- Meeting Rooms II (Leetcode Premium){#toc-meeting-rooms-ii-leetcode-premium}
- Reverse a Linked List{#toc-reverse-a-linked-list}
- Detect Cycle in a Linked List{#toc-detect-cycle-in-a-linked-list}
- Merge Two Sorted Lists{#toc-merge-two-sorted-lists}
- Merge K Sorted Lists{#toc-merge-k-sorted-lists}
- Remove Nth Node From End Of List{#toc-remove-nth-node-from-end-of-list}
- Reorder List{#toc-reorder-list}
- Set Matrix Zeroes{#toc-set-matrix-zeroes}
- Spiral Matrix{#toc-spiral-matrix}
- Rotate Image{#toc-rotate-image}
- Word Search{#toc-word-search}
- Longest Substring Without Repeating Characters{#toc-longest-substring-without-repeating-characters}
- Longest Repeating Character Replacement{#toc-longest-repeating-character-replacement}
- Minimum Window Substring{#toc-minimum-window-substring}
- Valid Anagram{#toc-valid-anagram}
- Group Anagrams{#toc-group-anagrams}
- Valid Parentheses{#toc-valid-parentheses}
- Valid Palindrome{#toc-valid-palindrome}
- Longest Palindromic Substring{#toc-longest-palindromic-substring}
- Palindromic Substrings{#toc-palindromic-substrings}
- Encode and Decode Strings (Leetcode Premium){#toc-encode-and-decode-strings-leetcode-premium}
- Maximum Depth of Binary Tree{#toc-maximum-depth-of-binary-tree}
- Same Tree{#toc-same-tree}
- Invert/Flip Binary Tree{#toc-invertflip-binary-tree}
- Binary Tree Maximum Path Sum{#toc-binary-tree-maximum-path-sum}
- Binary Tree Level Order Traversal{#toc-binary-tree-level-order-traversal}
- Serialize and Deserialize Binary Tree{#toc-serialize-and-deserialize-binary-tree}
- Subtree of Another Tree{#toc-subtree-of-another-tree}
- Construct Binary Tree from Preorder and Inorder Traversal{#toc-construct-binary-tree-from-preorder-and-inorder-traversal}
- Validate Binary Search Tree{#toc-validate-binary-search-tree}
- Kth Smallest Element in a BST{#toc-kth-smallest-element-in-a-bst}
- Lowest Common Ancestor of BST{#toc-lowest-common-ancestor-of-bst}
- Implement Trie (Prefix Tree){#toc-implement-trie-prefix-tree}
- Add and Search Word{#toc-add-and-search-word}
- Word Search II{#toc-word-search-ii}
- Merge K Sorted Lists{#toc-merge-k-sorted-lists-1}
- Top K Frequent Elements{#toc-top-k-frequent-elements}
- Find Median from Data Stream{#toc-find-median-from-data-stream}

# Two Sum

**Description**

---

Category Arrays Video https://youtu.be/KLlXCFG5TnA
Link https://leetcode.com/problems/two-sum

---

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to `target`*.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

**Example 1:**

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

**Example 2:**

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

**Example 3:**

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

**Constraints:**

- `2 <= nums.length <= 104`
- `-109 <= nums[i] <= 109`
- `-109 <= target <= 109`
- **Only one valid answer exists.**

**Follow-up:**Can you come up with an algorithm that is less than `O(n2)` time complexity? **Hint** use hash map to instantly check for difference value, map will add index of last occurrence of a num, don't use same element twice;

# Best Time to Buy and Sell Stock

**Description**

---

Category Arrays Video https://youtu.be/1pkOgXD63yU
Link https://leetcode.com/problems/best-time-to-buy-and-sell-stock

---

You are given an array `prices` where `prices[i]` is the price of a given stock on the `ith` day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return `0`.

**Example 1:**

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit =
6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must
buy before you sell.
```

**Example 2:**

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.
```

**Constraints:**

- `1 <= prices.length <= 105`
- `0 <= prices[i] <= 104`

**Hint** find local min and search for local max, sliding window;

# Contains Duplicate

**Description**

---

Category Arrays Video https://youtu.be/3OamzN90kPg
Link https://leetcode.com/problems/contains-duplicate

---

Given an integer array `nums`, return `true` if any value appears **at least twice** in the array, and return `false` if every element is distinct.

**Example 1:**

**Input:** nums = [1,2,3,1]

**Output:** true

**Explanation:**

The element 1 occurs at the indices 0 and 3.

**Example 2:**

**Input:** nums = [1,2,3,4]

**Output:** false

**Explanation:**

All elements are distinct.

**Example 3:**

**Input:** nums = [1,1,1,3,3,4,3,2,4,2]

**Output:** true

**Constraints:**

- `1 <= nums.length <= 105`
- `-109 <= nums[i] <= 109`

**Hint** hashset to get unique values in array, to check for duplicates easily

# Product of Array Except Self

**Description**

Category Arrays Video https://youtu.be/bNvIQI2wAjk
Link https://leetcode.com/problems/product-of-array-except-self

Given an integer array `nums`, return *an array* `answer` *such that* `answer[i]` *is equal to the product of all the elements of* `nums` *except* `nums[i]`.

The product of any prefix or suffix of `nums` is **guaranteed** to fit in a **32-bit** integer.

You must write an algorithm that runs in `O(n)` time and without using the division operation.

**Example 1:**

```
Input: nums = [1,2,3,4]
Output: [24,12,8,6]
```

**Example 2:**

```
Input: nums = [-1,1,0,-3,3]
Output: [0,0,9,0,0]
```

**Constraints:**

- `2 <= nums.length <= 105`
- `-30 <= nums[i] <= 30`
- The product of any prefix or suffix of `nums` is **guaranteed** to fit in a **32-bit** integer.

**Follow up:** Can you solve the problem in `O(1)` extra space complexity? (The output array **does not** count as extra space for space complexity analysis.)

**Hint** make two passes, first in-order, second in-reverse, to compute products

# Maximum Subarray

**Description**

Category Arrays Video https://youtu.be/5WZl3MMT0Eg
Link https://leetcode.com/problems/maximum-subarray

---

Given an integer array `nums`, find the subarray with the largest sum, and return *its sum*.

**Example 1:**

```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.
```

**Example 2:**

```
Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.
```

**Example 3:**

```
Input: nums = [5,4,-1,7,8]
Output: 23
Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.
```

**Constraints:**

- `1 <= nums.length <= 105`
- `-104 <= nums[i] <= 104`

**Follow up:** If you have figured out the `O(n)` solution, try coding another solution using the **divide and conquer** approach, which is more subtle.

**Hint** pattern: prev subarray cant be negative, dynamic programming: compute max sum for each prefix

# Maximum Product Subarray

---

**Description**

---

Category Arrays Video https://youtu.be/lXVy6YWFcRM
Link https://leetcode.com/problems/maximum-product-subarray

---

Given an integer array `nums`, find a subarray that has the largest product, and return *the product*.

The test cases are generated so that the answer will fit in a **32-bit** integer.

**Example 1:**

```
Input: nums = [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.
```

**Example 2:**

```
Input: nums = [-2,0,-1]
Output: 0
Explanation: The result cannot be 2, because [-2,-1] is not a subarray.
```

**Constraints:**

- `1 <= nums.length <= 2 * 104`
- `-10 <= nums[i] <= 10`
- The product of any subarray of `nums` is **guaranteed** to fit in a **32-bit** integer.

**Hint** dp: compute max and max-abs-val for each prefix subarr;

# Find Minimum in Rotated Sorted Array

**Description**

---

Category Arrays Video https://youtu.be/nIVW4P8b1VA
Link https://leetcode.com/problems/find-minimum-in-rotated-sorted-array

---

Suppose an array of length `n` sorted in ascending order is **rotated** between `1` and `n` times. For example, the array `nums = [0,1,2,4,5,6,7]` might become:

- `[4,5,6,7,0,1,2]` if it was rotated `4` times.
- `[0,1,2,4,5,6,7]` if it was rotated `7` times.

Notice that **rotating** an array `[a[0], a[1], a[2], ..., a[n-1]]` 1 time results in the array `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]`.

Given the sorted rotated array `nums` of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in `O(log n) time`.

**Example 1:**

```
Input: nums = [3,4,5,1,2]
Output: 1
Explanation: The original array was [1,2,3,4,5] rotated 3 times.
```

**Example 2:**

```
Input: nums = [4,5,6,7,0,1,2]
Output: 0
Explanation: The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.
```

**Example 3:**

```
Input: nums = [11,13,15,17]
Output: 11
Explanation: The original array was [11,13,15,17] and it was rotated 4 times.
```

**Constraints:**

- `n == nums.length`
- `1 <= n <= 5000`
- `-5000 <= nums[i] <= 5000`
- All the integers of `nums` are **unique**.
- `nums` is sorted and rotated between `1` and `n` times.

**Hint** check if half of array is sorted in order to find pivot, arr is guaranteed to be in at most two sorted subarrays

# Search in Rotated Sorted Array

**Description**

---

Category Arrays Video https://youtu.be/U8XENwh8Oy8
Link https://leetcode.com/problems/search-in-rotated-sorted-array

---

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` (`1 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`, *or* `-1` *if it is not in* `nums`.

You must write an algorithm with `O(log n)` runtime complexity.

**Example 1:**

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

**Example 2:**

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

**Example 3:**

```
Input: nums = [1], target = 0
Output: -1
```

**Constraints:**

- `1 <= nums.length <= 5000`
- `-104 <= nums[i] <= 104`
- All values of `nums` are **unique**.
- `nums` is an ascending array that is possibly rotated.
- `-104 <= target <= 104`

**Hint** at most two sorted halfs, mid will be apart of left sorted or right sorted, if target is in range of sorted portion then search it, otherwise search other half

# 3Sum {#3sum}

**Description**

---

Category Arrays Video https://youtu.be/jzZsG8n2R9A
Link https://leetcode.com/problems/3sum

---

Given an integer array nums, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j`, `i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

Notice that the solution set must not contain duplicate triplets.

**Example 1:**

```
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not
matter.
```

**Example 2:**

```
Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.
```

**Example 3:**

```
Input: nums = [0,0,0]
Output: [[0,0,0]]
Explanation: The only possible triplet sums up to 0.
```

**Constraints:**

- `3 <= nums.length <= 3000`
- `-105 <= nums[i] <= 105`

**Hint** sort input, for each first element, find next two where -a = b+c, if a=prevA, skip a, if b=prevB skip b to elim duplicates; to find b,c use two pointers, left/right on remaining list;

# Container With Most Water

**Description**

---

Category Arrays Video https://youtu.be/UuiTKBwPgAo
Link https://leetcode.com/problems/container-with-most-water

---

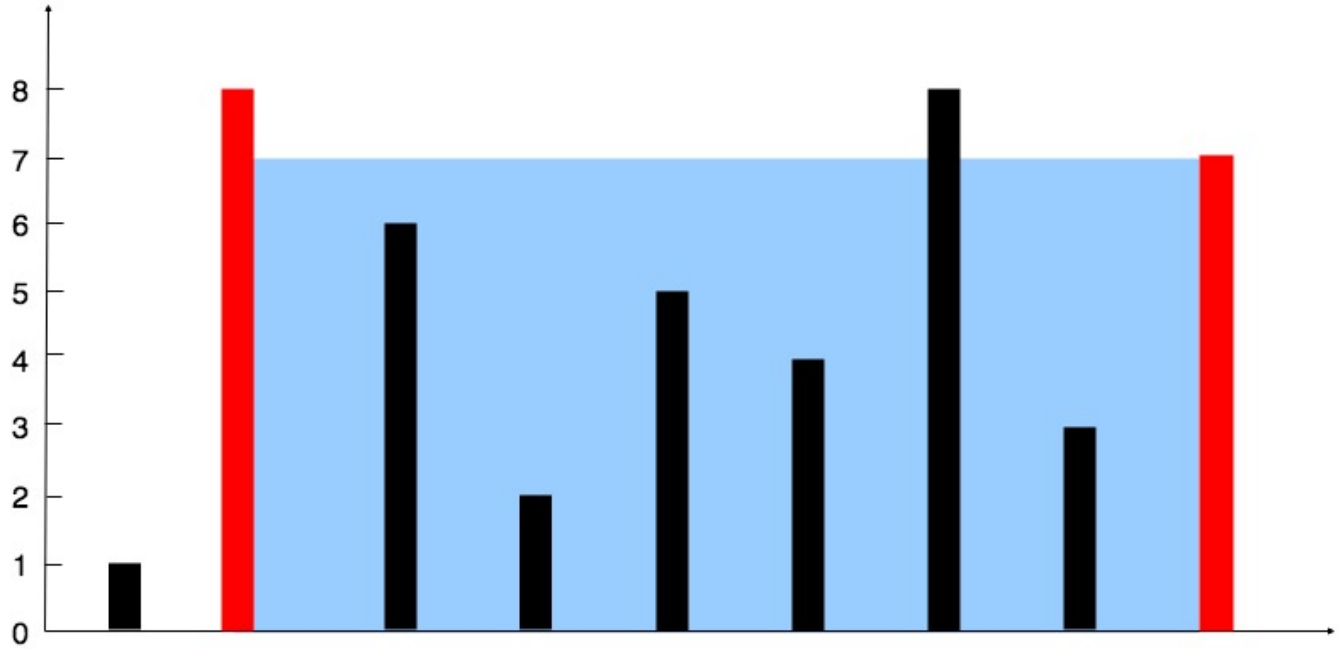
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `ith` line are `(i, 0)` and `(i, height[i])`.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

**Notice** that you may not slant the container.

**Example 1:**




{ width=350 }

```
Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array
[1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the
container can contain is 49.
```

**Example 2:**

```
Input: height = [1,1]
Output: 1
```

**Constraints:**

- `n == height.length`
- `2 <= n <= 105`
- `0 <= height[i] <= 104`

**Hint** shrinking window, left/right initially at endpoints, shift the pointer with min height;

# Sum of Two Integers

**Description**

---

Category Binary Video https://youtu.be/gVUrDV4tZfY
Link https://leetcode.com/problems/sum-of-two-integers

---

Given two integers `a` and `b`, return *the sum of the two integers without using the operators* `+` *and* `-`.

**Example 1:**

```
Input: a = 1, b = 2
Output: 3
```

**Example 2:**

```
Input: a = 2, b = 3
Output: 5
```

**Constraints:**

- `-1000 <= a, b <= 1000`

**Hint** add bit by bit, be mindful of carry, after adding, if carry is still 1, then add it as well;

# Number of 1 Bits

---

**Description**

---

Category Binary Video https://youtu.be/5Km3utixwZs
Link https://leetcode.com/problems/number-of-1-bits

---

Given a positive integer `n`, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

**Example 1:**

**Input:** n = 11

**Output:** 3

**Explanation:**

The input binary string **1011** has a total of three set bits.

**Example 2:**

**Input:** n = 128

**Output:** 1

**Explanation:**

The input binary string **10000000** has a total of one set bit.

**Example 3:**

**Input:** n = 2147483645

**Output:** 30

**Explanation:**

The input binary string **11111111111111111111111111111101** has a total of thirty set bits.

**Constraints:**

- `1 <= n <= 231 - 1`

**Follow up:** If this function is called many times, how would you optimize it? **Hint** modulo, and dividing n; mod and div are expensive, to divide use bit shift, instead of mod to get 1's place use bitwise & 1;

# Counting Bits

**Description**

---

Category Binary Video https://youtu.be/RyBM56RIWrM
Link https://leetcode.com/problems/counting-bits

---

Given an integer `n`, return *an array* `ans` *of length* `n + 1` *such that for each* `i`(`0 <= i <= n`), `ans[i]` *is the* ***number of*** `1`***'s*** *in the binary representation of* `i`.

**Example 1:**

```
Input: n = 2
Output: [0,1,1]
Explanation:
0 --> 0
1 --> 1
2 --> 10
```

**Example 2:**

```
Input: n = 5
Output: [0,1,1,2,1,2]
Explanation:
0 --> 0
1 --> 1
2 --> 10
3 --> 11
4 --> 100
5 --> 101
```

**Constraints:**

- `0 <= n <= 105`

**Follow up:**

- It is very easy to come up with a solution with a runtime of `O(n log n)`. Can you do it in linear time `O(n)` and possibly in a single pass?
- Can you do it without using any built-in function (i.e., like `__builtin_popcount` in C++)?

**Hint** write out result for num=16 to figure out pattern; res[i] = res[i - offset], where offset is the biggest power of 2 <= I;

# Missing Number

**Description**

---

Category Binary Video https://youtu.be/WnPLSRLSANE
Link https://leetcode.com/problems/missing-number

---

Given an array `nums` containing `n` distinct numbers in the range `[0, n]`, return *the only number in the range that is missing from the array.*

**Example 1:**

```
Input: nums = [3,0,1]
Output: 2
Explanation: n = 3 since there are 3 numbers, so all numbers are in the range
[0,3]. 2 is the missing number in the range since it does not appear in nums.
```

**Example 2:**

```
Input: nums = [0,1]
Output: 2
Explanation: n = 2 since there are 2 numbers, so all numbers are in the range
[0,2]. 2 is the missing number in the range since it does not appear in nums.
```

**Example 3:**

```
Input: nums = [9,6,4,2,3,5,7,0,1]
Output: 8
Explanation: n = 9 since there are 9 numbers, so all numbers are in the range
[0,9]. 8 is the missing number in the range since it does not appear in nums.
```

**Constraints:**

- `n == nums.length`
- `1 <= n <= 104`
- `0 <= nums[i] <= n`
- All the numbers of `nums` are **unique**.

