VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY

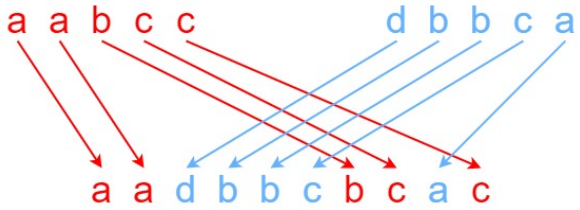
Problem 1: Interleaving String

Given strings `s1`, `s2`, and `s3`, find whether `s3` is formed by an **interleaving** of `s1` and `s2`.

An interleaving of two strings `s` and `t` is a configuration where s and t are divided into `n` and `m` **substrings** respectively, such that:

- s = $s_1 + s_2 + ... + s_n$
- t = $t_1 + t_2 + ... + t_m$
- |n - m| <= 1
- **The interleaving** is $s_1 + {}_t1 + s_2 + t_2 + s_3 + t_3 + ...$ or $t_1 + s_1 + t_2 + s_2 + t_3 + s_3 + ...$

Note: `a + b` is the concatenation of strings `a` and `b`.



Example 1:

Input: `s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac"`

Output: `True`

Explanation: `One way to obtain s3 is:`
`Split s1into s1 = "aa" + "bc" + "c", and s2 into s2 = "dbbc" + "a".`
`Interleaving the two splits, we get "aa" + "dbbc" + "bc" + "a" + "c" = "aadbbcbcac"`
`Since s3 can be obtained by interleaving s1 and s2, we return true.`

Example 2:

Input: `s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"`

Output: `False`

Explanation: `Notice how it is impossible to interleave s2 with any other string to obtain s3.`

Example 3:

Input: `s1 = "", s2 = "", s3 = ""`

Output: `True`

Constraints:
- `0 <= s1.length, s2.length <= 100`
- `0 <= s3.length <= 200`
- `s1`, `s2`, and `s3` consist of lowercase English letters.

Problem 2:

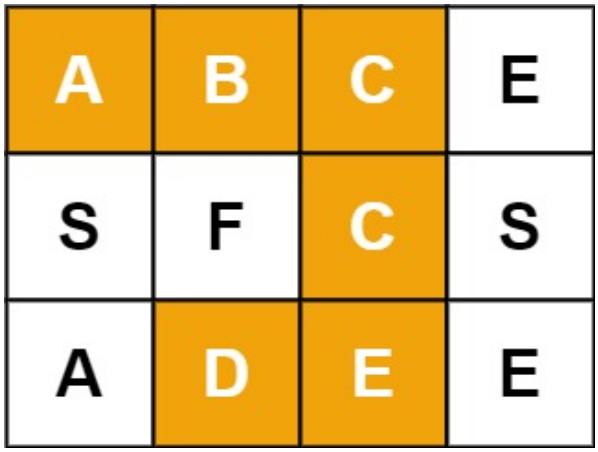
Given an `m x n` grid of characters `board` and a string `word`, return `true` if `word` exists in the `grid`.

The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example 1:

Input: `board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"`

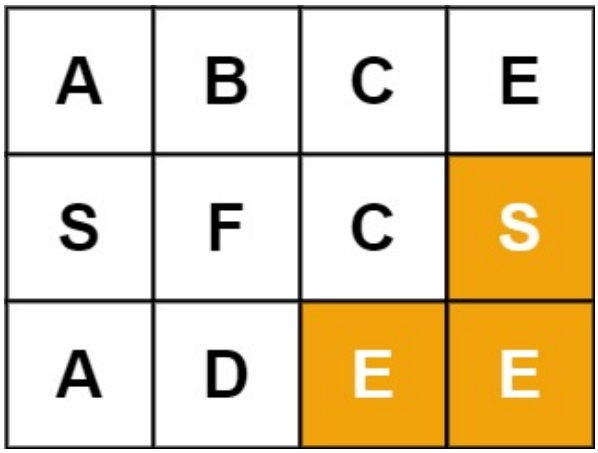
Output: `True`



Example 2:

Input: `board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "SEE"`

Output: `True`



Example 3:

Input: `board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCB"`

Output `False`

Constraints:

- `m == board.length`
- `n = board[i].length`
- `1 <= m, n <= 6`
- `board` and `word` consists of only lowercase and uppercase English letters.

A	B	C	E
S	F	C	S
A	D	E	E

Figure 1: Caption

Problem 3: Construct Quad Tree

Given a `n * n` matrix `grid` of `0's` and `1's` only. We want to pepresent `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;
   }
```

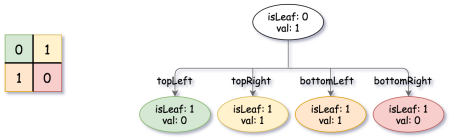
We can construct a Quad-Tree from a two-dimensional area using the following steps:

1. 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.
2. 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.
3. Recurse for each of the children with the proper sub-grid.

Example 1:

Input: `grid = [[0,1],[1,0]]`

Output: `[[0,1],[1,0],[1,1],[1,1],[1,0]]`



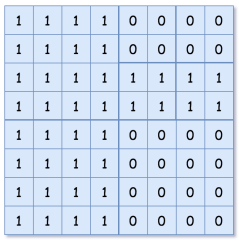
Explanation: The explanation of this example is shown below: Notice that 0 represents False and 1 represents True in the photo representing the Quad-Tree.

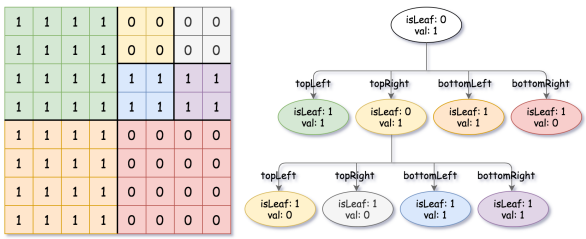
Example 2:

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,

Output: [[0,1],[1,1],[0,1],[1,1],[1,0],null,null,null,null,[1,0],[1,0],[1,1],[1,1]]

Explaination: 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