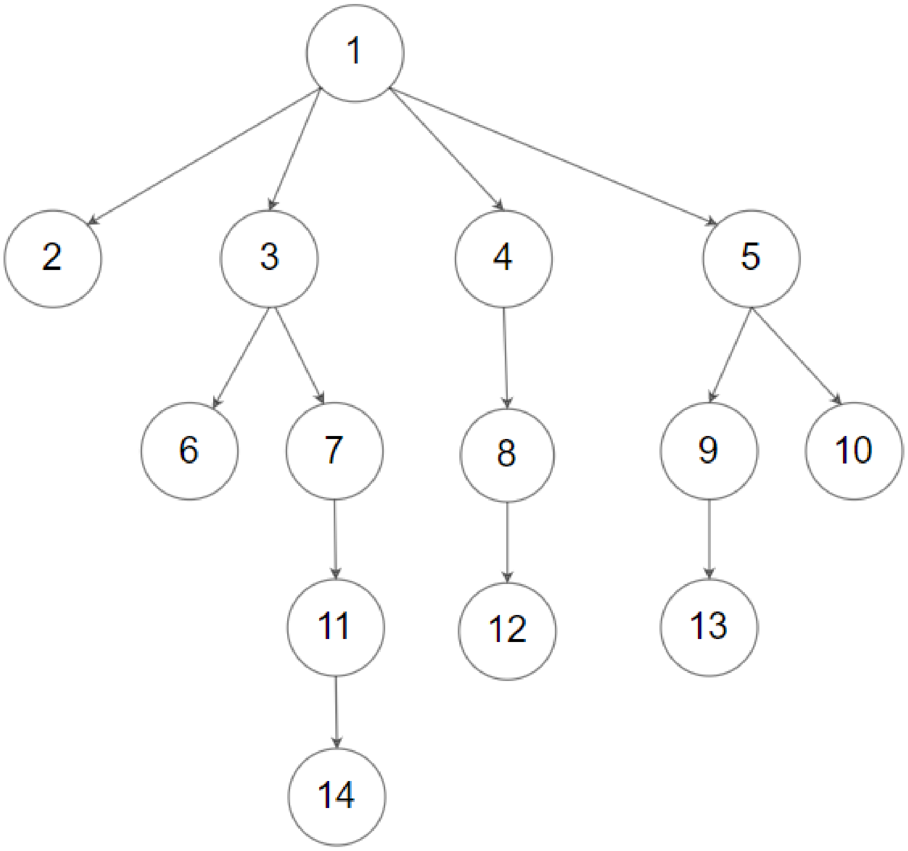
Input:

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

Output:

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

Constraints:

The height of the n-ary tree is less than or equal to

1000

The total number of nodes is between

[0, 10

4

]

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:
vector<vector<int>> levelOrder(Node* root) {

}
};
```

Java:

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

public Node() {}

public Node(int _val) {
val = _val;
}

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

class Solution {
public List<List<Integer>> levelOrder(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
"""

class Solution:
def levelOrder(self, root: 'Node') -> List[List[int]]:
```

Python:

```
"""
# Definition for a Node.
class Node(object):
def __init__(self, val=None, children=None):
self.val = val
self.children = children
"""

class Solution(object):
def levelOrder(self, root):
"""
:type root: Node
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* // Definition for a _Node.
* function _Node(val,children) {
* this.val = val;
* this.children = children;
* };
*/

/**
* @param {_Node|null} root
* @return {number[][]}
*/
var levelOrder = function(root) {

};
```

TypeScript:

```
/**
* Definition for _Node.
* class _Node {
* val: number
* children: _Node[]
*
* constructor(v: number) {
* this.val = v;
* this.children = [];
* }
* }
*/


function levelOrder(root: _Node | null): number[][] {

};
```

C#:

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

public Node() {}

public Node(int _val) {
val = _val;
}

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

public class Solution {
public IList<IList<int>> LevelOrder(Node root) {

}
}
```

C:

```
/**
* Definition for a Node.
* struct Node {
```

```
* int val;
* int numChildren;
* struct Node** children;
* };
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume caller calls free().
*/
int** levelOrder(struct Node* root, int* returnSize, int** returnColumnSizes)
{

}
```

Go:

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

func levelOrder(root *Node) [][]int {

}
```

Kotlin:

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

class Solution {
fun levelOrder(root: Node?): List<List<Int>> {


}
}
```

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 levelOrder(_ root: Node?) -> [[Int]] {

}
}
```

Ruby:

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

# @param {Node} root
# @return {List[List[int]]}
def level_order(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 $root
* @return integer[][]
*/
function levelOrder($root) {

}
}
```

Scala:

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

object Solution {
def levelOrder(root: Node): List[List[Int]] = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: N-ary Tree Level Order Traversal
* Difficulty: Medium
* Tags: tree, 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:
vector<vector<int>> levelOrder(Node* root) {

}
};
```

Java Solution:

```
/**
* Problem: N-ary Tree Level Order Traversal
* Difficulty: Medium
* Tags: tree, 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() {}

public Node(int _val) {
val = _val;
}

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

class Solution {
public List<List<Integer>> levelOrder(Node root) {

}
}
```

Python3 Solution:

```
"""
Problem: N-ary Tree Level Order Traversal
Difficulty: Medium
Tags: tree, 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
"""

class Solution:
def levelOrder(self, root: 'Node') -> List[List[int]]:
# 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
"""

class Solution(object):
def levelOrder(self, root):
"""
:type root: Node
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: N-ary Tree Level Order Traversal
* Difficulty: Medium
* Tags: tree, 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;
* this.children = children;
* };
*/

/**
* @param {_Node|null} root
* @return {number[][]}
*/
var levelOrder = function(root) {

};
```

TypeScript Solution:

```
/**
* Problem: N-ary Tree Level Order Traversal
* Difficulty: Medium
* Tags: tree, 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(v: number) {
* this.val = v;
* this.children = [];
* }
* }
*/



function levelOrder(root: _Node | null): number[][] {

};
```

C# Solution:

```
/*
* Problem: N-ary Tree Level Order Traversal
* Difficulty: Medium
* Tags: tree, 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() {}

public Node(int _val) {
val = _val;
}

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

```
public class Solution {
public IList<IList<int>> LevelOrder(Node root) {

}
}
```

C Solution:

```
/*
* Problem: N-ary Tree Level Order Traversal
* Difficulty: Medium
* Tags: tree, 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.
* struct Node {
* int val;
* int numChildren;
* struct Node** children;
* };
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** levelOrder(struct Node* root, int* returnSize, int** returnColumnSizes)
{

}
```

Go Solution:

```
// Problem: N-ary Tree Level Order Traversal
// Difficulty: Medium
// Tags: tree, 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 levelOrder(root *Node) [][]int {

}
```

Kotlin Solution:

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

class Solution {
fun levelOrder(root: Node?): List<List<Int>> {

}
}
```

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 levelOrder(_ root: Node?) -> [[Int]] {

}
}
```

Ruby Solution:

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

# @param {Node} root
# @return {List[List[int]]}
def level_order(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 $root
* @return integer[][]
*/
function levelOrder($root) {

}
}
```

Scala Solution:

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

object Solution {
def levelOrder(root: Node): List[List[Int]] = {

}
}
```