Problem 427: Construct Quad Tree

Problem Information

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

Problem Description

Given a

n * n

matrix

grid

of

0's

and

1's

only. We want to represent

grid

with a Quad-Tree.

Return

the root of the Quad-Tree representing

grid

.

A Quad-Tree is a tree data structure in which each internal node has exactly four children. Besides, each node has two attributes:

val

: True if the node represents a grid of 1's or False if the node represents a grid of 0's. Notice that you can assign the

val

to True or False when

isLeaf

is False, and both are accepted in the answer.

isLeaf

: True if the node is a leaf node on the tree or False if the node has four children.

class Node { public boolean val; public boolean isLeaf; public Node topLeft; public Node topRight; public Node bottomLeft; public Node bottomRight; }

We can construct a Quad-Tree from a two-dimensional area using the following steps:

If the current grid has the same value (i.e all

1's

or all

0's

) set

isLeaf

True and set

val

to the value of the grid and set the four children to Null and stop.

If the current grid has different values, set

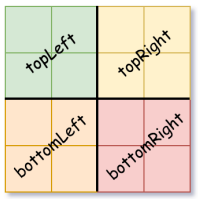
isLeaf

to False and set

val

to any value and divide the current grid into four sub-grids as shown in the photo.

Recurse for each of the children with the proper sub-grid.



If you want to know more about the Quad-Tree, you can refer to the

wiki

.

Quad-Tree format:

You don't need to read this section for solving the problem. This is only if you want to understand the output format here. The output represents the serialized format of a Quad-Tree using level order traversal, where

null

signifies a path terminator where no node exists below.

It is very similar to the serialization of the binary tree. The only difference is that the node is represented as a list

[isLeaf, val]

.

If the value of

isLeaf

or

val

is True we represent it as

1

in the list

[isLeaf, val]

and if the value of

isLeaf

or

val

is False we represent it as

0

.

Example 1:

Input:

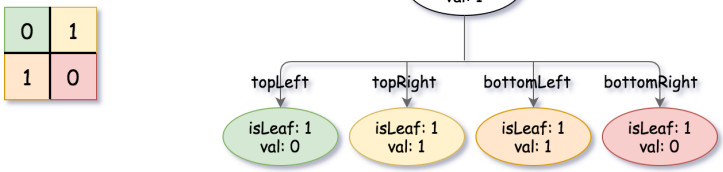
grid = [[0,1],[1,0]]

Output:

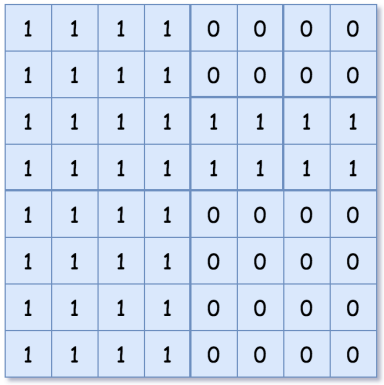
[[0,1],[1,0],[1,1],[1,1],[1,0]]

Explanation:

The explanation of this example is shown below: Notice that 0 represents False and 1 represents True in the photo representing the Quad-Tree.



Example 2:



Input:

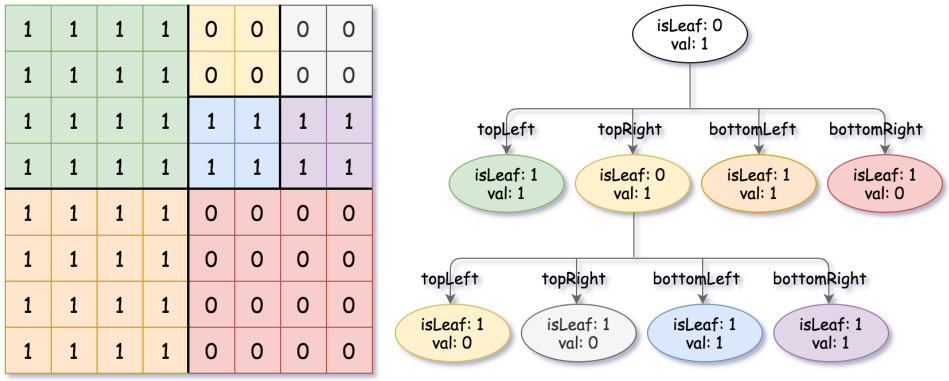
grid = [[1,1,1,1,0,0,0,0],[1,1,1,1,0,0,0,0],[1,1,1,1,1,1,1,1],[1,1,1,1,1,1,1,1],[1,1,1,1,0,0,0,0],[1,1,1,1,0,0,0,0],[1,1,1,1,0,0,0,0],[1,1,1,1,0,0,0,0]]

