1- Find pair that sums up to k

Description

Given an array of integers `arr` and an integer `k`, create a boolean function that checks if there exists two elements in `arr` such that we get `k` when we add them together.

Example 1:

- Input: `arr = [4, 5, 1, -3, 6]`, `k = 11`
- Output: `true`
- Explanation: `5 + 6` is equal to `11`

Example 2:

- Input: `arr = [4, 5, 1, -3, 6]`, `k = -2`
- Output: `true`
- Explanation: `1 + (-3)` is equal to `-2`

Example 3:

- Input: `arr = [4, 5, 1, -3, 6]`, `k = 8`
- Output: `false`
- Explanation: there is no pair that sums up to `8`

2- First repeating character

Description

Given a string `str`, create a function that returns the first repeating character.
If such character doesn't exist, return the null character `'\0'`.

Example 1:

- Input: `str = "inside code"`
- Output: `'i'`

Example 2:

- Input: `str = "programming"`
- Output: `'r'`

Example 3:

- Input: `str = "abcd"`
- Output: `'\0'`

Example 4:

- Input: `str = "abba"`
- Output: `'b'`

3- Remove duplicates

Description

Given an array of integers `arr`, create a function that returns an array containing the values of `arr` without duplicates (the order doesn't matter).

Example 1:

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

Example 2:

- Input: `arr = [1, 1, 1, 1, 1, 1, 1, 1]`
- Output: `[1]`

Example 3:

- Input: `arr = [4, 4, 2, 3, 2, 2, 4, 3]`
- Output: `[4, 2, 3]`

4- Find the duplicate

Description

Given an array of integers `arr` that contains `n+1` elements between `1` and `n` inclusive, create a function that returns the duplicate element (the element that appears more than once).
Assume that:
- There is only one duplicate number
- The duplicate number can be repeated more than once

Example 1:

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

Example 2:

- Input: `arr = [1, 4, 2, 2, 5, 2]`
- Output: `2`

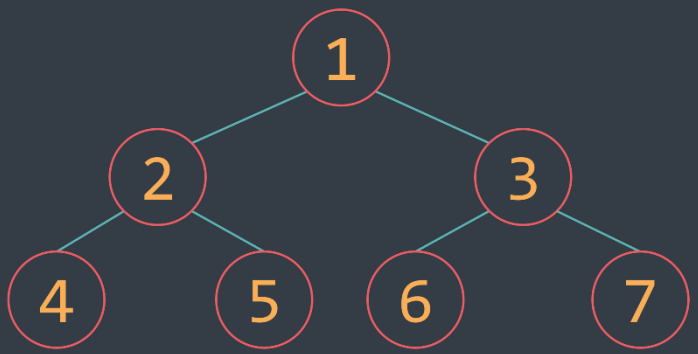
5- Tree depth first search (DFS)

Description

Given a binary tree of integers root, create 3 functions to print the tree nodes in preorder, inorder, and postorder traversal.
Preorder: print the root value, then print the left subtree, then print the right subtree.
Inorder: print the left subtree, then print the root value, then print the right subtree.
Postorder: print the left subtree, then print the right subtree, then print the root value.

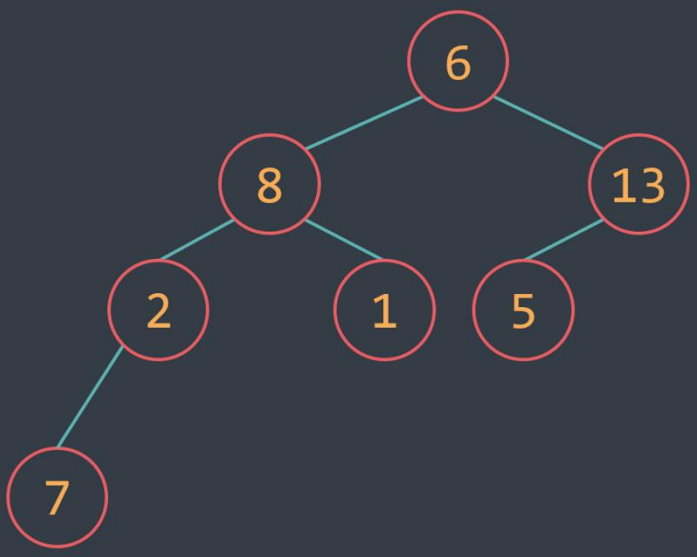
Example 1:

- Input: root = [1, 2, 3, 4, 5, 6, 7]
- Output:
  Preorder: 1 2 4 5 3 6 7
  Inorder: 4 2 5 1 6 3 7
  Postorder: 4 5 2 6 7 3 1



Example 2:

- Input: root = [6, 8, 13, 2, 1, 5, null, 7]
- Output:
  Preorder: 6 8 2 7 1 13 5
  Inorder: 7 2 8 1 6 5 13
  Postorder: 7 2 1 8 5 13 6



6- Maximum subarray

Description

Given a non-empty array of integers arr, create a function that returns the sum of the subarray that has the greatest sum.
We don't consider the empty array [] as a subarray.