**Follow up:** Could you implement a solution using only `O(1)` extra space complexity and `O(n)` runtime complexity?

**Hint** compute expected sum - real sum; xor n with each index and value;

# Reverse Bits

**Description**

---

Category Binary Video https://youtu.be/UcoN6UjAI64
Link https://leetcode.com/problems/reverse-bits

---

Reverse bits of a given 32 bits unsigned integer.

**Note:**

- Note that in some languages, such as Java, there is no unsigned integer type. In this case, both input and output will be given as a signed integer type. They should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in **Example 2** above, the input represents the signed integer `-3` and the output represents the signed integer `-1073741825`.

**Example 1:**

```
Input: n = 00000010100101000001111010011100
Output:    964176192 (00111001011110000010100101000000)
Explanation: The input binary string 00000010100101000001111010011100
represents the unsigned integer 43261596, so return 964176192 which its binary
representation is 00111001011110000010100101000000.
```

**Example 2:**

```
Input: n = 11111111111111111111111111111101
Output:   3221225471 (10111111111111111111111111111111)
Explanation: The input binary string 11111111111111111111111111111101
represents the unsigned integer 4294967293, so return 3221225471 which its
binary representation is 10111111111111111111111111111111.
```

**Constraints:**

- The input must be a **binary string** of length `32`

**Follow up:** If this function is called many times, how would you optimize it?

**Hint** reverse each of 32 bits;

# Climbing Stairs

**Description**

---

Category Dynamic Programming Video https://youtu.be/Y0lT9Fck7qI
Link https://leetcode.com/problems/climbing-stairs

---

You are climbing a staircase. It takes `n` steps to reach the top.

Each time you can either climb `1` or `2` steps. In how many distinct ways can you climb to the top?

**Example 1:**

```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

**Example 2:**

```
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```

**Constraints:**

- `1 <= n <= 45`

**Hint** subproblem find (n-1) and (n-2), sum = n;

# Coin Change

**Description**

---

Category Dynamic Programming Video https://youtu.be/H9bfqozjoqs
Link https://leetcode.com/problems/coin-change

---

You are given an integer array `coins` representing coins of different denominations and an integer `amount` representing a total amount of money.

Return *the fewest number of coins that you need to make up that amount*. If that amount of money cannot be made up by any combination of the coins, return `-1`.

You may assume that you have an infinite number of each kind of coin.

**Example 1:**

```
Input: coins = [1,2,5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1
```

**Example 2:**

```
Input: coins = [2], amount = 3
Output: -1
```

**Example 3:**

```
Input: coins = [1], amount = 0
Output: 0
```

**Constraints:**

- `1 <= coins.length <= 12`
- `1 <= coins[i] <= 231 - 1`
- `0 <= amount <= 104`

**Hint** top-down: recursive dfs, for amount, branch for each coin, cache to store prev coin_count for each amount; bottom-up: compute coins for amount = 1, up until n, using for each coin (amount - coin), cache prev values

# Longest Increasing Subsequence

**Description**

---

Category Dynamic Programming Video https://youtu.be/cjWnW0hdF1Y
Link https://leetcode.com/problems/longest-increasing-subsequence

---

Given an integer array `nums`, return *the length of the longest **strictly increasing subsequence***.

**Example 1:**

```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the
length is 4.
```

**Example 2:**

```
Input: nums = [0,1,0,3,2,3]
Output: 4
```

**Example 3:**

```
Input: nums = [7,7,7,7,7,7,7]
Output: 1
```

**Constraints:**

- `1 <= nums.length <= 2500`
- `-104 <= nums[i] <= 104`

**Follow up:** Can you come up with an algorithm that runs in `O(n log(n))` time complexity?

**Hint** recursive: foreach num, get subseq with num and without num, only include num if prev was less, cache solution of each; dp=subseq length which must end with each num, curr num must be after a prev dp or by itself;

# Longest Common Subsequence

**Description**

---

Category Dynamic Programming Video https://youtu.be/Ua0GhsJSlWM
Link https://leetcode.com/problems/longest-common-subsequence

---

Given two strings `text1` and `text2`, return *the length of their longest* ***common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings.

**Example 1:**

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.
```

**Example 2:**

```
Input: text1 = "abc", text2 = "abc"
Output: 3
Explanation: The longest common subsequence is "abc" and its length is 3.
```

**Example 3:**

```
Input: text1 = "abc", text2 = "def"
Output: 0
Explanation: There is no such common subsequence, so the result is 0.
```

**Constraints:**

- `1 <= text1.length, text2.length <= 1000`
- `text1` and `text2` consist of only lowercase English characters.

**Hint** recursive: if first chars are equal find lcs of remaining of each, else max of: lcs of first and remain of 2nd and lcs of 2nd remain of first, cache result; nested forloop to compute the cache without recursion;

# Word Break Problem

---

**Description**

---

Category Dynamic Programming Video https://youtu.be/Sx9NNgInc3A
Link https://leetcode.com/problems/word-break

---

Given a string `s` and a dictionary of strings `wordDict`, return `true` if `s` can be segmented into a space-separated sequence of one or more dictionary words.

**Note** that the same word in the dictionary may be reused multiple times in the segmentation.

**Example 1:**

```
Input: s = "leetcode", wordDict = ["leet","code"]
Output: true
Explanation: Return true because "leetcode" can be segmented as "leet code".
```

**Example 2:**

```
Input: s = "applepenapple", wordDict = ["apple","pen"]
Output: true
Explanation: Return true because "applepenapple" can be segmented as "apple pen
apple".
Note that you are allowed to reuse a dictionary word.
```

**Example 3:**

```
Input: s = "catsandog", wordDict = ["cats","dog","sand","and","cat"]
Output: false
```

**Constraints:**

- `1 <= s.length <= 300`
- `1 <= wordDict.length <= 1000`
- `1 <= wordDict[i].length <= 20`
- `s` and `wordDict[i]` consist of only lowercase English letters.
- All the strings of `wordDict` are **unique**.

**Hint** for each prefix, if prefix is in dict and wordbreak(remaining str)=True, then return True, cache result of wordbreak;

# Combination Sum

**Description**

---

Category Dynamic Programming Video https://youtu.be/GBKI9VSKdGg
Link https://leetcode.com/problems/combination-sum

---

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all* ***unique combinations*** *of* `candidates` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

**Example 1:**

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple
times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

**Example 2:**

```
Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]
```

**Example 3:**

```
Input: candidates = [2], target = 1
Output: []
```

**Constraints:**

- `1 <= candidates.length <= 30`
- `2 <= candidates[i] <= 40`
- All elements of `candidates` are **distinct**.
- `1 <= target <= 40`

**Hint** visualize the decision tree, base case is curSum = or > target, each candidate can have children of itself or elements to right of it inorder to elim duplicate solutions;

# House Robber

**Description**

---

Category Dynamic Programming Video https://youtu.be/73r3KWiEvyk
Link https://leetcode.com/problems/house-robber

---

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight **without alerting the police***.

**Example 1:**

```
Input: nums = [1,2,3,1]
Output: 4
```

```
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
```

**Example 2:**

```
Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5
(money = 1).
Total amount you can rob = 2 + 9 + 1 = 12.
```

**Constraints:**

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 400`

**Hint** for each num, get max of prev subarr, or num + prev subarr not including last element, store results of prev, and prev not including last element

# House Robber II

**Description**

---

Category Dynamic Programming Video https://youtu.be/rWAJCfYYOvM
Link https://leetcode.com/problems/house-robber-ii

---

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are **arranged in a circle.** That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have a security system connected, and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight **without alerting the police***.

**Example 1:**

```
Input: nums = [2,3,2]
Output: 3
```

```
Explanation: You cannot rob house 1 (money = 2) and then rob house 3 (money =
2), because they are adjacent houses.
```

**Example 2:**

```
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
```

**Example 3:**

```
Input: nums = [1,2,3]
Output: 3
```

**Constraints:**

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 1000`

**Hint** subarr = arr without first & last, get max of subarr, then pick which of first/last should be added to it

# Decode Ways

**Description**

---

Category Dynamic Programming Video https://youtu.be/6aEyTjOwlJU
Link https://leetcode.com/problems/decode-ways

---

You have intercepted a secret message encoded as a string of numbers. The message is **decoded** via the following mapping:

`"1" -> 'A'

"2" -> 'B'

...

"25" -> 'Y'

"26" -> 'Z'`

However, while decoding the message, you realize that there are many different ways you can decode the message because some codes are contained in other codes (`"2"` and `"5"` vs `"25"`).

For example, `"11106"` can be decoded into:

- `"AAJF"` with the grouping `(1, 1, 10, 6)`
- `"KJF"` with the grouping `(11, 10, 6)`
- The grouping `(1, 11, 06)` is invalid because `"06"` is not a valid code (only `"6"` is valid).

Note: there may be strings that are impossible to decode.

Given a string s containing only digits, return the **number of ways** to **decode** it. If the entire string cannot be decoded in any valid way, return `0`.

The test cases are generated so that the answer fits in a **32-bit** integer.

**Example 1:**

**Input:** s = "12"

**Output:** 2

**Explanation:**

"12" could be decoded as "AB" (1 2) or "L" (12).

**Example 2:**

**Input:** s = "226"

**Output:** 3

**Explanation:**

"226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

**Example 3:**

**Input:** s = "06"

**Output:** 0

**Explanation:**

"06" cannot be mapped to "F" because of the leading zero ("6" is different from "06"). In this case, the string is not a valid encoding, so return 0.

**Constraints:**

- `1 <= s.length <= 100`
- `s` contains only digits and may contain leading zero(s).

**Hint** can cur char be decoded in one or two ways? Recursion -> cache -> iterative dp solution, a lot of edge cases to determine, 52, 31, 29, 10, 20 only decoded one way, 11, 26 decoded two ways

# Unique Paths

**Description**

---

Category Dynamic Programming Video https://youtu.be/IlEsdxuD4lY
Link https://leetcode.com/problems/unique-paths

---

There is a robot on an `m x n` grid. The robot is initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

Given the two integers `m` and `n`, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The test cases are generated so that the answer will be less than or equal to `2 * 109`.

**Example 1:**

{ width=350 }

```
Input: m = 3, n = 7
Output: 28
```

**Example 2:**

```
Input: m = 3, n = 2
Output: 3
Explanation: From the top-left corner, there are a total of 3 ways to reach the
bottom-right corner:
1. Right -> Down -> Down
2. Down -> Down -> Right
3. Down -> Right -> Down
```

**Constraints:**

- `1 <= m, n <= 100`

**Hint** work backwards from solution, store paths for each position in grid, to further optimize, we don't store whole grid, only need to store prev row;

# Jump Game

**Description**

---

Category Dynamic Programming Video https://youtu.be/Yan0cv2cLy8
Link https://leetcode.com/problems/jump-game

---

You are given an integer array `nums`. You are initially positioned at the array's **first index**, and each element in the array represents your maximum jump length at that position.

Return `true` *if you can reach the last index, or* `false` *otherwise*.

**Example 1:**

```
Input: nums = [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.
```

**Example 2:**

```
Input: nums = [3,2,1,0,4]
Output: false
Explanation: You will always arrive at index 3 no matter what. Its maximum jump
length is 0, which makes it impossible to reach the last index.
```

**Constraints:**

- `1 <= nums.length <= 104`
- `0 <= nums[i] <= 105`

**Hint** visualize the recursive tree, cache solution for O(n) time/mem complexity, iterative is O(1) mem, just iterate backwards to see if element can reach goal node, if yes, then set it equal to goal node, continue;

# Clone Graph

**Description**

---

Category Graph Video https://youtu.be/mQeF6bN8hMk
Link https://leetcode.com/problems/clone-graph

---

Given a reference of a node in a **connected** undirected graph.

Return a **deep copy** (clone) of the graph.

Each node in the graph contains a value (`int`) and a list (`List[Node]`) of its neighbors.

```
class Node {
    public int val;
    public List<Node> neighbors;
}
```

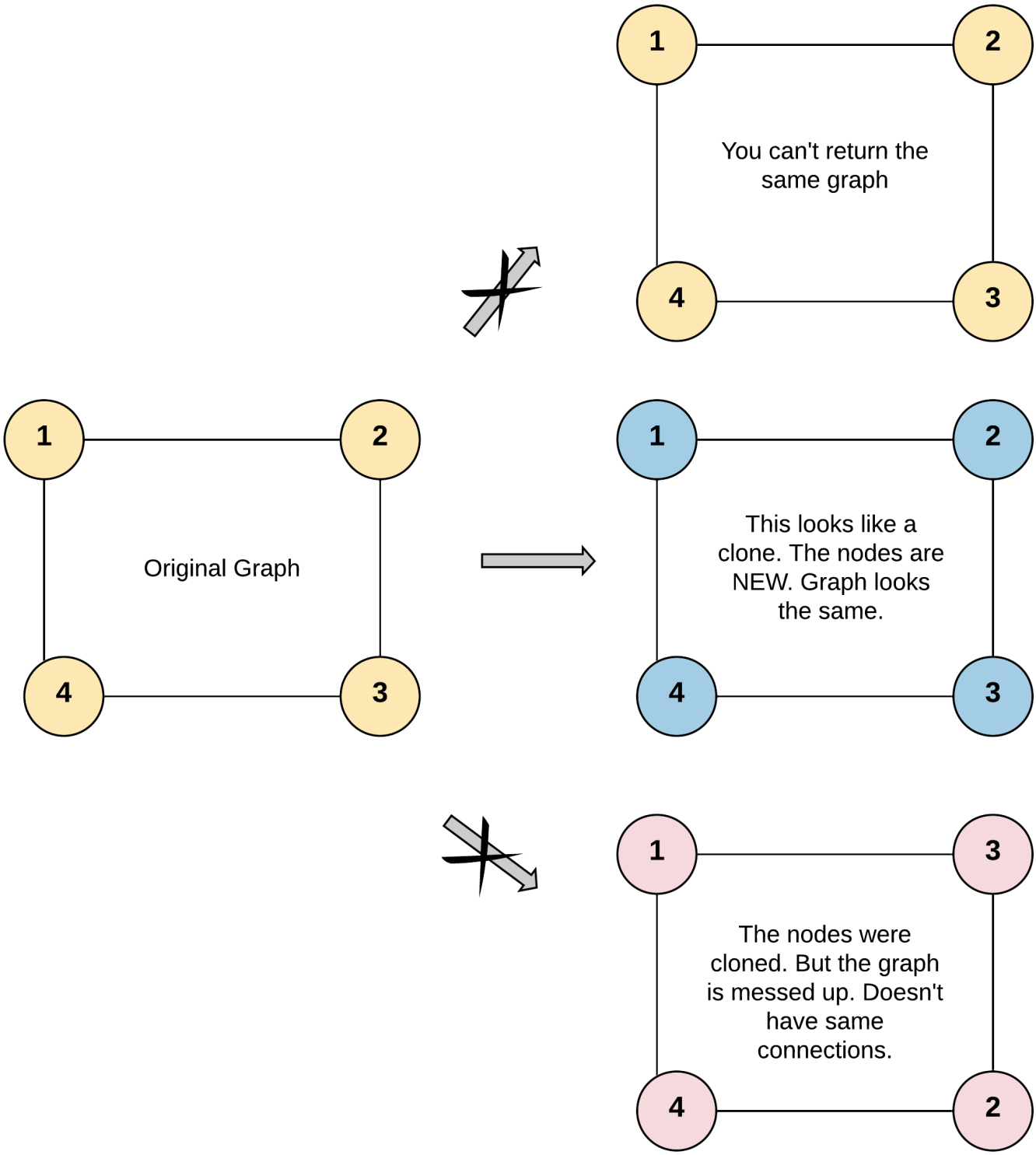
**Test case format:**

For simplicity, each node's value is the same as the node's index (1-indexed). For example, the first node with `val == 1`, the second node with `val == 2`, and so on. The graph is represented in the test case using an adjacency list.

**An adjacency list** is a collection of unordered **lists** used to represent a finite graph. Each list describes the set of neighbors of a node in the graph.

The given node will always be the first node with `val = 1`. You must return the **copy of the given node** as a reference to the cloned graph.

**Example 1:**




{ width=65% }

```
Input: adjList = [[2,4],[1,3],[2,4],[1,3]]
Output: [[2,4],[1,3],[2,4],[1,3]]
Explanation: There are 4 nodes in the graph.
1st node (val = 1)'s neighbors are 2nd node (val = 2) and 4th node (val = 4).
2nd node (val = 2)'s neighbors are 1st node (val = 1) and 3rd node (val = 3).
3rd node (val = 3)'s neighbors are 2nd node (val = 2) and 4th node (val = 4).
4th node (val = 4)'s neighbors are 1st node (val = 1) and 3rd node (val = 3).
```

**Example 2:**


```
Input: adjList = [[]]
Output: [[]]
Explanation: Note that the input contains one empty list. The graph consists of
only one node with val = 1 and it does not have any neighbors.
```

**Example 3:**

```
Input: adjList = []
Output: []
Explanation: This an empty graph, it does not have any nodes.
```

**Constraints:**

- The number of nodes in the graph is in the range `[0, 100]`.
- `1 <= Node.val <= 100`
- `Node.val` is unique for each node.
- There are no repeated edges and no self-loops in the graph.
- The Graph is connected and all nodes can be visited starting from the given node.

**Hint** recursive dfs, hashmap for visited nodes

# Course Schedule

**Description**

Category Graph Video https://youtu.be/EgI5nU9etnU
Link https://leetcode.com/problems/course-schedule

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [ai, bi]` indicates that you **must** take course `bi` first if you want to take course `ai`.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return `true` if you can finish all courses. Otherwise, return `false`.

**Example 1:**

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0. So it is possible.
```

**Example 2:**

```
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0, and to take course 0 you
should also have finished course 1. So it is impossible.
```

**Constraints:**

- `1 <= numCourses <= 2000`
- `0 <= prerequisites.length <= 5000`
- `prerequisites[i].length == 2`
- `0 <= ai, bi < numCourses`

- All the pairs prerequisites[i] are **unique**.