Output:

[[0,1],[1,1],[0,1],[1,1],[1,0],null,null,null,null,[1,0],[1,0],[1,1],[1,1]]

Explanation:

All values in the grid are not the same. We divide the grid into four sub-grids. The topLeft, bottomLeft and bottomRight each has the same value. The topRight have different values so we divide it into 4 sub-grids where each has the same value. Explanation is shown in the photo below:



Constraints:

n == grid.length == grid[i].length

n == 2

x

where

0 <= x <= 6

Code Snippets

C++:

```
/*
// Definition for a QuadTree node.
class Node {
public:
bool val;
bool isLeaf;
Node* topLeft;
Node* topRight;
Node* bottomLeft;
Node* bottomRight;

Node() {
val = false;
isLeaf = false;
topLeft = NULL;
topRight = NULL;
bottomLeft = NULL;
bottomRight = NULL;
}

Node(bool _val, bool _isLeaf) {
val = _val;
isLeaf = _isLeaf;
topLeft = NULL;
topRight = NULL;
bottomLeft = NULL;
bottomRight = NULL;
}

Node(bool _val, bool _isLeaf, Node* _topLeft, Node* _topRight, Node*
_bottomLeft, Node* _bottomRight) {
val = _val;
isLeaf = _isLeaf;
topLeft = _topLeft;
topRight = _topRight;
bottomLeft = _bottomLeft;
bottomRight = _bottomRight;
}
};
*/

class Solution {
public:
```

```
Node* construct(vector<vector<int>>& grid) {

}
};
```

Java:

```
/*
// Definition for a QuadTree node.
class Node {
public boolean val;
public boolean isLeaf;
public Node topLeft;
public Node topRight;
public Node bottomLeft;
public Node bottomRight;


public Node() {
this.val = false;
this.isLeaf = false;
this.topLeft = null;
this.topRight = null;
this.bottomLeft = null;
this.bottomRight = null;
}

public Node(boolean val, boolean isLeaf) {
this.val = val;
this.isLeaf = isLeaf;
this.topLeft = null;
this.topRight = null;
this.bottomLeft = null;
this.bottomRight = null;
}

public Node(boolean val, boolean isLeaf, Node topLeft, Node topRight, Node
bottomLeft, Node bottomRight) {
this.val = val;
this.isLeaf = isLeaf;
this.topLeft = topLeft;
this.topRight = topRight;
```

```
this.bottomLeft = bottomLeft;
this.bottomRight = bottomRight;
}
}
*/

class Solution {
public Node construct(int[][] grid) {

}
}
```

Python3:

```
"""
# Definition for a QuadTree node.
class Node:
def __init__(self, val, isLeaf, topLeft, topRight, bottomLeft, bottomRight):
self.val = val
self.isLeaf = isLeaf
self.topLeft = topLeft
self.topRight = topRight
self.bottomLeft = bottomLeft
self.bottomRight = bottomRight
"""

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

Python:

```
"""
# Definition for a QuadTree node.
class Node(object):
def __init__(self, val=False, isLeaf=False, topLeft=None, topRight=None,
bottomLeft=None, bottomRight=None):
self.val = val
self.isLeaf = isLeaf
self.topLeft = topLeft
self.topRight = topRight
self.bottomLeft = bottomLeft
self.bottomRight = bottomRight
```

```
"""

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

JavaScript:

```
/**
* // Definition for a QuadTree node.
* function _Node(val,isLeaf,topLeft,topRight,bottomLeft,bottomRight) {
* this.val = val;
* this.isLeaf = isLeaf;
* this.topLeft = topLeft;
* this.topRight = topRight;
* this.bottomLeft = bottomLeft;
* this.bottomRight = bottomRight;
* };
*/

/**
* @param {number[][]} grid
* @return {_Node}
*/
var construct = function(grid) {

};
```

