SCS
CONCORDIA

ALGORITHM TIME

Train your mind

Weekly sessions every Tuesdays at MB 3.285

1. Move Zeroes (Easy)

Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.
Note that you must do this in-place without making a copy of the array.

Example 1:
Input: nums = [0,1,0,3,12]
Output: [1,3,12,0,0]

Example 2:
Input: nums = [0]
Output: [0]

Constraints:
$1 <= nums.length <= 10^4$
$-2^{31} <= nums[i] <= 2^{31} - 1$

2. Intersection of Two Arrays (Easy)

Given two integer arrays nums1 and nums2, return *an array of their intersection*. Each element in the result must be **unique** and you may return the result in **any order**.

Example 1:
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2]

Example 2:
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [9,4]
Explanation: [4,9] is also accepted.

Constraints:
1 <= nums1.length, nums2.length <= 1000
0 <= nums1[i], nums2[i] <= 1000

3. Longest Palindrome (Easy)

Given a string s which consists of lowercase or uppercase letters, return *the length of the **longest palindrome*** that can be built with those letters.
Letters are **case sensitive**, for example, "Aa" is not considered a palindrome here.

Example 1:
Input: s = "abccccdd"
Output: 7
Explanation: One longest palindrome that can be built is "dccaccd", whose length is 7.

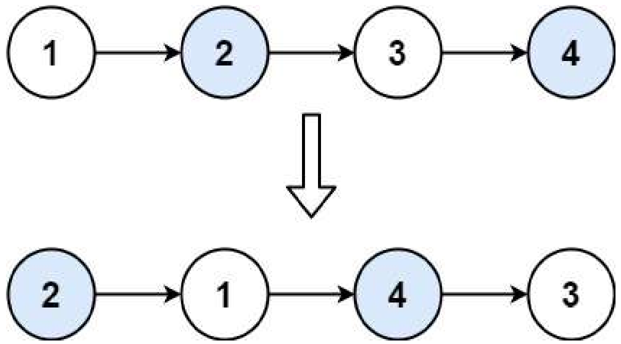
Example 2:
Input: s = "a"
Output: 1
Explanation: The longest palindrome that can be built is "a", whose length is 1.

Constraints:
1 <= s.length <= 2000
s consists of lowercase **and/or** uppercase English letters only.

4. Swap Nodes in Pairs (Medium)

Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed.)

Example 1:
Input: head = [1,2,3,4]
Output: [2,1,4,3]



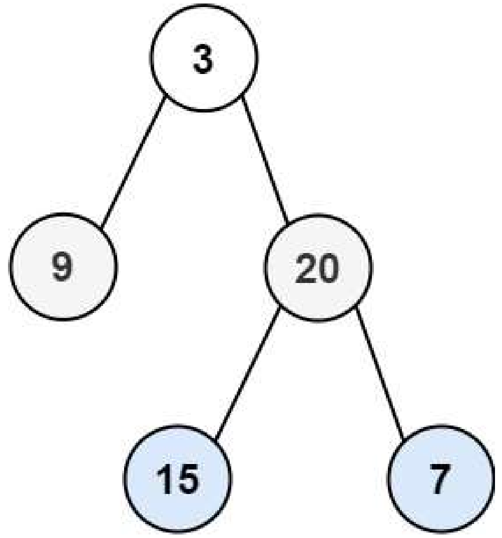
Example 2:
Input: head = []
Output: []

Example 3:
Input: head = [1]
Output: [1]

Constraints:
The number of nodes in the list is in the range [0, 100].
0 <= Node.val <= 100

5. Binary Tree Level Order Traversal II (Medium)

Given the `root` of a binary tree, return *the bottom-up level order traversal of its nodes' values*. (i.e., from left to right, level by level from leaf to root).



Example 1:

```
Input: root = [3,9,20,null,null,15,7]
Output: [[15,7],[9,20],[3]]
```

Example 2:

```
Input: root = [1]
Output: [[1]]
```

Example 3:

```
Input: root = []
Output: []
```

Constraints:

- The number of nodes in the tree is in the range `[0, 2000]`.
- `-1000 <= Node.val <= 1000`

6. Validate Stack Sequence (Medium)

Given two integer arrays `pushed` and `popped` each with distinct values, return `true` *if this could have been the result of a sequence of push and pop operations on an initially empty stack, or* `false` *otherwise.*

Example 1:

```
Input: pushed = [1,2,3,4,5], popped = [4,5,3,2,1]
Output: true
Explanation: We might do the following sequence:
push(1), push(2), push(3), push(4),
pop() -> 4,
push(5),
pop() -> 5, pop() -> 3, pop() -> 2, pop() -> 1
```

Example 2:

```
Input: pushed = [1,2,3,4,5], popped = [4,3,5,1,2]
Output: false
Explanation: 1 cannot be popped before 2.
```

Constraints:

- `1 <= pushed.length <= 1000`
- `0 <= pushed[i] <= 1000`
- All the elements of `pushed` are **unique**.
- `popped.length == pushed.length`
- `popped` is a permutation of `pushed`.