**Hint** build adjacentcy_list with edges, run dfs on each V, if while dfs on V we see V again, then loop exists, otherwise V isnt in a loop, 3 states= not visited, visited, still visiting

# Pacific Atlantic Water Flow

---

**Description**

---

Category Graph Video https://youtu.be/s-VkcjHqkGI
Link https://leetcode.com/problems/pacific-atlantic-water-flow

---

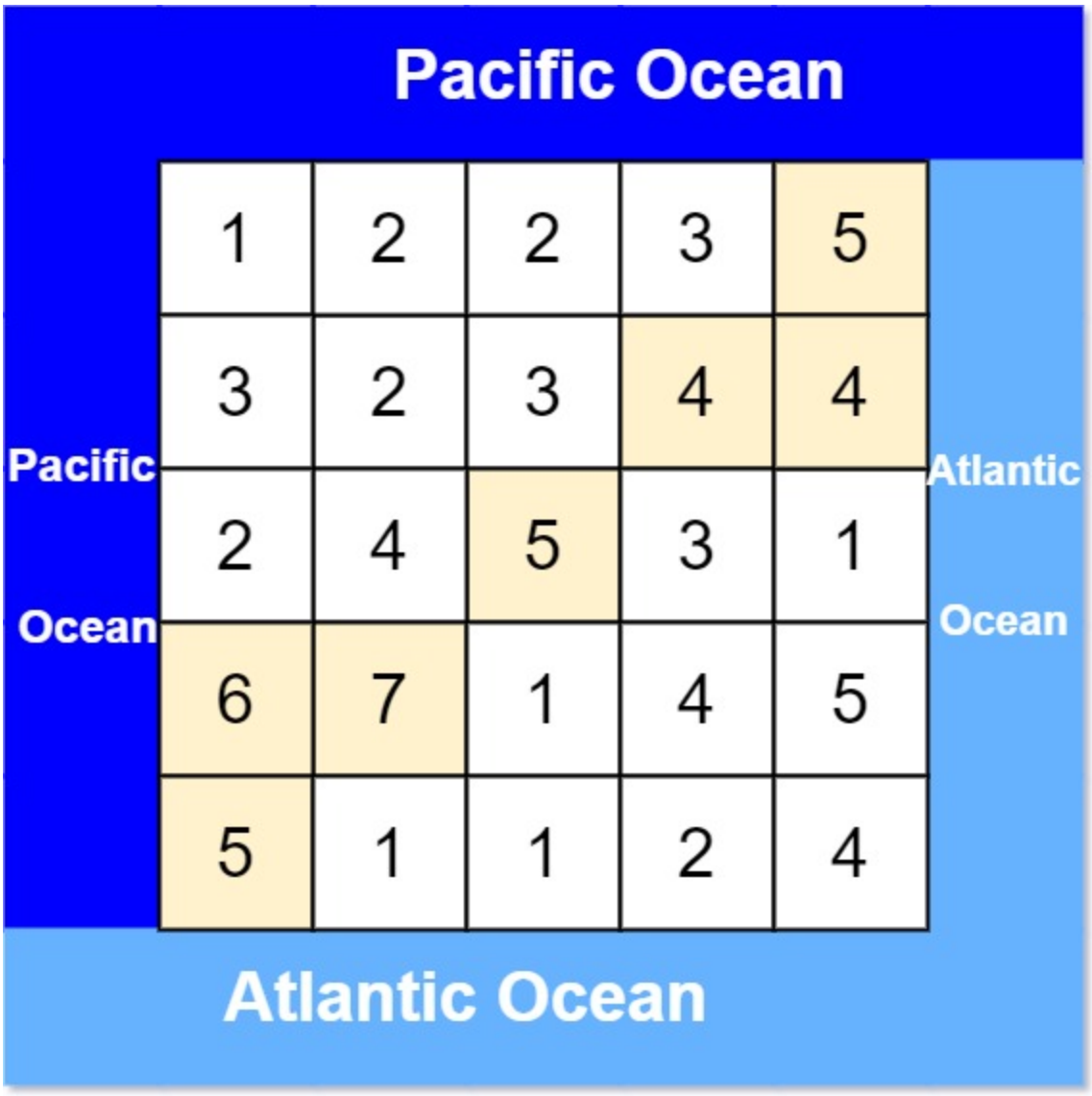
There is an `m x n` rectangular island that borders both the **Pacific Ocean** and **Atlantic Ocean**. The **Pacific Ocean** touches the island's left and top edges, and the **Atlantic Ocean** touches the island's right and bottom edges.

The island is partitioned into a grid of square cells. You are given an `m x n` integer matrix `heights` where `heights[r][c]` represents the **height above sea level** of the cell at coordinate `(r, c)`.

The island receives a lot of rain, and the rain water can flow to neighboring cells directly north, south, east, and west if the neighboring cell's height is **less than or equal to** the current cell's height. Water can flow from any cell adjacent to an ocean into the ocean.

Return *a **2D list** of grid coordinates* `result` *where* `result[i] = [ri, ci]` *denotes that rain water can flow from cell* `(ri, ci)` *to **both** the Pacific and Atlantic oceans*.

**Example 1:**




{ width=350 }

```
Input: heights = [[1,2,2,3,5],[3,2,3,4,4],[2,4,5,3,1],[6,7,1,4,5],[5,1,1,2,4]]
Output: [[0,4],[1,3],[1,4],[2,2],[3,0],[3,1],[4,0]]
Explanation: The following cells can flow to the Pacific and Atlantic oceans,
as shown below:
[0,4]: [0,4] -> Pacific Ocean
       [0,4] -> Atlantic Ocean
[1,3]: [1,3] -> [0,3] -> Pacific Ocean
       [1,3] -> [1,4] -> Atlantic Ocean
[1,4]: [1,4] -> [1,3] -> [0,3] -> Pacific Ocean
       [1,4] -> Atlantic Ocean
[2,2]: [2,2] -> [1,2] -> [0,2] -> Pacific Ocean
       [2,2] -> [2,3] -> [2,4] -> Atlantic Ocean
[3,0]: [3,0] -> Pacific Ocean
```

```
        [3,0] -> [4,0] -> Atlantic Ocean
[3,1]: [3,1] -> [3,0] -> Pacific Ocean
       [3,1] -> [4,1] -> Atlantic Ocean
[4,0]: [4,0] -> Pacific Ocean
       [4,0] -> Atlantic Ocean
Note that there are other possible paths for these cells to flow to the Pacific
and Atlantic oceans.
```

**Example 2:**

```
Input: heights = [[1]]
Output: [[0,0]]
Explanation: The water can flow from the only cell to the Pacific and Atlantic
oceans.
```

**Constraints:**

- `m == heights.length`
- `n == heights[r].length`
- `1 <= m, n <= 200`
- `0 <= heights[r][c] <= 105`

**Hint** dfs each cell, keep track of visited, and track which reach pac, atl; dfs on cells adjacent to pac, atl, find overlap of cells that are visited by both pac and atl cells;

# Number of Islands

**Description**

---

Category Graph Video https://youtu.be/pV2kpPD66nE
Link https://leetcode.com/problems/number-of-islands

---

Given an `m x n` 2D binary grid `grid` which represents a map of `'1'`s (land) and `'0'`s (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Example 1:**

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

**Example 2:**

```
Input: grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
Output: 3
```

**Constraints:**

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 300`
- `grid[i][j]` is `'0'` or `'1'`.

**Hint** foreach cell, if cell is 1 and unvisited run dfs, increment cound and marking each contigous 1 as visited

# Longest Consecutive Sequence

**Description**

---

Category Graph Video https://youtu.be/P6RZZMu_maU
Link https://leetcode.com/problems/longest-consecutive-sequence

---

Given an unsorted array of integers `nums`, return *the length of the longest consecutive elements sequence.*

You must write an algorithm that runs in `O(n)` time.

**Example 1:**

```
Input: nums = [100,4,200,1,3,2]
Output: 4
Explanation: The longest consecutive elements sequence is [1, 2, 3, 4].
Therefore its length is 4.
```

**Example 2:**

```
Input: nums = [0,3,7,2,5,8,4,6,0,1]
Output: 9
```

**Constraints:**

- `0 <= nums.length <= 105`
- `-109 <= nums[i] <= 109`

**Hint** use bruteforce and try to optimize, consider the max subseq containing each num; add each num to hashset, for each num if num-1 doesn't exist, count the consecutive nums after num, ie num+1; there is also a union-find solution;

# Alien Dictionary (Leetcode Premium)

**Description**

---

Category Graph Video https://youtu.be/6kTZYvNNyps
Link https://leetcode.com/problems/alien-dictionary

---

There is a new alien language which uses the latin alphabet. However, the order among letters are unknown to you. You receive a list of **non-empty** words from the dictionary, where **words are sorted lexicographically by the rules of this new language** . Derive the order of letters in this language.

**Example 1:**

**Input:**

```
[
  "wrt",
```

```
  "wrf",
  "er",
  "ett",
  "rftt"
]
```

**Output:** `"wertf"`

**Example 2:**

**Input:**

```
[
  "z",
  "x"
]
```

**Output:** `"zx"`

**Example 3:**

**Input:**

```
[
  "z",
  "x",
  "z"
]
```

**Output:** `""`

**Explanation:** The order is invalid, so return `""`.

**Note:**

1. You may assume all letters are in lowercase.
2. You may assume that if a is a prefix of b, then a must appear before b in the given dictionary.
3. If the order is invalid, return an empty string.
4. There may be multiple valid order of letters, return any one of them is fine.

**Hint** chars of a word not in order, the words are in order, find adjacency list of each unique char by iterating through adjacent words and finding first chars that are different, run topsort on graph and do loop detection;

# Graph Valid Tree (Leetcode Premium)

**Description**

---

Category Graph Video https://youtu.be/bXsUuownnoQ
Link https://leetcode.com/problems/graph-valid-tree

---

Given `n` nodes labeled from `0` to `n-1` and a list of undirected edges (each edge is a pair of nodes), write a function to check whether these edges make up a valid tree.

**Example 1:**

**Input:** `n = 5`, and `edges = [[0,1], [0,2], [0,3], [1,4]]` **Output:** true

**Example 2:**

**Input:** `n = 5,` and `edges = [[0,1], [1,2], [2,3], [1,3], [1,4]]` **Output:** false

**Note** : you can assume that no duplicate edges will appear in `edges`. Since all edges are undirected, `[0,1]` is the same as `[1,0]` and thus will not appear together in `edges`.

**Hint** union find, if union return false, loop exists, at end size must equal n, or its not connected; dfs to get size and check for loop, since each edge is double, before dfs on neighbor of N, remove N from neighbor list of neighbor;

# Number of Connected Components in an Undirected Graph (Leetcode Premium)

**Description**

---

Category Graph Video https://youtu.be/8f1XPm4WOUc
Link https://leetcode.com/problems/number-of-connected-components-in-an-undirected-graph

---

**Hint** dfs on each node that hasn't been visited, increment component count, adjacency list; bfs and union find are possible;

# Insert Interval

**Description**

---

Category Interval Video https://youtu.be/A8NUOmlwOlM
Link https://leetcode.com/problems/insert-interval

---

You are given an array of non-overlapping intervals `intervals` where `intervals[i] = [starti, endi]` represent the start and the end of the `ith` interval and `intervals` is sorted in ascending order by `starti`. You are also given an interval `newInterval = [start, end]` that represents the start and end of another interval.

Insert `newInterval` into `intervals` such that `intervals` is still sorted in ascending order by `starti` and `intervals` still does not have any overlapping intervals (merge overlapping intervals if necessary).

Return `intervals` *after the insertion*.

**Note** that you don't need to modify `intervals` in-place. You can make a new array and return it.

**Example 1:**

```
Input: intervals = [[1,3],[6,9]], newInterval = [2,5]
Output: [[1,5],[6,9]]
```

**Example 2:**

```
Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]
Output: [[1,2],[3,10],[12,16]]
Explanation: Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].
```

**Constraints:**

- `0 <= intervals.length <= 104`
- `intervals[i].length == 2`
- `0 <= starti <= endi <= 105`
- `intervals` is sorted by `starti` in **ascending** order.
- `newInterval.length == 2`
- `0 <= start <= end <= 105`

**Hint** insert new interval in order, then merge intervals; newinterval could only merge with one interval that comes before it, then add remaining intervals;

# Merge Intervals

**Description**

---

Category Interval Video https://youtu.be/44H3cEC2fFM
Link https://leetcode.com/problems/merge-intervals

---

Given an array of `intervals` where `intervals[i] = [starti, endi]`, merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input*.

**Example 1:**

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

**Example 2:**

```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

**Constraints:**

- `1 <= intervals.length <= 104`
- `intervals[i].length == 2`
- `0 <= starti <= endi <= 104`

**Hint** sort each interval, overlapping intervals should be adjacent, iterate and build solution; also graph method, less efficient, more complicated

# Non-overlapping Intervals

**Description**

---

Category Interval Video https://youtu.be/nONCGxWoUfM
Link https://leetcode.com/problems/non-overlapping-intervals

---

Given an array of intervals `intervals` where `intervals[i] = [starti, endi]`, return *the minimum number of intervals you need to remove to make the rest of the intervals non-overlapping*.

**Note** that intervals which only touch at a point are **non-overlapping**. For example, `[1, 2]` and `[2, 3]` are non-overlapping.

**Example 1:**

```
Input: intervals = [[1,2],[2,3],[3,4],[1,3]]
Output: 1
Explanation: [1,3] can be removed and the rest of the intervals are non-
overlapping.
```

**Example 2:**

```
Input: intervals = [[1,2],[1,2],[1,2]]
Output: 2
Explanation: You need to remove two [1,2] to make the rest of the intervals
non-overlapping.
```

**Example 3:**

```
Input: intervals = [[1,2],[2,3]]
Output: 0
Explanation: You don't need to remove any of the intervals since they're
already non-overlapping.
```

**Constraints:**

- `1 <= intervals.length <= 105`
- `intervals[i].length == 2`
- `-5 * 104 <= starti < endi <= 5 * 104`

**Hint** instead of removing, count how max num of intervals you can include, sort intervals, dp to compute max intervals up until the i-th interval;

# Meeting Rooms (Leetcode Premium)

**Description**

---

Category Interval Video https://youtu.be/PaJxqZVPhbg
Link https://leetcode.com/problems/meeting-rooms

---

Given an array of meeting time intervals consisting of start and end times `[(s1,e1),(s2,e2),...] (si < ei)`, determine if a person could attend all meetings.

Only $39.9 for the "Twitter Comment System Project Practice" within a limited time of 7 days!

WeChat Notes Twitter for more information（WeChat ID jiuzhangfeifei）

- $0 \leq intervals.length \leq 10^4$
- $intervals[i].length == 2$
- $0 \leq start_i < end_i \leq 10^6$
- `[(0,8), (8,10)]` is not conflict at **8**

Example

**Example1** Input: intervals = [(0,30),(5,10),(15,20)] Output: false Explanation: (0,30), (5,10) and (0,30),(15,20) will conflict

**Example2** Input: intervals = [(5,8),(9,15)] Output: true Explanation: Two times will notconflict

**Hint** sort intervals by start time, if second interval doesn't overlap with first, then third def wont overlap with first;

# Meeting Rooms II (Leetcode Premium)

**Description**

---

Category Interval Video https://youtu.be/FdzJmTCVyJU
Link https://leetcode.com/problems/meeting-rooms-ii

---

Given an array of meeting time intervals consisting of start and end times `[[s1,e1],[s2,e2],...] (si < ei)`, find the minimum number of conference rooms required.



(0,8),(8,10) is not conflict at 8

Example

**Example1**

```
Input: intervals = [(0,30),(5,10),(15,20)]
Output: 2
Explanation:
We need two meeting rooms
room1: (0,30)
room2: (5,10),(15,20)
```

**Example2**

