

Problem 559: Maximum Depth of N-ary Tree

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

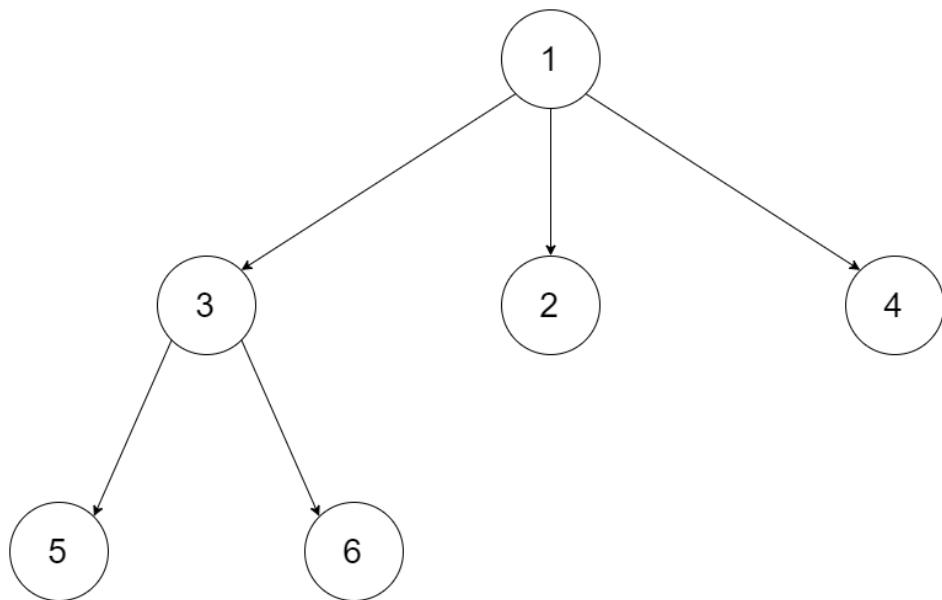
Problem Description

Given a n-ary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

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:



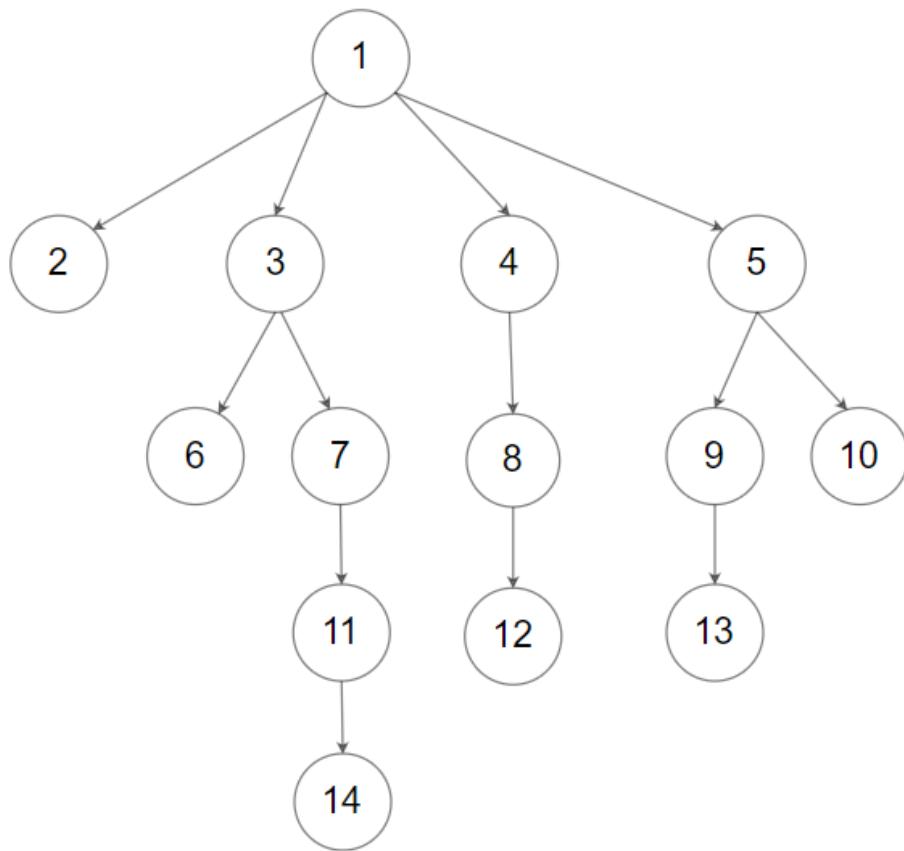
Input:

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

Output:

3

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:

5

Constraints:

The total number of nodes is in the range

[0, 10

4

]

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

1000

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:  
int maxDepth(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 int maxDepth(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 maxDepth(self, root: 'Node') -> 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 maxDepth(self, root):
        """
:type root: Node
:rtype: int
"""

"""

class Solution:
    def maxDepth(self, root: 'Node') -> int:
```

JavaScript:

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

/**
 * @param {_Node|null} root
 * @return {number}
 */
var maxDepth = 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 maxDepth(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 int MaxDepth(Node root) {

}

```

```
}
```

C:

```
/**  
 * Definition for a Node.  
 * struct Node {  
 *     int val;  
 *     int numChildren;  
 *     struct Node** children;  
 * };  
 */  
  
int maxDepth(struct Node* root) {  
  
}
```

Go:

```
/**  
 * Definition for a Node.  
 * type Node struct {  
 *     Val int  
 *     Children []*Node  
 * }  
 */  
  
func maxDepth(root *Node) int {  
  
}
```

Kotlin:

```
/**  
 * Definition for a Node.  
 * class Node(var `val`: Int) {  
 *     var children: List<Node?> = listOf()  
 * }  
 */  
  
class Solution {  
    fun maxDepth(root: Node?): 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 maxDepth(_ 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 {int}  
def maxDepth(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 $root
 * @return integer
 */
function maxDepth($root) {

}
}

```

Scala:

```

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

object Solution {
def maxDepth(root: Node): Int = {

}
}

```

Solutions

C++ Solution:

```

/*
 * Problem: Maximum Depth of N-ary Tree
 * Difficulty: Easy
 * 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() {
        // TODO: Implement optimized solution
        return 0;
    }

    Node(int _val) {
        val = _val;
    }

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

class Solution {
public:
    int maxDepth(Node* root) {

    }
};

```

Java Solution:

```

/**
 * Problem: Maximum Depth of N-ary Tree
 * Difficulty: Easy
 * 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() {
    // TODO: Implement optimized solution
    return 0;
}

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

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

class Solution {
public int maxDepth(Node root) {

}
}

```

Python3 Solution:

```

"""
Problem: Maximum Depth of N-ary Tree

```

```

Difficulty: Easy
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 maxDepth(self, root: 'Node') -> 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 maxDepth(self, root):
        """
:type root: Node
:rtype: int
"""

```

JavaScript Solution:

```

    /**
 * Problem: Maximum Depth of N-ary Tree
 * Difficulty: Easy
 * 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 === undefined ? null : val;
 *   this.children = children === undefined ? null : children;
 * }
 */

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

};

```

TypeScript Solution:

```

    /**
 * Problem: Maximum Depth of N-ary Tree
 * Difficulty: Easy
 * 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(val?: number, children?: _Node[]) {
*   this.val = (val==undefined ? 0 : val)
*   this.children = (children==undefined ? [] : children)
* }
*
*/

```



```

function maxDepth(root: _Node | null): number {
}

```

C# Solution:

```

/*
* Problem: Maximum Depth of N-ary Tree
* Difficulty: Easy
* 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 int MaxDepth(Node root) {

}

}

```

C Solution:

```

/*
 * Problem: Maximum Depth of N-ary Tree
 * Difficulty: Easy
 * 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;
 * };
 */

int maxDepth(struct Node* root) {

}

```

Go Solution:

```

// Problem: Maximum Depth of N-ary Tree
// Difficulty: Easy
// 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 maxDepth(root *Node) int {
}

```

Kotlin Solution:

```

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

class Solution {
    fun maxDepth(root: Node?): 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 maxDepth(_ 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 {int}
def maxDepth(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 $root  
     * @return integer  
     */  
    function maxDepth($root) {  
  
    }  
}
```

Scala Solution:

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