7. Recover a Tree From Preorder Traversal (Hard)

We run a preorder depth-first search (DFS) on the root of a binary tree.
At each node in this traversal, we output D dashes (where D is the depth of this node), then we output the value of this node. If the depth of a node is D, the depth of its immediate child is D + 1. The depth of the root node is 0.
If a node has only one child, that child is guaranteed to be **the left child**.
Given the output traversal of this traversal, recover the tree and return *its* root.

Example 1:

```
Input: traversal = "1-2--3--4-5--6--7"
Output: [1,2,5,3,4,6,7]
```

Example 2:

```
Input: traversal = "1-2--3---4-5--6---7"
Output: [1,2,5,3,null,6,null,4,null,7]
```

Example 3:

```
Input: traversal = "1-401--349---90--88"
Output: [1,401,null,349,88,90]
```

Constraints:

- The number of nodes in the original tree is in the range `[1, 1000]`.
- $1 <= Node.val <= 10^9$

8. Count All Valid Pickup and Delivery Options (Hard)

Given n orders, each order consist in pickup and delivery services.
Count all valid pickup/delivery possible sequences such that delivery(i) is always after of pickup(i).
Since the answer may be too large, return it modulo 10^9 + 7.

Example 1:
Input: n = 1
Output: 1
Explanation: Unique order (P1, D1), Delivery 1 always is after of Pickup 1.
Example 2:
Input: n = 2
Output: 6
Explanation: All possible orders: (P1,P2,D1,D2), (P1,P2,D2,D1), (P1,D1,P2,D2), (P2,P1,D1,D2), (P2,P1,D2,D1) and (P2,D2,P1,D1).This is an invalid order (P1,D2,P2,D1) because Pickup 2 is after of Delivery 2.
Example 3:
Input: n = 3
Output: 90

Constraints:

- 1 <= n <= 500

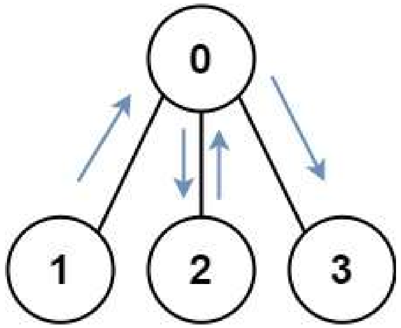
9. Shortest Path Visiting All Nodes

You have an undirected, connected graph of `n` nodes labeled from `0` to `n - 1`. You are given an array `graph` where `graph[i]` is a list of all the nodes connected with node `i` by an edge.

Return *the length of the shortest path that visits every node*. You may start and stop at any node, you may revisit nodes multiple times, and you may reuse edges.

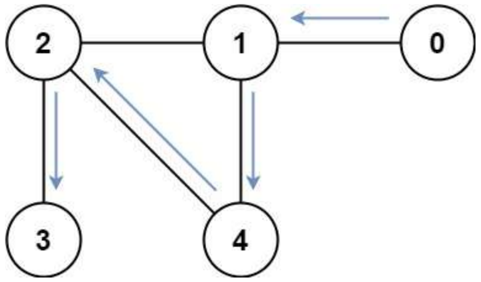
Example 1:

```
Input: graph = [[1,2,3],[0],[0],[0]]
Output: 4
Explanation: One possible path is [1,0,2,0,3]
```



Example 2:

```
Input: graph = [[1],[0,2,4],[1,3,4],[2],[1,2]]
Output: 4
Explanation: One possible path is [0,1,4,2,3]
```



Constraints:

- `n == graph.length`
- `1 <= n <= 12`
- `0 <= graph[i].length < n`
- `graph[i]` does not contain `i`.
- If `graph[a]` contains `b`, then `graph[b]` contains `a`.
- The input graph is always connected.

Links for Question #:

1. https://leetcode.com/problems/move-zeroes/
2. https://leetcode.com/problems/intersection-of-two-arrays/
3. https://leetcode.com/problems/longest-palindrome/
4. https://leetcode.com/problems/swap-nodes-in-pairs/
5. https://leetcode.com/problems/binary-tree-level-order-traversal/
6. https://leetcode.com/problems/validate-stack-sequences/
7. https://leetcode.com/problems/recover-a-tree-from-preorder-traversal/
8. https://leetcode.com/problems/count-all-valid-pickup-and-delivery-options/
9. https://leetcode.com/problems/shortest-path-visiting-all-nodes/