```
Input: intervals = [(2,7)]
Output:1
Explanation:
Only need one meeting room
```

**Hint** we care about the points in time where we are starting/ending a meeting, we already are given those, just separate start/end and traverse counting num of meetings going at these points in time; for each meeting check if a prev meeting has finished before curr started, using min heap;

# Reverse a Linked List

**Description**

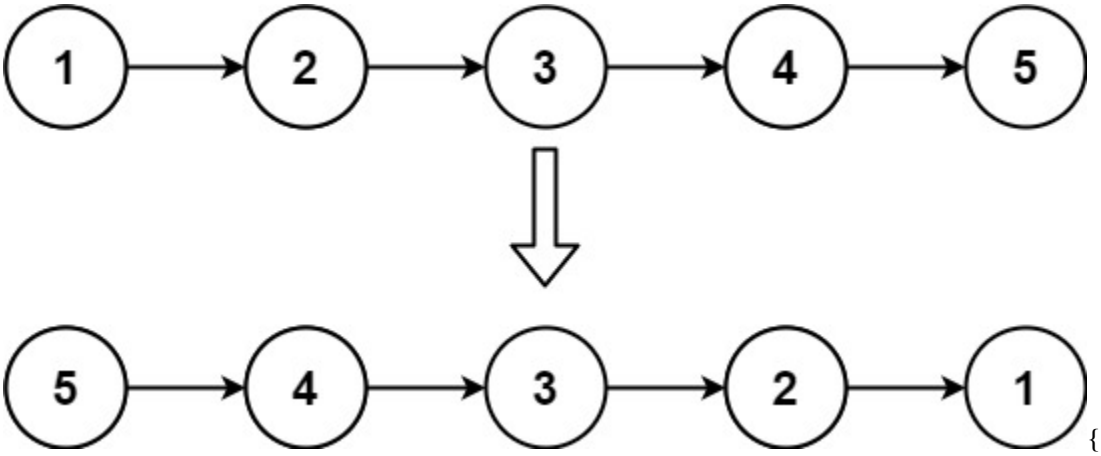
---

Category Linked List Video https://youtu.be/G0_I-ZF0S38
Link https://leetcode.com/problems/reverse-linked-list

---

Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

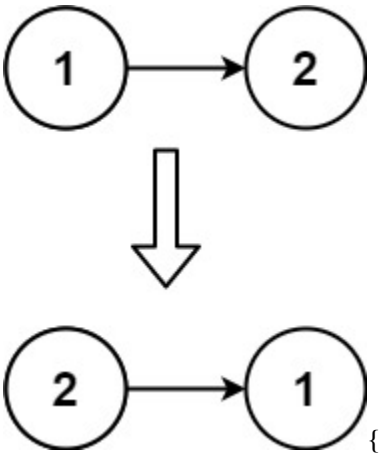
**Example 1:**



{ width=350 }

```
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]
```

**Example 2:**



{ width=175 }

```
Input: head = [1,2]
Output: [2,1]
```

**Example 3:**

```
Input: head = []
Output: []
```

**Constraints:**

- The number of nodes in the list is the range `[0, 5000]`.
- `-5000 <= Node.val <= 5000`

**Follow up:** A linked list can be reversed either iteratively or recursively. Could you implement both?

**Hint** iterate through maintaining cur and prev; recursively reverse, return new head of list

# Detect Cycle in a Linked List

**Description**

---

Category Linked List Video https://youtu.be/gBTe7lFR3vc
Link https://leetcode.com/problems/linked-list-cycle

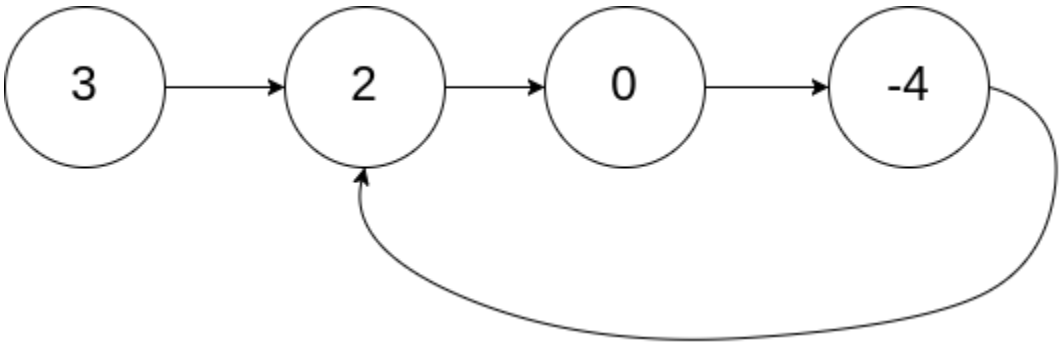
---

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter**.

Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

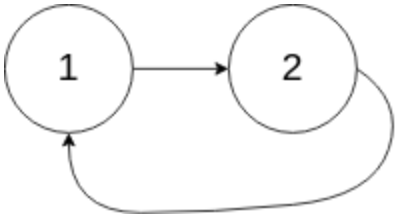
**Example 1:**




{ width=350 }

```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to
the 1st node (0-indexed).
```

**Example 2:**



{ width=150 }

```
Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to
the 0th node.
```

**Example 3:**


```
Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.
```

**Constraints:**

- The number of the nodes in the list is in the range `[0, 104]`.
- `-105 <= Node.val <= 105`
- `pos` is `-1` or a **valid index** in the linked-list.

**Follow up:** Can you solve it using `O(1)` (i.e. constant) memory?

**Hint** dict to remember visited nodes; two pointers at different speeds, if they meet there is loop

# Merge Two Sorted Lists

**Description**

---

Category Linked List Video https://youtu.be/XIdigk956u0
Link https://leetcode.com/problems/merge-two-sorted-lists

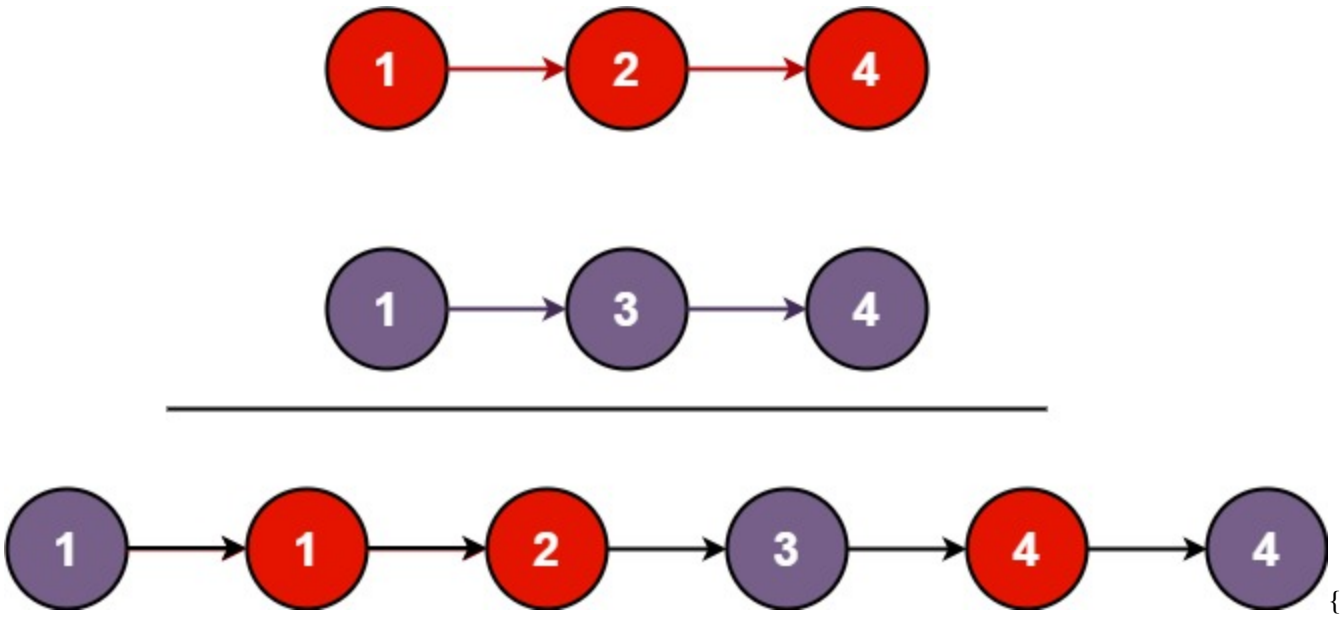
---

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

**Example 1:**




{ width=350 }

```
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
```

**Example 2:**

```
Input: list1 = [], list2 = []
Output: []
```

**Example 3:**

```
Input: list1 = [], list2 = [0]
Output: [0]
```

**Constraints:**

- The number of nodes in both lists is in the range `[0, 50]`.
- `-100 <= Node.val <= 100`
- Both `list1` and `list2` are sorted in **non-decreasing** order.

**Hint** insert each node from one list into the other

# Merge K Sorted Lists

**Description**

---

Category Linked List Video https://youtu.be/q5a5OiGbT6Q
Link https://leetcode.com/problems/merge-k-sorted-lists

---

You are given an array of `k` linked-lists `lists`, each linked-list is sorted in ascending order.

*Merge all the linked-lists into one sorted linked-list and return it.*

**Example 1:**

```
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6
```

**Example 2:**

```
Input: lists = []
Output: []
```

**Example 3:**

```
Input: lists = [[]]
Output: []
```

**Constraints:**

- `k == lists.length`
- `0 <= k <= 104`
- `0 <= lists[i].length <= 500`
- `-104 <= lists[i][j] <= 104`
- `lists[i]` is sorted in **ascending order**.
- The sum of `lists[i].length` will not exceed `104`.

**Hint** divied and conquer, merge lists, N totalnodes, k-lists, O(N*logk). For each list, find min val, insert it into list, use priorityQ to optimize finding min O(N*logk)

# Remove Nth Node From End Of List

**Description**

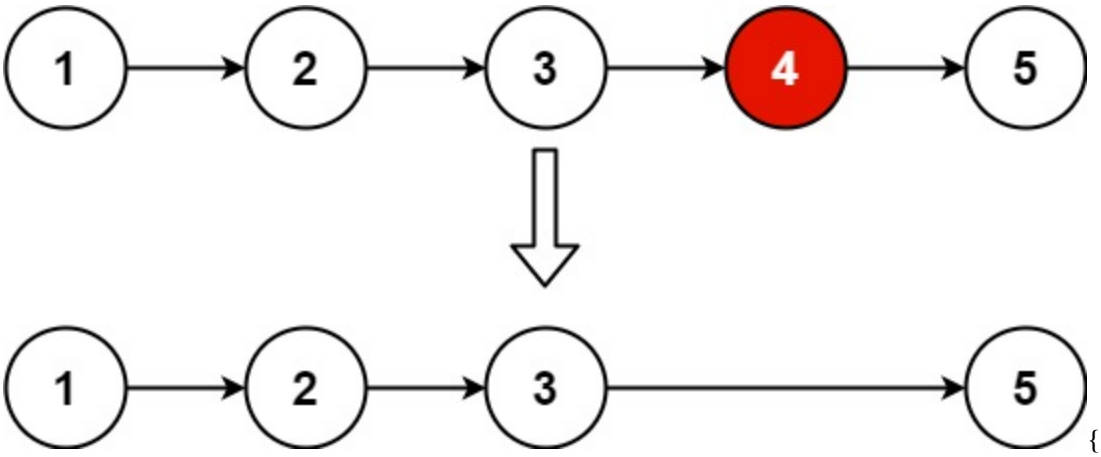
---

Category Linked List Video https://youtu.be/XVuQxVej6y8
Link https://leetcode.com/problems/remove-nth-node-from-end-of-list

---

Given the `head` of a linked list, remove the `nth` node from the end of the list and return its head.

**Example 1:**



{ width=350 }

```
Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]
```

**Example 2:**

```
Input: head = [1], n = 1
Output: []
```

**Example 3:**

```
Input: head = [1,2], n = 1
Output: [1]
```

**Constraints:**

- The number of nodes in the list is `sz`.
- `1 <= sz <= 30`
- `0 <= Node.val <= 100`
- `1 <= n <= sz`

**Follow up:** Could you do this in one pass?

**Hint** use dummy node at head of list, compute len of list; two pointers, second has offset of n from first;

# Reorder List

**Description**

---

Category Linked List Video https://youtu.be/S5bfdUTrKLM
Link https://leetcode.com/problems/reorder-list

---

You are given the head of a singly linked-list. The list can be represented as:

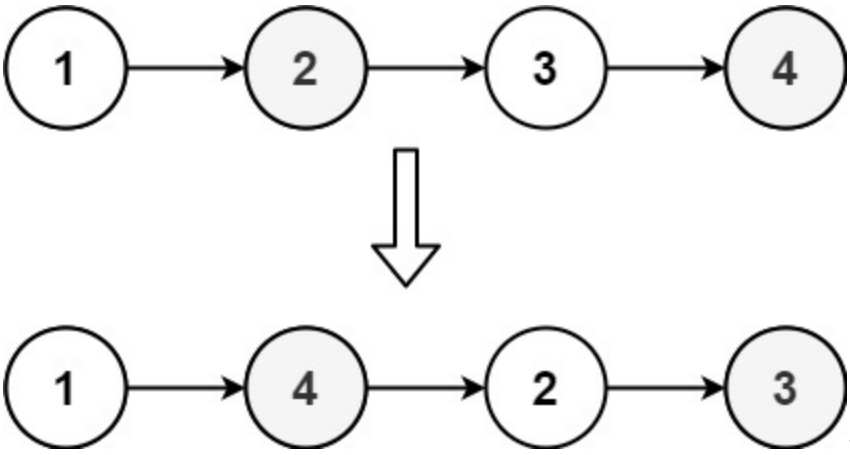
```
L0 → L1 → … → Ln - 1 → Ln
```

*Reorder the list to be on the following form:*

```
L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …
```

You may not modify the values in the list's nodes. Only nodes themselves may be changed.

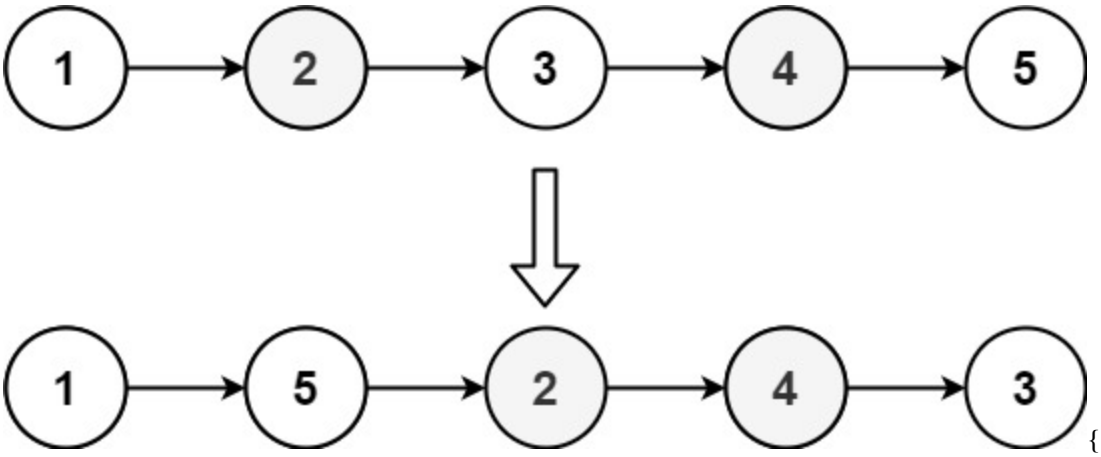
**Example 1:**



{ width=350 }

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

**Example 2:**



{ width=350 }