TypeScript:

```
/**
* Definition for _Node.
* class _Node {
* val: boolean
* isLeaf: boolean
* topLeft: _Node | null
* topRight: _Node | null
* bottomLeft: _Node | null
* bottomRight: _Node | null
```

```
* constructor(val?: boolean, isLeaf?: boolean, topLeft?: _Node, topRight?:
_Node, bottomLeft?: _Node, bottomRight?: _Node) {
* this.val = (val===undefined ? false : val)
* this.isLeaf = (isLeaf===undefined ? false : isLeaf)
* this.topLeft = (topLeft===undefined ? null : topLeft)
* this.topRight = (topRight===undefined ? null : topRight)
* this.bottomLeft = (bottomLeft===undefined ? null : bottomLeft)
* this.bottomRight = (bottomRight===undefined ? null : bottomRight)
* }
* }
*/


function construct(grid: number[][]): _Node | null {

};
```

C#:

```
/*
// Definition for a QuadTree node.
public class Node {
public bool val;
public bool isLeaf;
public Node topLeft;
public Node topRight;
public Node bottomLeft;
public Node bottomRight;

public Node() {
val = false;
isLeaf = false;
topLeft = null;
topRight = null;
bottomLeft = null;
bottomRight = null;
}

public Node(bool _val, bool _isLeaf) {
val = _val;
isLeaf = _isLeaf;
topLeft = null;
```

```
topRight = null;
bottomLeft = null;
bottomRight = null;
}

public Node(bool _val,bool _isLeaf,Node _topLeft,Node _topRight,Node
_bottomLeft,Node _bottomRight) {
val = _val;
isLeaf = _isLeaf;
topLeft = _topLeft;
topRight = _topRight;
bottomLeft = _bottomLeft;
bottomRight = _bottomRight;
}
}
*/

public class Solution {
public Node Construct(int[][] grid) {

}
}
```

Go:

```
/**
* Definition for a QuadTree node.
* type Node struct {
* Val bool
* IsLeaf bool
* TopLeft *Node
* TopRight *Node
* BottomLeft *Node
* BottomRight *Node
* }
*/

func construct(grid [][]int) *Node {

}
```

Kotlin:

```
/**
* Definition for a QuadTree node.
* class Node(var `val`: Boolean, var isLeaf: Boolean) {
* var topLeft: Node? = null
* var topRight: Node? = null
* var bottomLeft: Node? = null
* var bottomRight: Node? = null
* }
*/

class Solution {
fun construct(grid: Array<IntArray>): Node? {

}
}
```

Swift:

```
/**
* Definition for a QuadTree node.
* public class Node {
* public var val: Bool
* public var isLeaf: Bool
* public var topLeft: Node?
* public var topRight: Node?
* public var bottomLeft: Node?
* public var bottomRight: Node?
* public init(_ val: Bool, _ isLeaf: Bool) {
* self.val = val
* self.isLeaf = isLeaf
* self.topLeft = nil
* self.topRight = nil
* self.bottomLeft = nil
* self.bottomRight = nil
* }
* }
*/

class Solution {
func construct(_ grid: [[Int]]) -> Node? {
```

```
}
}
```

Ruby:

```
# Definition for a QuadTree node.
# class Node
# attr_accessor :val, :isLeaf, :topLeft, :topRight, :bottomLeft, :bottomRight
# def initialize(val=false, isLeaf=false, topLeft=nil, topRight=nil,
bottomLeft=nil, bottomRight=nil)
# @val = val
# @isLeaf = isLeaf
# @topLeft = topLeft
# @topRight = topRight
# @bottomLeft = bottomLeft
# @bottomRight = bottomRight
# end
# end

# @param {Integer[][]} grid
# @return {Node}
def construct(grid)

end
```

PHP:

```
/**
* Definition for a QuadTree node.
* class Node {
* public $val = null;
* public $isLeaf = null;
* public $topLeft = null;
* public $topRight = null;
* public $bottomLeft = null;
* public $bottomRight = null;
* function __construct($val, $isLeaf) {
* $this->val = $val;
* $this->isLeaf = $isLeaf;
* $this->topLeft = null;
* $this->topRight = null;
* $this->bottomLeft = null;
```

```
* $this->bottomRight = null;
* }
* }
*/

class Solution {
/**
* @param Integer[][] $grid
* @return Node
*/
function construct($grid) {

}
}
```