```
Input: head = [1,2,3,4,5]
Output: [1,5,2,4,3]
```

**Constraints:**

- The number of nodes in the list is in the range `[1, 5 * 104]`.
- `1 <= Node.val <= 1000`

**Hint** reverse second half of list, then easily reorder it; non-optimal way is to store list in array;

# Set Matrix Zeroes

**Description**

---

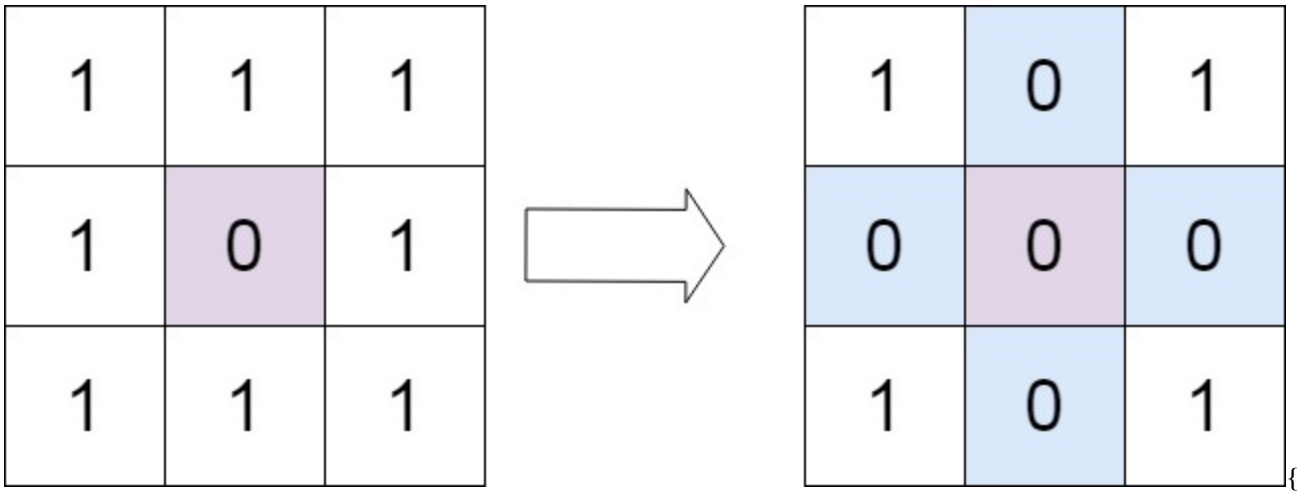
Category Matrix Video https://youtu.be/T41rL0L3Pnw
Link https://leetcode.com/problems/set-matrix-zeroes

---

Given an `m x n` integer matrix `matrix`, if an element is `0`, set its entire row and column to `0`'s.

You must do it in place.

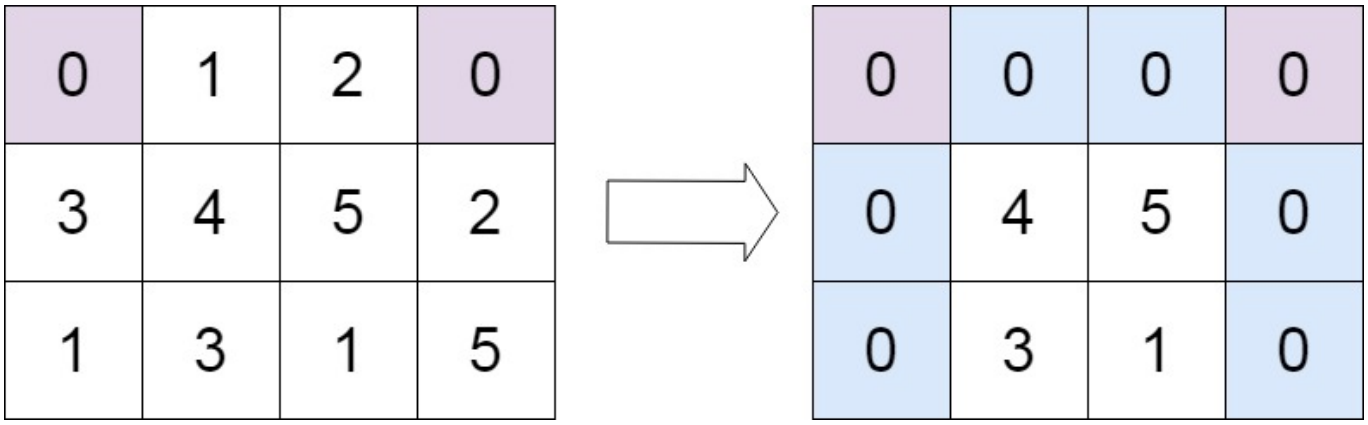
**Example 1:**



{ width=350 }

```
Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]
Output: [[1,0,1],[0,0,0],[1,0,1]]
```

**Example 2:**



{ width=350 }

```
Input: matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]
Output: [[0,0,0,0],[0,4,5,0],[0,3,1,0]]
```

**Constraints:**

- `m == matrix.length`
- `n == matrix[0].length`
- `1 <= m, n <= 200`

- `-231 <= matrix[i][j] <= 231 - 1`

**Follow up:**

- A straightforward solution using `O(mn)` space is probably a bad idea.
- A simple improvement uses `O(m + n)` space, but still not the best solution.
- Could you devise a constant space solution?

**Hint** use sets to keep track of all rows, cols to zero out, after, for each num if it is in a zero row or col then change it to 0; flag first cell in row, and col to mark row/col that needs to be zeroed;

# Spiral Matrix

---

**Description**

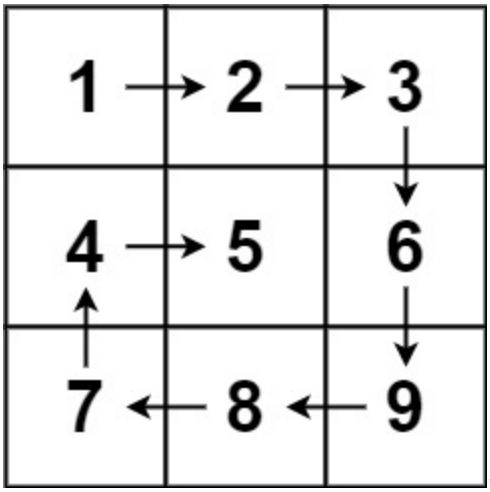
---

Category Matrix Video https://youtu.be/BJnMZNwUk1M
Link https://leetcode.com/problems/spiral-matrix

---

Given an `m x n matrix`, return *all elements of the* `matrix` *in spiral order*.

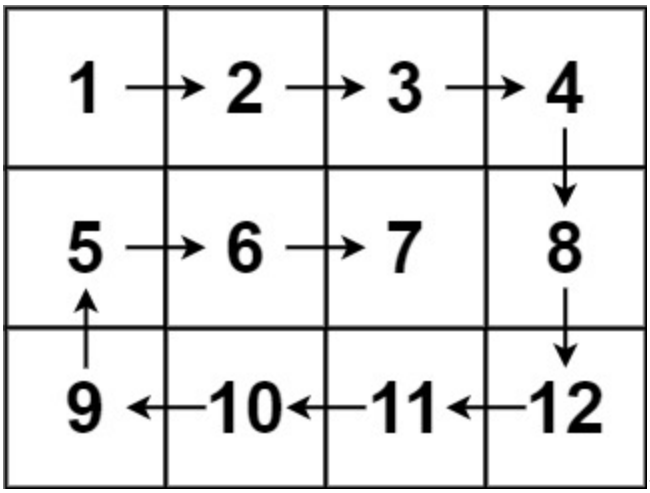
**Example 1:**



{ width=250 }

```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [1,2,3,6,9,8,7,4,5]
```

**Example 2:**



{ width=250 }

```
Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]
```

**Constraints:**

- `m == matrix.length`
- `n == matrix[i].length`
- `1 <= m, n <= 10`
- `-100 <= matrix[i][j] <= 100`

**Hint** keep track of visited cells; keep track of boundaries, layer-by-layer;

# Rotate Image

**Description**

---

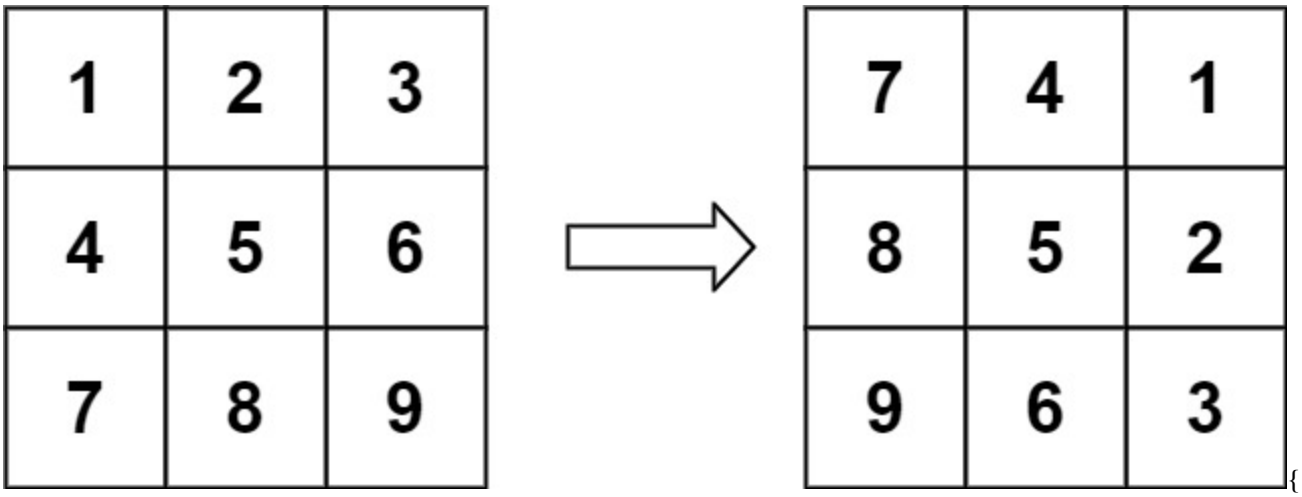
Category Matrix Video https://youtu.be/fMSJSS7eO1w
Link https://leetcode.com/problems/rotate-image

---

You are given an `n x n` 2D `matrix` representing an image, rotate the image by **90** degrees (clockwise).

You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

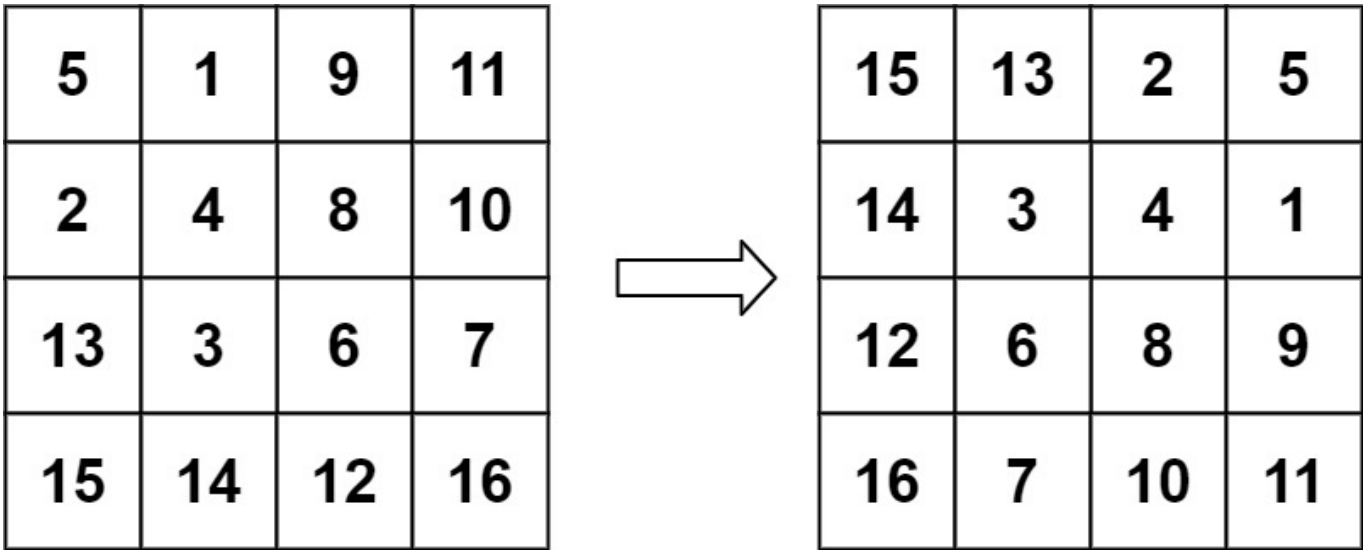
**Example 1:**



{ width=350 }

```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]
```

**Example 2:**



{ width=350 }

```
Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]
```

**Constraints:**

- `n == matrix.length == matrix[i].length`
- `1 <= n <= 20`
- `-1000 <= matrix[i][j] <= 1000`

**Hint** rotate layer-by-layer, use that it's a square as advantage, rotate positions in reverse order, store a in temp, a = b, b = c, c = d, d = temp;

# Word Search

---

**Description**

---

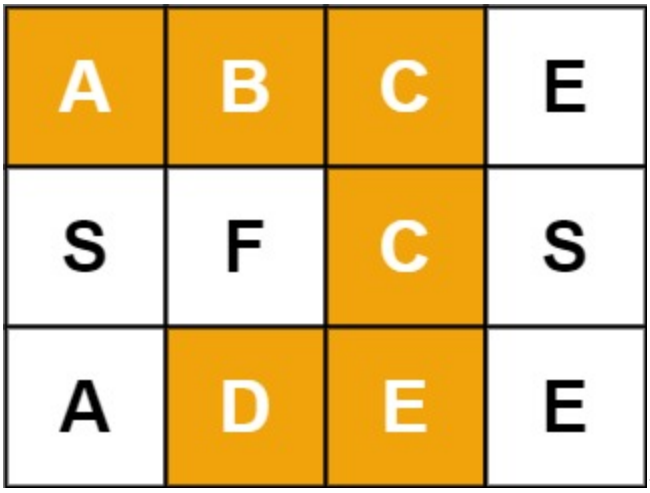
Category Matrix Video https://youtu.be/pfiQ_PS1g8E
Link https://leetcode.com/problems/word-search

---

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.

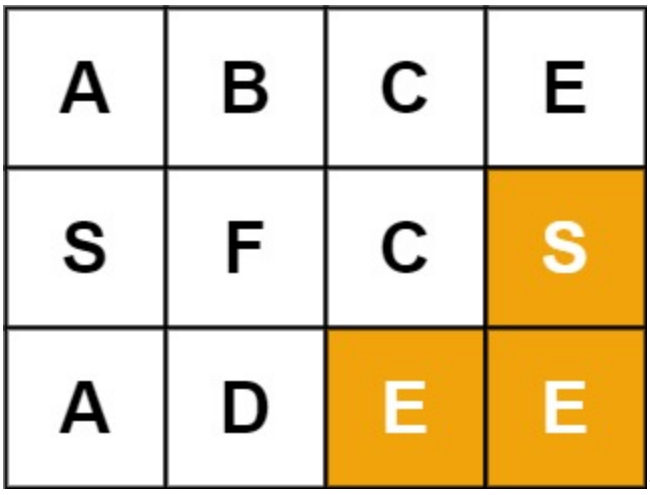
**Example 1:**



{ width=275 }

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

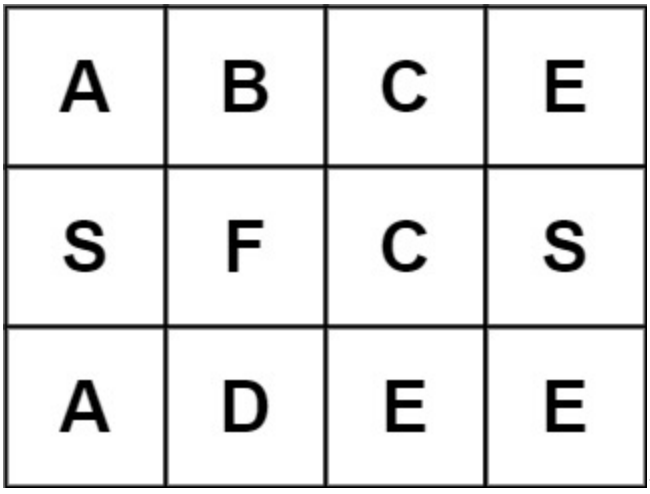
**Example 2:**



{ width=275 }

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

**Example 3:**



{ width=275 }

```
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`
- `1 <= word.length <= 15`
- `board` and `word` consists of only lowercase and uppercase English letters.

**Follow up:** Could you use search pruning to make your solution faster with a larger `board`?

**Hint** dfs on each cell, for each search remember visited cells, and remove cur visited cell right before you return from dfs;

# Longest Substring Without Repeating Characters

**Description**

---

Category String Video https://youtu.be/wiGpQwVHdE0
Link https://leetcode.com/problems/longest-substring-without-repeating-characters

---

Given a string `s`, find the length of the **longest substring** without repeating characters.

**Example 1:**

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

**Example 2:**

```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

**Example 3:**

```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a
substring.
```

**Constraints:**

- `0 <= s.length <= 5 * 104`
- `s` consists of English letters, digits, symbols and spaces.

**Hint** sliding window, if we see same char twice within curr window, shift start position;

# Longest Repeating Character Replacement

**Description**

---

Category String Video https://youtu.be/gqXU1UyA8pk
Link https://leetcode.com/problems/longest-repeating-character-replacement

---

You are given a string `s` and an integer `k`. You can choose any character of the string and change it to any other uppercase English character. You can perform this operation at most `k` times.

Return *the length of the longest substring containing the same letter you can get after performing the above operations*.

**Example 1:**

```
Input: s = "ABAB", k = 2
Output: 4
Explanation: Replace the two 'A's with two 'B's or vice versa.
```

**Example 2:**

```
Input: s = "AABABBA", k = 1
Output: 4
Explanation: Replace the one 'A' in the middle with 'B' and form "AABBBBA".
The substring "BBBB" has the longest repeating letters, which is 4.
There may exists other ways to achieve this answer too.
```

**Constraints:**

- `1 <= s.length <= 105`
- `s` consists of only uppercase English letters.

- `0 <= k <= s.length`

**Hint** PAY ATTENTION: limited to chars A-Z; for each capital char, check if it could create the longest repeating substr, use sliding window to optimize; check if windowlen=1 works, if yes, increment len, if not, shift window right;

# Minimum Window Substring

**Description**

---

Category String Video https://youtu.be/jSto0O4AJbM
Link https://leetcode.com/problems/minimum-window-substring

---

Given two strings `s` and `t` of lengths `m` and `n` respectively, return *the **minimum window substring** of `s` such that every character in `t` (**including duplicates**) is included in the window*. If there is no such substring, return *the empty string* `""`.

The testcases will be generated such that the answer is **unique**.

**Example 1:**

```
Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C'
from string t.
```

**Example 2:**

```
Input: s = "a", t = "a"
Output: "a"
Explanation: The entire string s is the minimum window.
```

**Example 3:**

```
Input: s = "a", t = "aa"
Output: ""
Explanation: Both 'a's from t must be included in the window.
Since the largest window of s only has one 'a', return empty string.
```

**Constraints:**

- `m == s.length`
- `n == t.length`
- `1 <= m, n <= 105`
- `s` and `t` consist of uppercase and lowercase English letters.

**Follow up:** Could you find an algorithm that runs in `O(m + n)` time?

**Hint** need is num of unique char in T, HAVE is num of char we have valid count for, sliding window, move right until valid, if valid, increment left until invalid, to check validity keep track if the count of each unique char is satisfied;

# Valid Anagram

**Description**

---

Category String Video https://youtu.be/9UtInBqnCgA
Link https://leetcode.com/problems/valid-anagram

---

Given two strings `s` and `t`, return `true` if `t` is an anagram of `s`, and `false` otherwise.

**Example 1:**

**Input:** s = "anagram", t = "nagaram"

**Output:** true

**Example 2:**

**Input:** s = "rat", t = "car"

**Output:** false

**Constraints:**

- `1 <= s.length, t.length <= 5 * 104`
- `s` and `t` consist of lowercase English letters.

**Follow up:** What if the inputs contain Unicode characters? How would you adapt your solution to such a case?

**Hint** hashmap to count each char in str1, decrement for str2;

# Group Anagrams

**Description**

---

Category String Video https://youtu.be/vzdNOK2oB2E
Link https://leetcode.com/problems/group-anagrams

---

Given an array of strings `strs`, group the anagrams together. You can return the answer in **any order**.

**Example 1:**

**Input:** strs = ["eat","tea","tan","ate","nat","bat"]

**Output:** [["bat"],["nat","tan"],["ate","eat","tea"]]

**Explanation:**

- There is no string in strs that can be rearranged to form `"bat"`.
- The strings `"nat"` and `"tan"` are anagrams as they can be rearranged to form each other.
- The strings `"ate"`, `"eat"`, and `"tea"` are anagrams as they can be rearranged to form each other.

**Example 2:**

**Input:** strs = [""]

**Output:** [[""]]

**Example 3:**

**Input:** strs = ["a"]

**Output:** [["a"]]

**Constraints:**

- `1 <= strs.length <= 104`
- `0 <= strs[i].length <= 100`
- `strs[i]` consists of lowercase English letters.

**Hint** for each of 26 chars, use count of each char in each word as tuple for key in dict, value is the list of anagrams;

# Valid Parentheses

---

**Description**

---

Category String Video https://youtu.be/WTzjTskDFMg
Link https://leetcode.com/problems/valid-parentheses

---

Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

**Example 1:**

**Input:** s = "()"

**Output:** true

**Example 2:**

**Input:** s = "()[]{}"

**Output:** true

**Example 3:**

**Input:** s = "(]"

**Output:** false

**Example 4:**

**Input:** s = "([])"

**Output:** true

**Constraints:**

- `1 <= s.length <= 104`
- `s` consists of parentheses only `'()[]{}'`.

**Hint** push opening brace on stack, pop if matching close brace, at end if stack empty, return true;

# Valid Palindrome

---

**Description**

---

Category String Video https://youtu.be/jJXJ16kPFWg
Link https://leetcode.com/problems/valid-palindrome

---

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string `s`, return `true` *if it is a* ***palindrome****, or* `false` *otherwise*.

**Example 1:**

```
Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.
```

**Example 2:**

```
Input: s = "race a car"
Output: false
Explanation: "raceacar" is not a palindrome.
```

**Example 3:**

```
Input: s = " "
Output: true
Explanation: s is an empty string "" after removing non-alphanumeric
characters.
Since an empty string reads the same forward and backward, it is a palindrome.
```

**Constraints:**

- `1 <= s.length <= 2 * 105`
- `s` consists only of printable ASCII characters.

**Hint** left, right pointers, update left and right until each points at alphanum, compare left and right, continue until left >= right, don't distinguish between upper/lowercase;

# Longest Palindromic Substring

**Description**

---

Category String Video https://youtu.be/XYQecbcd6_c
Link https://leetcode.com/problems/longest-palindromic-substring

---

Given a string `s`, return *the longest palindromic substring* in `s`.

**Example 1:**

```
Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.
```

**Example 2:**

```
Input: s = "cbbd"
Output: "bb"
```

**Constraints:**

- `1 <= s.length <= 1000`
- `s` consist of only digits and English letters.

**Hint** foreach char in str, consider it were the middle, consider if pali was odd or even;

# Palindromic Substrings

**Description**

---

Category String Video https://youtu.be/4RACzI5-du8
Link https://leetcode.com/problems/palindromic-substrings

---

Given a string `s`, return *the number of **palindromic substrings** in it*.

A string is a **palindrome** when it reads the same backward as forward.

A **substring** is a contiguous sequence of characters within the string.

**Example 1:**

```
Input: s = "abc"
Output: 3
Explanation: Three palindromic strings: "a", "b", "c".
```

**Example 2:**

```
Input: s = "aaa"
Output: 6
Explanation: Six palindromic strings: "a", "a", "a", "aa", "aa", "aaa".
```

**Constraints:**

- `1 <= s.length <= 1000`
- `s` consists of lowercase English letters.

**Hint** same as longest palindromic string, each char in str as middle and expand outwards, do same for pali of even len; maybe read up on manachers alg

# Encode and Decode Strings (Leetcode Premium)

**Description**

---

Category String Video https://youtu.be/B1k_sxOSgv8
Link https://leetcode.com/problems/encode-and-decode-strings

---

Design an algorithm to encode **a list of strings** to **a string** . The encoded string is then sent over the network and is decoded back to the original list of strings.

Machine 1 (sender) has the function:

```
string encode(vector<string> strs) {
  // ... your code
  return encoded_string;
}
```

Machine 2 (receiver) has the function:

```
vector<string> decode(string s) {
  //... your code
  return strs;
}
```

So Machine 1 does:

```
string encoded_string = encode(strs);
```

and Machine 2 does:

```
vector<string> strs2 = decode(encoded_string);
```

`strs2` in Machine 2 should be the same as `strs` in Machine 1.

Implement the `encode` and `decode` methods.

**Note:**

- The string may contain any possible characters out of 256 valid ascii characters. Your algorithm should be generalized enough to work on any possible characters.
- Do not use class member/global/static variables to store states. Your encode and decode algorithms should be stateless.
- Do not rely on any library method such as `eval` or serialize methods. You should implement your own encode/decode algorithm.

**Hint** store length of str before each string and delimiter like '#';

# Maximum Depth of Binary Tree

**Description**

---

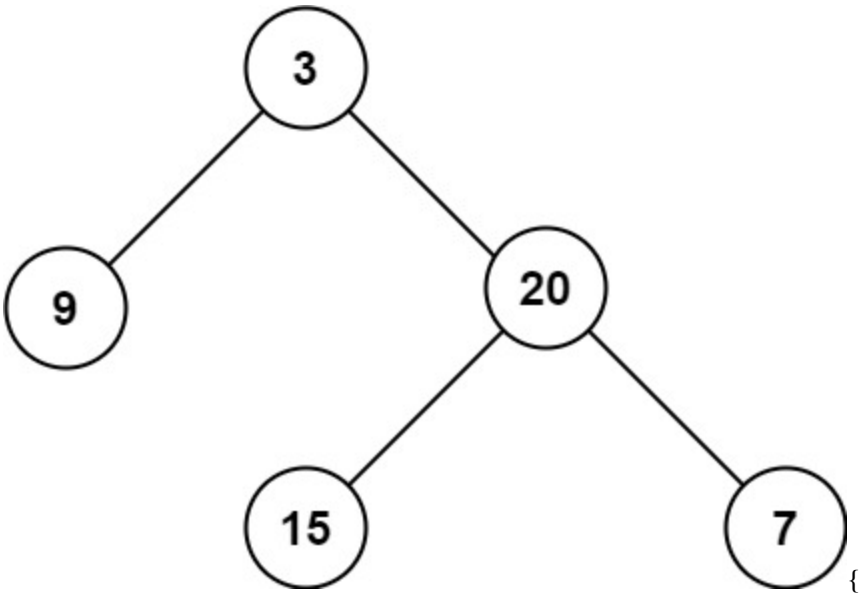
Category Tree Video https://youtu.be/hTM3phVI6YQ
Link https://leetcode.com/problems/maximum-depth-of-binary-tree

---

Given the `root` of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

**Example 1:**



{ width=350 }

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

**Example 2:**

```
Input: root = [1,null,2]
Output: 2
```

**Constraints:**

- The number of nodes in the tree is in the range `[0, 104]`.
- `-100 <= Node.val <= 100`

**Hint** recursive dfs to find max-depth of subtrees; iterative bfs to count number of levels in tree

# Same Tree

**Description**

---

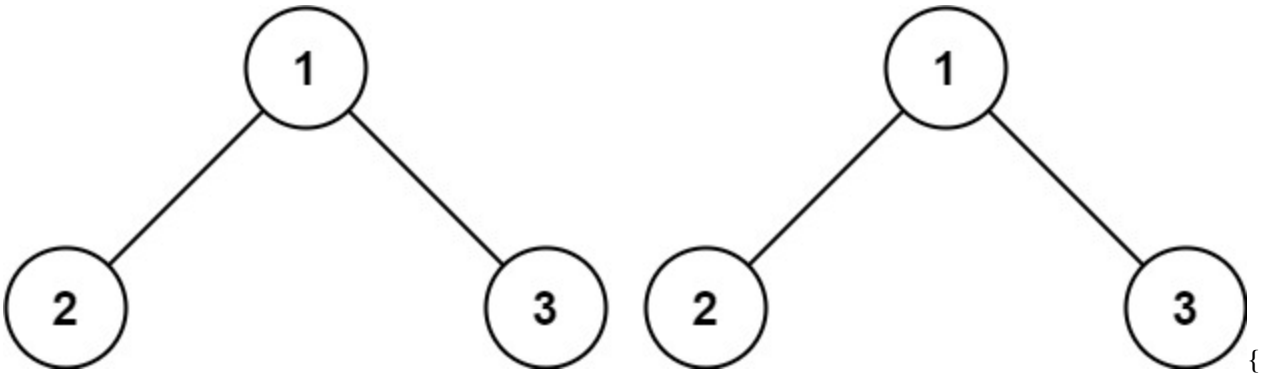
Category Tree Video https://youtu.be/vRbbcKXCxOw
Link https://leetcode.com/problems/same-tree

---

Given the roots of two binary trees `p` and `q`, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

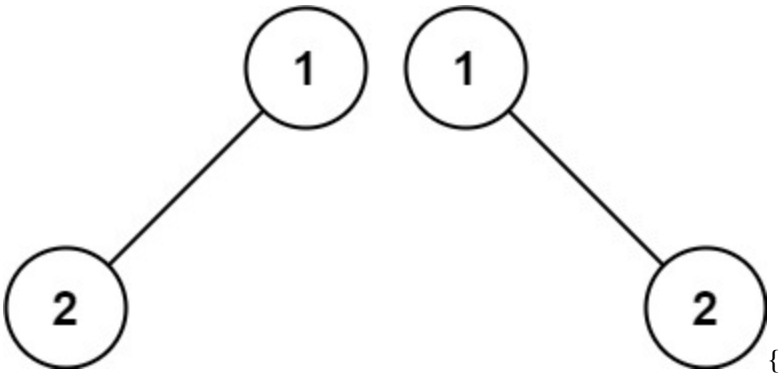
**Example 1:**



{
width=350 }

```
Input: p = [1,2,3], q = [1,2,3]
Output: true
```

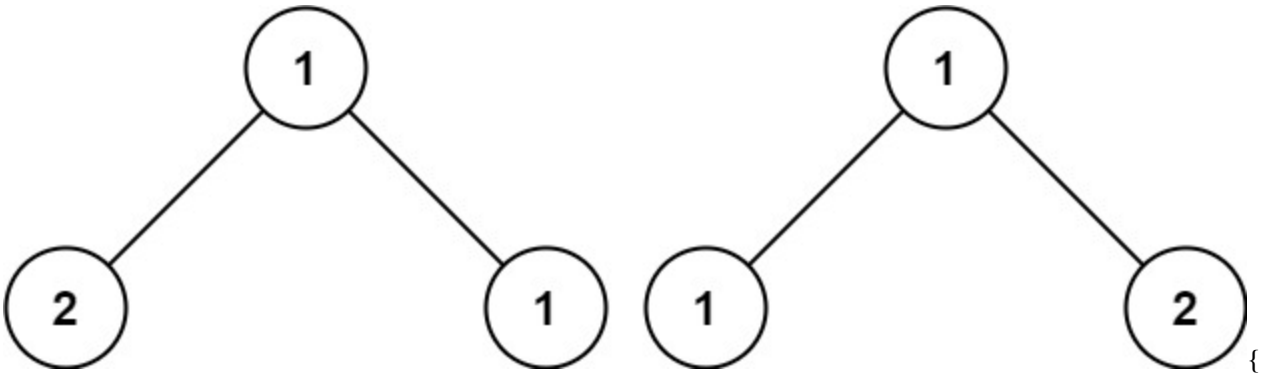
**Example 2:**



{ width=350 }

```
Input: p = [1,2], q = [1,null,2]
Output: false
```

**Example 3:**



{ width=350 }

```
Input: p = [1,2,1], q = [1,1,2]
Output: false
```

**Constraints:**

- The number of nodes in both trees is in the range `[0, 100]`.
- `-104 <= Node.val <= 104`

**Hint** recursive dfs on both trees at the same time; iterative bfs compare each level of both trees

# Invert/Flip Binary Tree

**Description**

---

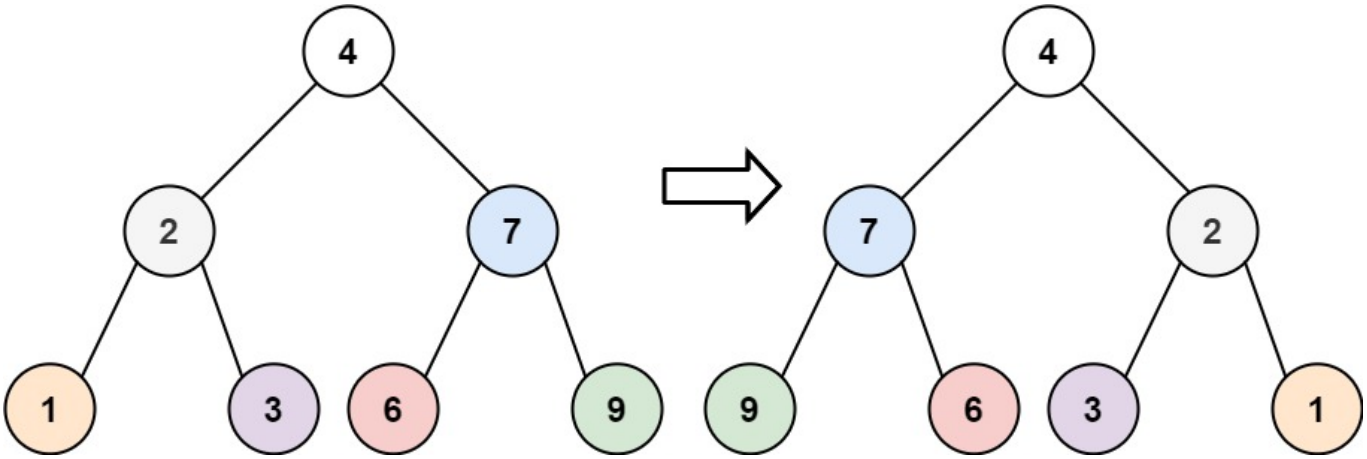
Category Tree Video https://youtu.be/OnSn2XEQ4MY

Link https://leetcode.com/problems/invert-binary-tree

---

Given the `root` of a binary tree, invert the tree, and return *its root*.

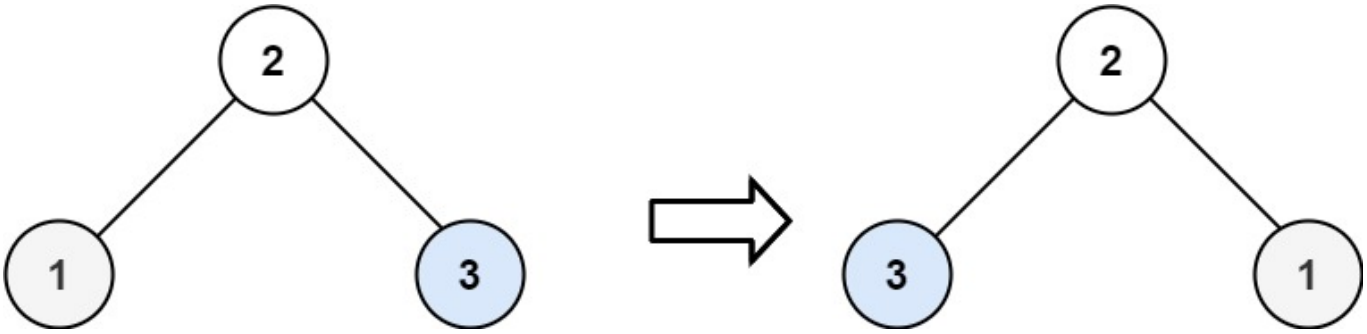
**Example 1:**




{ width=350 }

```
Input: root = [4,2,7,1,3,6,9]
Output: [4,7,2,9,6,3,1]
```

**Example 2:**




{ width=350 }