Scala:

```
/**
* Definition for a QuadTree node.
* class Node(var _value: Boolean, var _isLeaf: Boolean) {
* var value: Int = _value
* var isLeaf: Boolean = _isLeaf
* var topLeft: Node = null
* var topRight: Node = null
* var bottomLeft: Node = null
* var bottomRight: Node = null
* }
*/

object Solution {
def construct(grid: Array[Array[Int]]): Node = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Construct Quad Tree
* Difficulty: Medium
* Tags: array, tree
*
* 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 QuadTree node.
class Node {
public:
bool val;
bool isLeaf;
Node* topLeft;
Node* topRight;
Node* bottomLeft;
Node* bottomRight;

Node() {
val = false;
isLeaf = false;
topLeft = NULL;
topRight = NULL;
bottomLeft = NULL;
bottomRight = NULL;
}

Node(bool _val, bool _isLeaf) {
val = _val;
isLeaf = _isLeaf;
topLeft = NULL;
topRight = NULL;
bottomLeft = NULL;
bottomRight = NULL;
}

Node(bool _val, bool _isLeaf, Node* _topLeft, Node* _topRight, Node*
_bottomLeft, Node* _bottomRight) {
val = _val;
isLeaf = _isLeaf;
```

```
topLeft = _topLeft;
topRight = _topRight;
bottomLeft = _bottomLeft;
bottomRight = _bottomRight;
}
};
*/

class Solution {
public:
Node* construct(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Construct Quad Tree
* Difficulty: Medium
* Tags: array, tree
*
* 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 QuadTree node.
class Node {
public boolean val;
public boolean isLeaf;
public Node topLeft;
public Node topRight;
public Node bottomLeft;
public Node bottomRight;


public Node() {
this.val = false;
this.isLeaf = false;
```

```
this.topLeft = null;
this.topRight = null;
this.bottomLeft = null;
this.bottomRight = null;
}

public Node(boolean val, boolean isLeaf) {
this.val = val;
this.isLeaf = isLeaf;
this.topLeft = null;
this.topRight = null;
this.bottomLeft = null;
this.bottomRight = null;
}

public Node(boolean val, boolean isLeaf, Node topLeft, Node topRight, Node
bottomLeft, Node bottomRight) {
this.val = val;
this.isLeaf = isLeaf;
this.topLeft = topLeft;
this.topRight = topRight;
this.bottomLeft = bottomLeft;
this.bottomRight = bottomRight;
}
}
*/

class Solution {
public Node construct(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Construct Quad Tree
Difficulty: Medium
Tags: array, tree

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 QuadTree node.
class Node:
def __init__(self, val, isLeaf, topLeft, topRight, bottomLeft, bottomRight):
self.val = val
self.isLeaf = isLeaf
self.topLeft = topLeft
self.topRight = topRight
self.bottomLeft = bottomLeft
self.bottomRight = bottomRight
"""

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

Python Solution:

```
"""
# Definition for a QuadTree node.
class Node(object):
def __init__(self, val=False, isLeaf=False, topLeft=None, topRight=None,
bottomLeft=None, bottomRight=None):
self.val = val
self.isLeaf = isLeaf
self.topLeft = topLeft
self.topRight = topRight
self.bottomLeft = bottomLeft
self.bottomRight = bottomRight
"""

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Construct Quad Tree
* Difficulty: Medium
* Tags: array, tree
*
* 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 QuadTree node.
* function _Node(val,isLeaf,topLeft,topRight,bottomLeft,bottomRight) {
* this.val = val;
* this.isLeaf = isLeaf;
* this.topLeft = topLeft;
* this.topRight = topRight;
* this.bottomLeft = bottomLeft;
* this.bottomRight = bottomRight;
* };
*/

/**
* @param {number[][]} grid
* @return {_Node}
*/
var construct = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Construct Quad Tree
* Difficulty: Medium
* Tags: array, tree
*
```

```
* 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: boolean
* isLeaf: boolean
* topLeft: _Node | null
* topRight: _Node | null
* bottomLeft: _Node | null
* bottomRight: _Node | null
* constructor(val?: boolean, isLeaf?: boolean, topLeft?: _Node, topRight?:
_Node, bottomLeft?: _Node, bottomRight?: _Node) {
* this.val = (val===undefined ? false : val)
* this.isLeaf = (isLeaf===undefined ? false : isLeaf)
* this.topLeft = (topLeft===undefined ? null : topLeft)
* this.topRight = (topRight===undefined ? null : topRight)
* this.bottomLeft = (bottomLeft===undefined ? null : bottomLeft)
* this.bottomRight = (bottomRight===undefined ? null : bottomRight)
* }
* }
*/


function construct(grid: number[][]): _Node | null {

};
```

C# Solution:

```
/*
* Problem: Construct Quad Tree
* Difficulty: Medium
* Tags: array, tree
*
* 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 QuadTree node.
public class Node {
public bool val;
public bool isLeaf;
public Node topLeft;
public Node topRight;
public Node bottomLeft;
public Node bottomRight;

public Node() {
val = false;
isLeaf = false;
topLeft = null;
topRight = null;
bottomLeft = null;
bottomRight = null;
}

public Node(bool _val, bool _isLeaf) {
val = _val;
isLeaf = _isLeaf;
topLeft = null;
topRight = null;
bottomLeft = null;
bottomRight = null;
}

public Node(bool _val,bool _isLeaf,Node _topLeft,Node _topRight,Node
_bottomLeft,Node _bottomRight) {
val = _val;
isLeaf = _isLeaf;
topLeft = _topLeft;
topRight = _topRight;
bottomLeft = _bottomLeft;
bottomRight = _bottomRight;
}
}
*/
```

```
public class Solution {
public Node Construct(int[][] grid) {

}
}
```

Go Solution:

```
// Problem: Construct Quad Tree
// Difficulty: Medium
// Tags: array, tree
//
// 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 QuadTree node.
* type Node struct {
* Val bool
* IsLeaf bool
* TopLeft *Node
* TopRight *Node
* BottomLeft *Node
* BottomRight *Node
* }
*/

func construct(grid [][]int) *Node {

}
```

Kotlin Solution:

```
/**
* Definition for a QuadTree node.
* class Node(var `val`: Boolean, var isLeaf: Boolean) {
* var topLeft: Node? = null
* var topRight: Node? = null
* var bottomLeft: Node? = null
* var bottomRight: Node? = null
```

```
* }
*/

class Solution {
fun construct(grid: Array<IntArray>): Node? {

}
}
```

Swift Solution:

```
/**
* Definition for a QuadTree node.
* public class Node {
* public var val: Bool
* public var isLeaf: Bool
* public var topLeft: Node?
* public var topRight: Node?
* public var bottomLeft: Node?
* public var bottomRight: Node?
* public init(_ val: Bool, _ isLeaf: Bool) {
* self.val = val
* self.isLeaf = isLeaf
* self.topLeft = nil
* self.topRight = nil
* self.bottomLeft = nil
* self.bottomRight = nil
* }
* }
*/

class Solution {
func construct(_ grid: [[Int]]) -> Node? {

}
}
```

Ruby Solution:

```
# Definition for a QuadTree node.
# class Node
```

```
# attr_accessor :val, :isLeaf, :topLeft, :topRight, :bottomLeft, :bottomRight
# def initialize(val=false, isLeaf=false, topLeft=nil, topRight=nil,
bottomLeft=nil, bottomRight=nil)
# @val = val
# @isLeaf = isLeaf
# @topLeft = topLeft
# @topRight = topRight
# @bottomLeft = bottomLeft
# @bottomRight = bottomRight
# end
# end

# @param {Integer[][]} grid
# @return {Node}
def construct(grid)

end
```

PHP Solution:

```
/**
* Definition for a QuadTree node.
* class Node {
* public $val = null;
* public $isLeaf = null;
* public $topLeft = null;
* public $topRight = null;
* public $bottomLeft = null;
* public $bottomRight = null;
* function __construct($val, $isLeaf) {
* $this->val = $val;
* $this->isLeaf = $isLeaf;
* $this->topLeft = null;
* $this->topRight = null;
* $this->bottomLeft = null;
* $this->bottomRight = null;
* }
* }
*/

class Solution {
```

```
/**
* @param Integer[][] $grid
* @return Node
*/
function construct($grid) {

}
}
```

Scala Solution:

```
/**
* Definition for a QuadTree node.
* class Node(var _value: Boolean, var _isLeaf: Boolean) {
* var value: Int = _value
* var isLeaf: Boolean = _isLeaf
* var topLeft: Node = null
* var topRight: Node = null
* var bottomLeft: Node = null
* var bottomRight: Node = null
* }
*/

object Solution {
def construct(grid: Array[Array[Int]]): Node = {

}
}
```