```
Input: root = [2,1,3]
Output: [2,3,1]
```

**Example 3:**

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

**Constraints:**

- The number of nodes in the tree is in the range `[0, 100]`.
- `-100 <= Node.val <= 100`

**Hint** recursive dfs to invert subtrees; bfs to invert levels, use collections.deque; iterative dfs is easy with stack if doing pre-order traversal

# Binary Tree Maximum Path Sum

**Description**

---

Category Tree Video https://youtu.be/Hr5cWUld4vU
Link https://leetcode.com/problems/binary-tree-maximum-path-sum

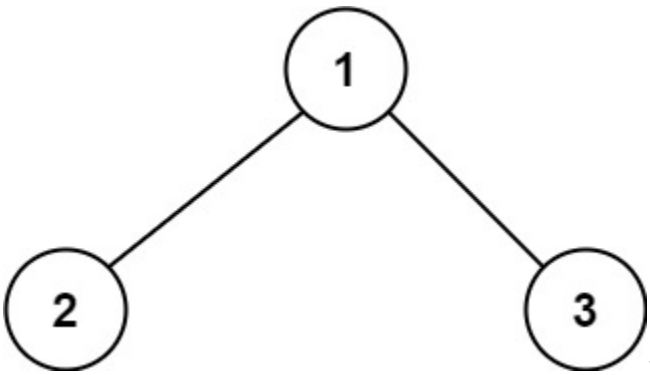
---

A **path** in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence **at most once**. Note that the path does not need to pass through the root.

The **path sum** of a path is the sum of the node's values in the path.

Given the `root` of a binary tree, return *the maximum **path sum** of any **non-empty** path*.

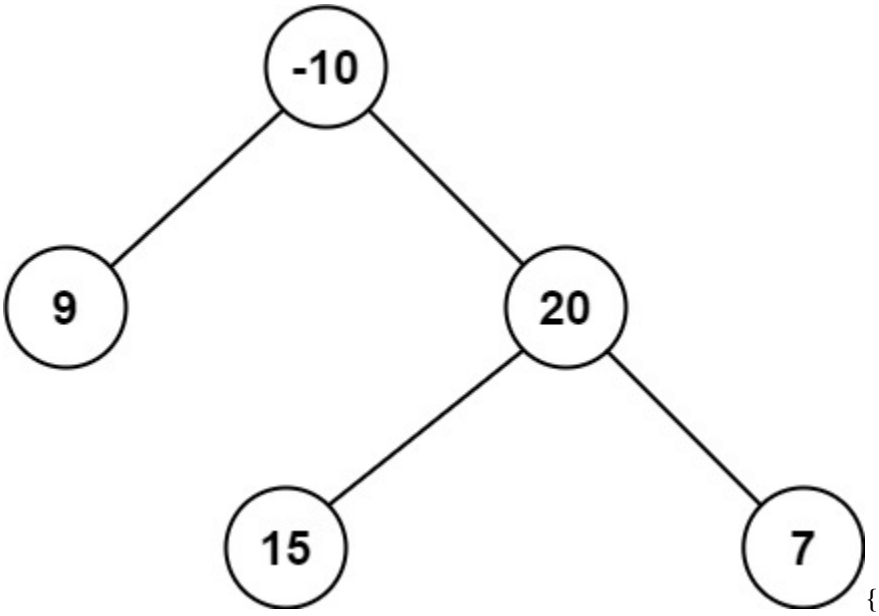
**Example 1:**



{ width=350 }

```
Input: root = [1,2,3]
Output: 6
Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.
```

**Example 2:**



{ width=350 }

```
Input: root = [-10,9,20,null,null,15,7]
Output: 42
Explanation: The optimal path is 15 -> 20 -> 7 with a path sum of 15 + 20 + 7 =
42.
```

**Constraints:**

- The number of nodes in the tree is in the range `[1, 3 * 104]`.
- `-1000 <= Node.val <= 1000`

**Hint** helper returns maxpathsum without splitting branches, inside helper we also update maxSum by computing maxpathsum WITH a split;

# Binary Tree Level Order Traversal

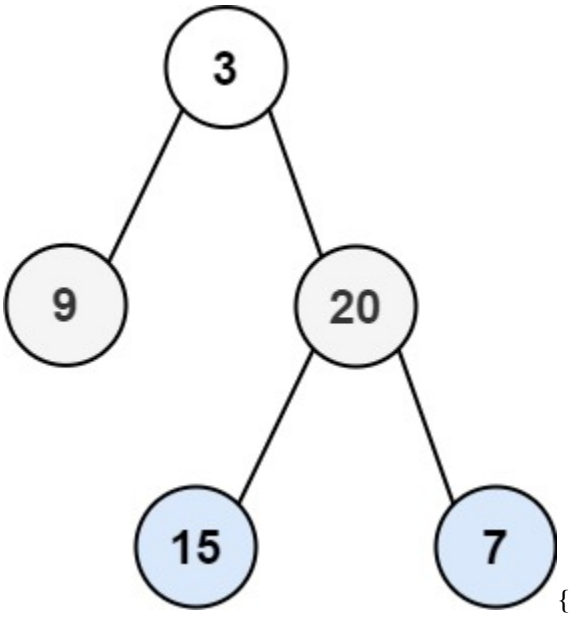
**Description**

Category Tree Video https://youtu.be/6ZnyEApgFYg
Link https://leetcode.com/problems/binary-tree-level-order-traversal

---

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

**Example 1:**



{ width=350 }

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

**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`

**Hint** iterative bfs, add prev level which doesn't have any nulls to the result;

# Serialize and Deserialize Binary Tree

**Description**

---

Category Tree Video https://youtu.be/u4JAi2JJhI8
Link https://leetcode.com/problems/serialize-and-deserialize-binary-tree

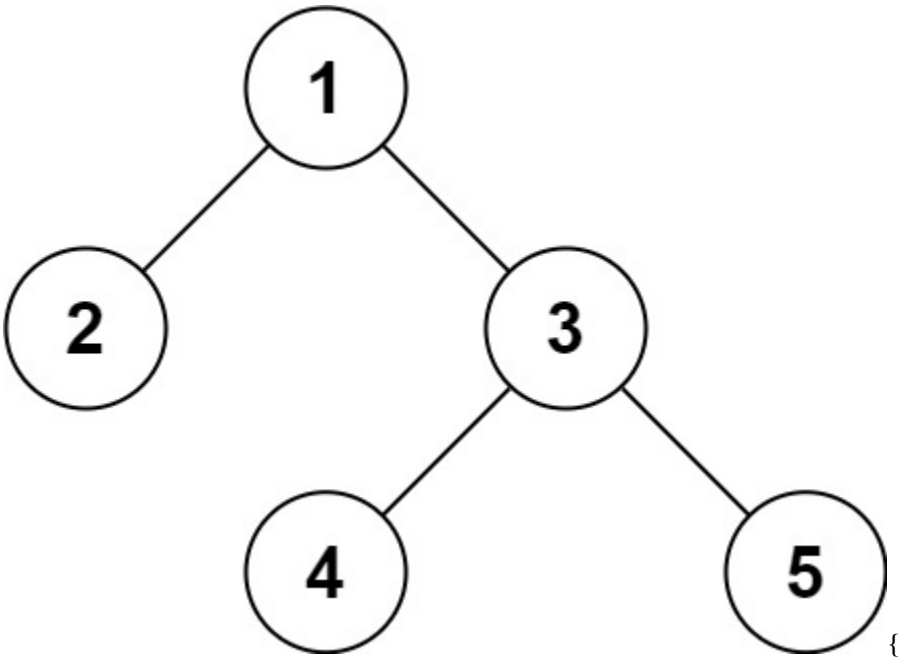
---

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

**Clarification:** The input/output format is the same as how LeetCode serializes a binary tree. You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

**Example 1:**



{ width=350 }

```
Input: root = [1,2,3,null,null,4,5]
Output: [1,2,3,null,null,4,5]
```

**Example 2:**

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

**Constraints:**

- The number of nodes in the tree is in the range `[0, 104]`.
- `-1000 <= Node.val <= 1000`

**Hint** bfs every single non-null node is added to string, and it's children are added too, even if they're null, deserialize by adding each non-null node to queue, deque node, it's children are next two nodes in string;

# Subtree of Another Tree

**Description**

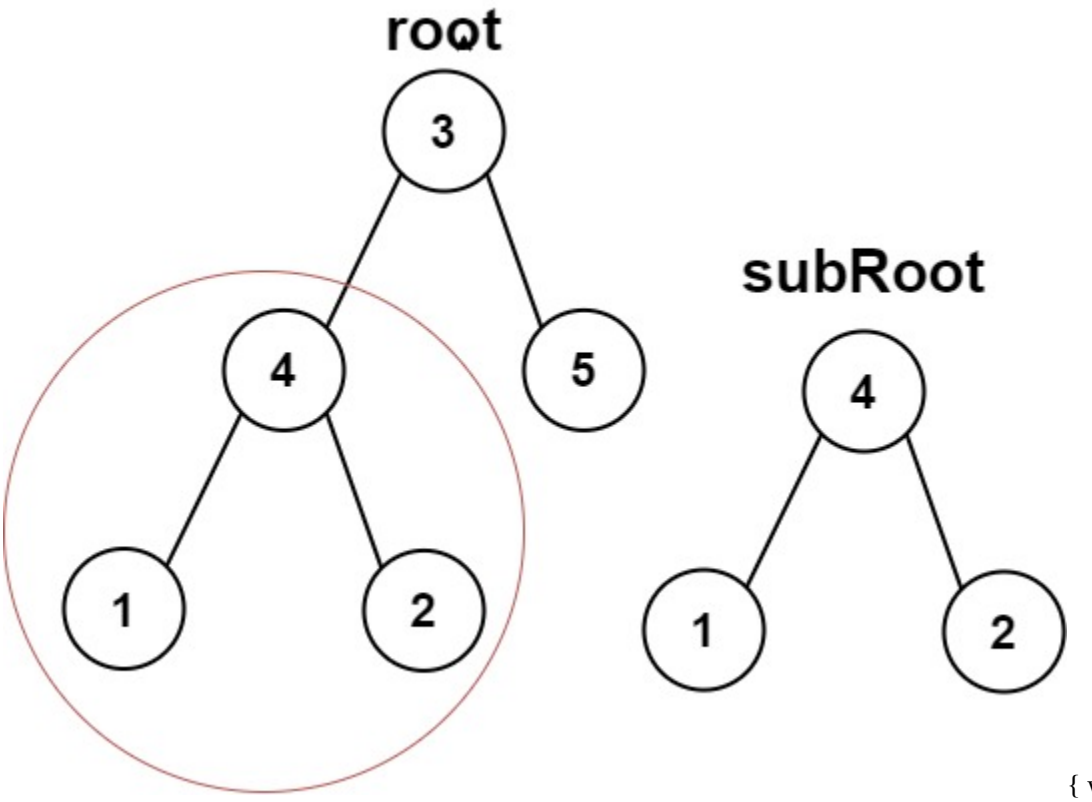
Category Tree Video https://youtu.be/E36O5SWp-LE
Link https://leetcode.com/problems/subtree-of-another-tree

---

Given the roots of two binary trees `root` and `subRoot`, return `true` if there is a subtree of `root` with the same structure and node values of `subRoot` and `false` otherwise.

A subtree of a binary tree `tree` is a tree that consists of a node in `tree` and all of this node's descendants. The tree `tree` could also be considered as a subtree of itself.

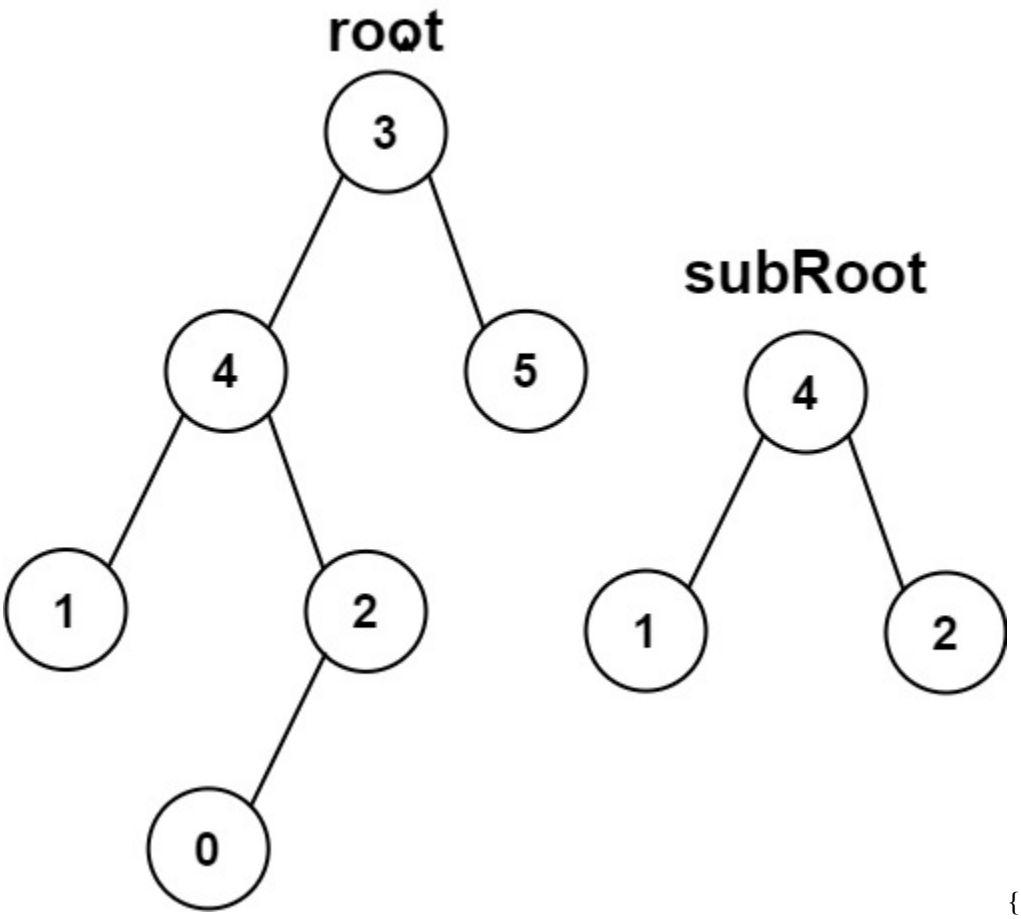
**Example 1:**




{ width=350 }

```
Input: root = [3,4,5,1,2], subRoot = [4,1,2]
Output: true
```

**Example 2:**




{ width=350 }

```
Input: root = [3,4,5,1,2,null,null,null,null,0], subRoot = [4,1,2]
Output: false
```

**Constraints:**

- The number of nodes in the `root` tree is in the range `[1, 2000]`.
- The number of nodes in the `subRoot` tree is in the range `[1, 1000]`.
- `-104 <= root.val <= 104`
- `-104 <= subRoot.val <= 104`

**Hint** traverse s to check if any subtree in s equals t; merkle hashing?

# Construct Binary Tree from Preorder and Inorder Traversal

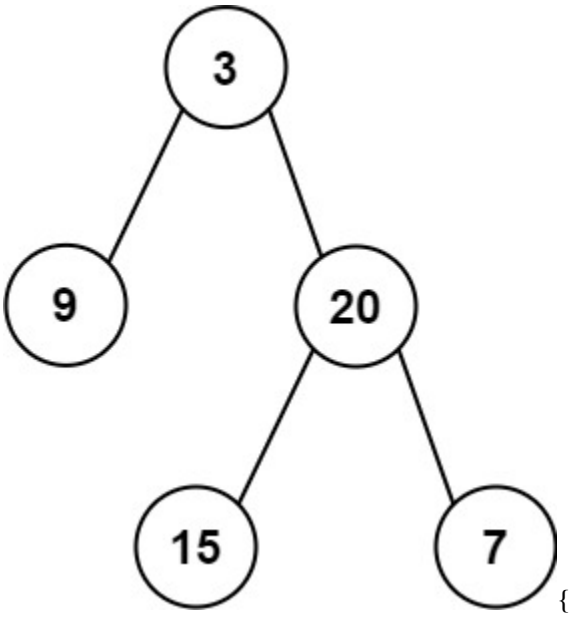
**Description**

Category Tree Video https://youtu.be/ihj4IQGZ2zc
Link https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal

---

Given two integer arrays `preorder` and `inorder` where `preorder` is the preorder traversal of a binary tree and `inorder` is the inorder traversal of the same tree, construct and return *the binary tree*.

**Example 1:**



{ width=350 }

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

**Example 2:**

```
Input: preorder = [-1], inorder = [-1]
Output: [-1]
```

**Constraints:**

- `1 <= preorder.length <= 3000`
- `inorder.length == preorder.length`
- `-3000 <= preorder[i], inorder[i] <= 3000`
- `preorder` and `inorder` consist of **unique** values.
- Each value of `inorder` also appears in `preorder`.

- `preorder` is **guaranteed** to be the preorder traversal of the tree.
- `inorder` is **guaranteed** to be the inorder traversal of the tree.

**Hint** first element in pre-order is root, elements left of root in in-order are left subtree, right of root are right subtree, recursively build subtrees;

# Validate Binary Search Tree

---

**Description**

---

Category Tree Video https://youtu.be/s6ATEkipzow
Link https://leetcode.com/problems/validate-binary-search-tree

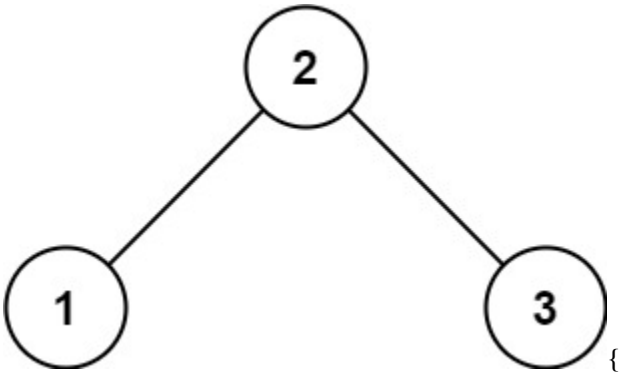
---

Given the `root` of a binary tree, *determine if it is a valid binary search tree (BST)*.

A **valid BST** is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

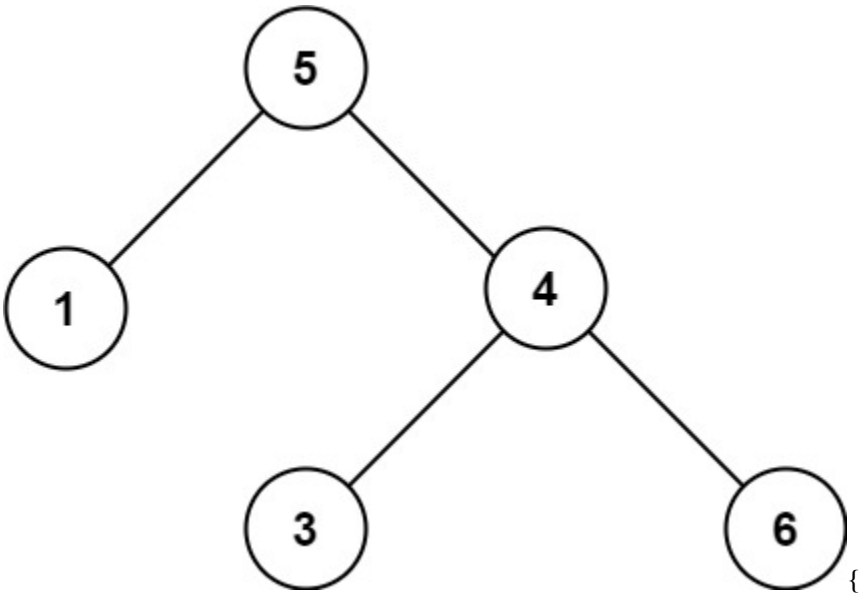
**Example 1:**



{ width=350 }

```
Input: root = [2,1,3]
Output: true
```

**Example 2:**



{ width=350 }

```
Input: root = [5,1,4,null,null,3,6]
Output: false
Explanation: The root node's value is 5 but its right child's value is 4.
```

**Constraints:**

- The number of nodes in the tree is in the range `[1, 104]`.
- `-231 <= Node.val <= 231 - 1`

**Hint** trick is use built in python min/max values float("inf"), "-inf", as parameters; iterative in-order traversal, check each val is greater than prev;

# Kth Smallest Element in a BST

**Description**

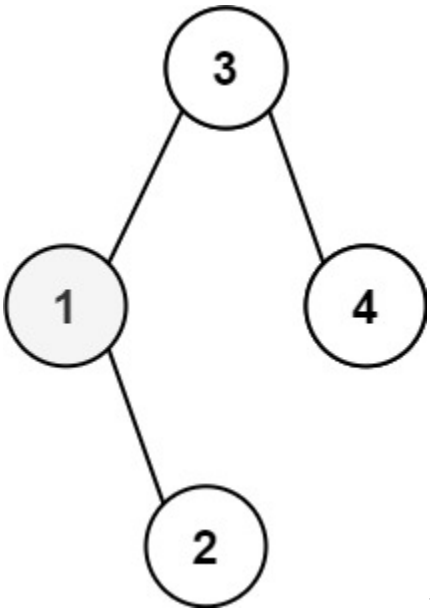
---

Category Tree Video https://youtu.be/5LUXSvjmGCw
Link https://leetcode.com/problems/kth-smallest-element-in-a-bst

---

Given the `root` of a binary search tree, and an integer `k`, return *the* `kth` *smallest value (**1-indexed**) of all the values of the nodes in the tree*.

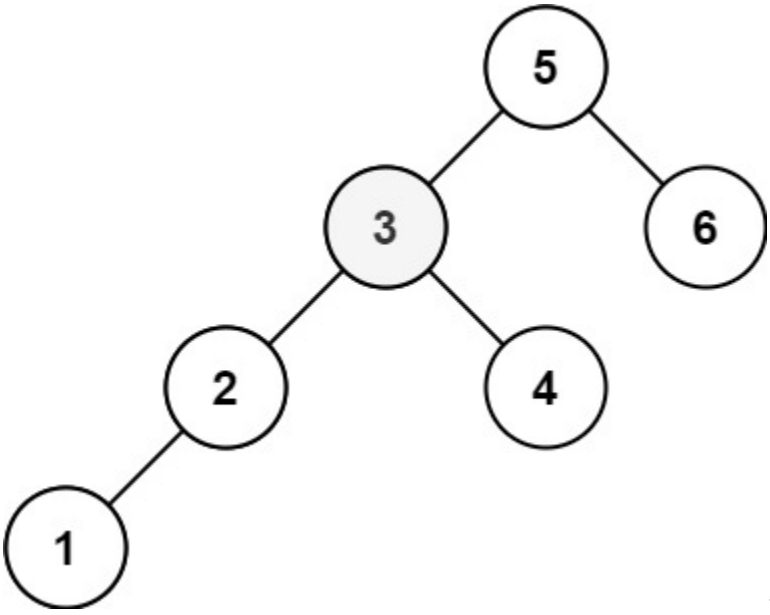
**Example 1:**




{ width=350 }

```
Input: root = [3,1,4,null,2], k = 1
Output: 1
```

**Example 2:**




{ width=350 }

```
Input: root = [5,3,6,2,4,null,null,1], k = 3
Output: 3
```

**Constraints:**

- The number of nodes in the tree is `n`.
- `1 <= k <= n <= 104`
- `0 <= Node.val <= 104`

**Follow up:** If the BST is modified often (i.e., we can do insert and delete operations) and you need to find the kth smallest frequently, how would you optimize?

**Hint** non-optimal store tree in sorted array; iterative dfs in-order and return the kth element processed, go left until null, pop, go right once;

# Lowest Common Ancestor of BST

**Description**

---

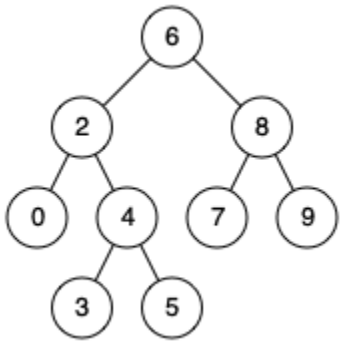
Category Tree Video https://youtu.be/gs2LMfuOR9k
Link https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree

---

Given a binary search tree (BST), find the lowest common ancestor (LCA) node of two given nodes in the BST.

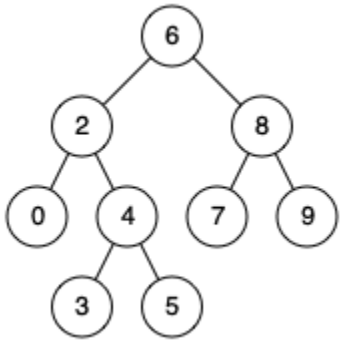
According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

**Example 1:**




```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
Output: 6
Explanation: The LCA of nodes 2 and 8 is 6.
```

**Example 2:**




```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
Output: 2
Explanation: The LCA of nodes 2 and 4 is 2, since a node can be a descendant of
itself according to the LCA definition.
```

**Example 3:**

```
Input: root = [2,1], p = 2, q = 1
Output: 2
```

**Constraints:**

- The number of nodes in the tree is in the range `[2, 105]`.
- `-109 <= Node.val <= 109`
- All `Node.val` are **unique**.
- `p != q`
- `p` and `q` will exist in the BST.

**Hint** compare p, q values to curr node, base case: one is in left, other in right subtree, then curr is lca;

# Implement Trie (Prefix Tree)

**Description**

Category Tree Video https://youtu.be/oobqoCJlHA0

Link https://leetcode.com/problems/implement-trie-prefix-tree

A **trie** (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

Implement the Trie class:

- `Trie()` Initializes the trie object.
- `void insert(String word)` Inserts the string `word` into the trie.
- `boolean search(String word)` Returns `true` if the string `word` is in the trie (i.e., was inserted before), and `false` otherwise.
- `boolean startsWith(String prefix)` Returns `true` if there is a previously inserted string `word` that has the prefix `prefix`, and `false` otherwise.

**Example 1:**

```
Input
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
Output
[null, null, true, false, true, null, true]

Explanation
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // return True
trie.search("app");     // return False
trie.startsWith("app"); // return True
trie.insert("app");
trie.search("app");     // return True
```

**Constraints:**

- `1 <= word.length, prefix.length <= 2000`
- `word` and `prefix` consist only of lowercase English letters.
- At most `3 * 104` calls **in total** will be made to `insert`, `search`, and `startsWith`.

**Hint** node has children characters, and bool if its an ending character, node DOESN'T have or need char, since root node doesn't have a char, only children;

# Add and Search Word

**Description**

---

Category Tree Video https://youtu.be/BTf05gs_8iU
Link https://leetcode.com/problems/add-and-search-word-data-structure-design

---

Design a data structure that supports adding new words and finding if a string matches any previously added string.

Implement the `WordDictionary` class:

- `WordDictionary()` Initializes the object.
- `void addWord(word)` Adds `word` to the data structure, it can be matched later.
- `bool search(word)` Returns `true` if there is any string in the data structure that matches `word` or `false` otherwise. `word` may contain dots `'.'` where dots can be matched with any letter.

**Example:**

**Input**

["WordDictionary","addWord","addWord","addWord","search","search","search","search"] [[],["bad"],["dad"],["mad"],["pad"],["bad"],[".ad"],["b.."]]

**Output**

[null,null,null,null,false,true,true,true]

**Explanation** WordDictionary wordDictionary = new WordDictionary(); wordDictionary.addWord("bad"); wordDictionary.addWord("dad"); wordDictionary.addWord("mad"); wordDictionary.search("pad"); // return False wordDictionary.search("bad"); // return True wordDictionary.search(".ad"); // return True wordDictionary.search("b.."); // return True

**Constraints:**

- `1 <= word.length <= 25`
- `word` in `addWord` consists of lowercase English letters.
- `word` in `search` consist of `'.'` or lowercase English letters.
- There will be at most `2` dots in `word` for `search` queries.
- At most `10<sup>4</sup>` calls will be made to `addWord` and `search`.

**Hint** if char = "." run search for remaining portion of word on all of curr nodes children;

# Word Search II

**Description**

---

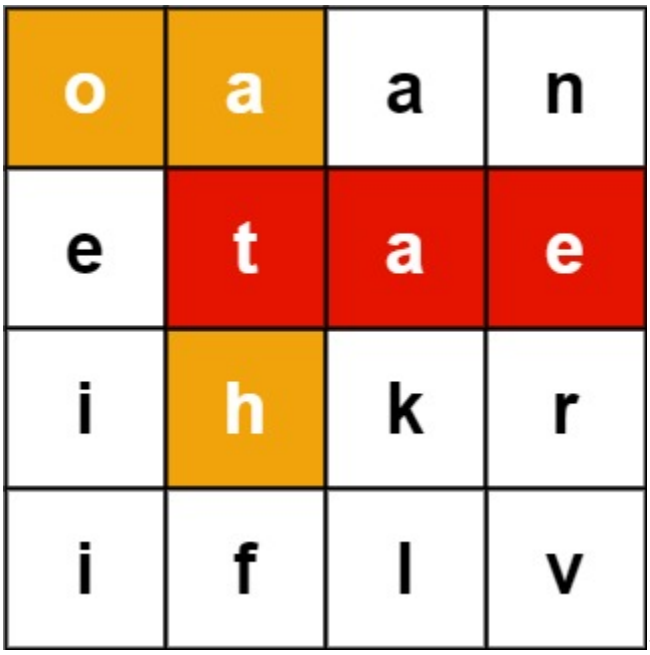
Category Tree Video https://youtu.be/asbcE9mZz_U
Link https://leetcode.com/problems/word-search-ii

---

Given an `m x n board` of characters and a list of strings `words`, return *all words on the board*.

Each word must 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 in a word.

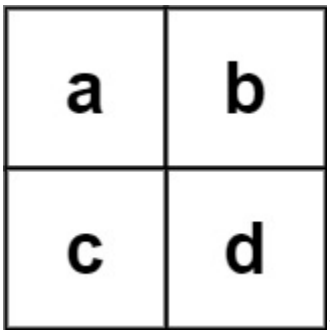
**Example 1:**



{ width=350 }

```
Input: board = [["o","a","a","n"],["e","t","a","e"],["i","h","k","r"],
["i","f","l","v"]], words = ["oath","pea","eat","rain"]
Output: ["eat","oath"]
```

**Example 2:**



{ width=175 }

```
Input: board = [["a","b"],["c","d"]], words = ["abcb"]
Output: []
```

**Constraints:**

- `m == board.length`
- `n == board[i].length`
- `1 <= m, n <= 12`
- `board[i][j]` is a lowercase English letter.
- `1 <= words.length <= 3 * 104`
- `1 <= words[i].length <= 10`
- `words[i]` consists of lowercase English letters.
- All the strings of `words` are unique.

**Hint** trick: I though use trie to store the grid, reverse thinking, instead store dictionary words, dfs on each cell, check if cell's char exists as child of root node in trie, if it does, update currNode, and check neighbors, a word could exist multiple times in grid, so don't add duplicates;

# Merge K Sorted Lists

**Description**

---

Category Heap Video https://youtu.be/q5a5OiGbT6Q
Link https://leetcode.com/problems/merge-k-sorted-lists

---

You are given an array of `k` linked-lists `lists`, each linked-list is sorted in ascending order.

*Merge all the linked-lists into one sorted linked-list and return it.*

**Example 1:**

```
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6
```

**Example 2:**

```
Input: lists = []
Output: []
```

**Example 3:**

```
Input: lists = [[]]
Output: []
```

**Constraints:**

- `k == lists.length`
- `0 <= k <= 104`
- `0 <= lists[i].length <= 500`
- `-104 <= lists[i][j] <= 104`
- `lists[i]` is sorted in **ascending order**.
- The sum of `lists[i].length` will not exceed `104`.

**Hint** we always want the min of the current frontier, we can store frontier in heap of size k for efficient pop/push; divide and conquer merging lists;

# Top K Frequent Elements

**Description**

Category Heap Video https://youtu.be/YPTqKIgVk-k
Link https://leetcode.com/problems/top-k-frequent-elements

---

Given an integer array `nums` and an integer `k`, return *the* `k` *most frequent elements*. You may return the answer in **any order**.

**Example 1:**

```
Input: nums = [1,1,1,2,2,3], k = 2
Output: [1,2]
```

**Example 2:**

```
Input: nums = [1], k = 1
Output: [1]
```

**Constraints:**

- `1 <= nums.length <= 105`
- `-104 <= nums[i] <= 104`
- `k` is in the range `[1, the number of unique elements in the array]`.
- It is **guaranteed** that the answer is **unique**.

**Follow up:** Your algorithm's time complexity must be better than `O(n log n)`, where n is the array's size.

**Hint** minheap that's kept at size k, if its bigger than k pop the min, by the end it should be left with k largest;

# Find Median from Data Stream

---

**Description**

---

Category Heap Video https://youtu.be/itmhHWaHupI
Link https://leetcode.com/problems/find-median-from-data-stream

---

The **median** is the middle value in an ordered integer list. If the size of the list is even, there is no middle value, and the median is the mean of the two middle values.

- For example, for `arr = [2,3,4]`, the median is `3`.
- For example, for `arr = [2,3]`, the median is `(2 + 3) / 2 = 2.5`.

Implement the MedianFinder class:

- `MedianFinder()` initializes the `MedianFinder` object.
- `void addNum(int num)` adds the integer `num` from the data stream to the data structure.
- `double findMedian()` returns the median of all elements so far. Answers within `10-5` of the actual answer will be accepted.

**Example 1:**

```
Input
["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"]
[[], [1], [2], [], [3], []]
Output
[null, null, null, 1.5, null, 2.0]

Explanation
MedianFinder medianFinder = new MedianFinder();
medianFinder.addNum(1);    // arr = [1]
medianFinder.addNum(2);    // arr = [1, 2]
medianFinder.findMedian(); // return 1.5 (i.e., (1 + 2) / 2)
medianFinder.addNum(3);    // arr[1, 2, 3]
medianFinder.findMedian(); // return 2.0
```

**Constraints:**

- `-105 <= num <= 105`
- There will be at least one element in the data structure before calling `findMedian`.
- At most `5 * 104` calls will be made to `addNum` and `findMedian`.

**Follow up:**

- If all integer numbers from the stream are in the range `[0, 100]`, how would you optimize your solution?
- If `99%` of all integer numbers from the stream are in the range `[0, 100]`, how would you optimize your solution?

**Hint** maintain curr median, and all num greater than med in a minHeap, and all num less than med in a maxHeap, after every insertion update median depending on odd/even num of elements;