Example 1:

- Input: arr = [2, 3, -6, 4, 2, -8, 3]
- Output: 6
- Explanation: the maximum subarray is [4, 2], its sum is 6

Example 2:

- Input: arr = [2, 3, -1, 4, -10, 2, 5]
- Output: 8
- Explanation: the maximum subarray is [2, 3, -1, 4], its sum is 8

Example 3:

- Input: arr = [-3, -1, -2]
- Output: -1
- Explanation: the maximum subarray is [-1], its sum is -1

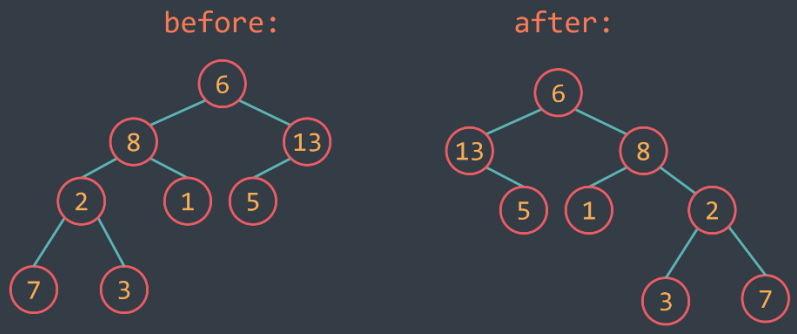
7- Reverse a binary tree

Description

Given a binary tree of integers `root`, create a function that reverses it left to right in-place.

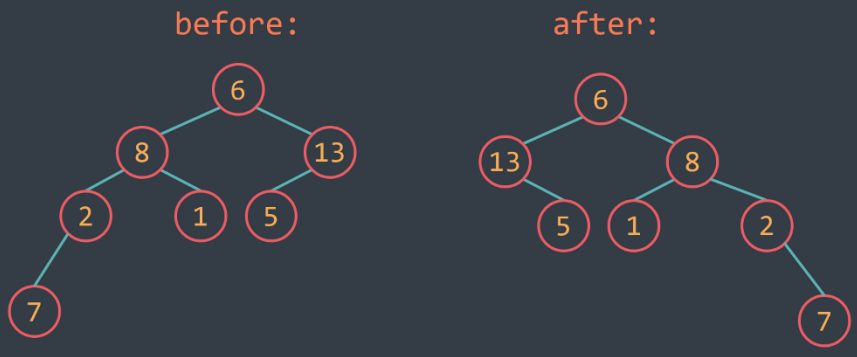
Example 1:

- Input: root = [1, 2, 3, 4, 5, 6, 7]
- Output: [1, 3, 2, 7, 6, 5, 4]
- Explanation:



Example 2:

- Input: root = [6, 8, 13, 2, 1, 5, null, 7]
- Output: [6, 13, 8, null, 5, 1, 2, null, null, null, null, null, 7]
- Explanation:



8- Longest substring without repeating characters

Description

Given a string `str` made of alphabetical letters only, create a function that returns the length of the longest substring without repeating characters.

Example 1:

- Input: `str = "abcdbeghef"`
- Output: `6`
- Explanation: the longest substring without repeating characters is "cdbegh", its length is 6

Example 2:

- Input: `str = "aaaaa"`
- Output: `1`
- Explanation: the longest substring without repeating characters is "a", its length is 1

Example 3:

- Input: `str = "eddy"`
- Output: `2`
- Explanation: the longest substring without repeating characters is "ed" (or "dy"), its length is 2

9- Reverse linked list

Description

Given a linked list of integers `list`, create a function that reverses it in-place without using an additional data structure.

Example 1:

- Input: `list = 5 -> 3 -> 6 -> 4 -> 7 -> null`
- Output: `7 -> 4 -> 6 -> 3 -> 5 -> null`

Example 2:

- Input: `list = 1 -> 2 -> 3 -> null`
- Output: `3 -> 2 -> 1 -> null`

10- Peak finding

Description

Given a non-empty array of integers arr, create a function that returns the index of a peak element. We consider an element as peak if it's greater than or equal to its neighbors, the next and previous element (assume that arr[-1] and arr[n] are equal to -∞). And if there are multiple peaks in arr, just return the index of one of them.

Example 1:

- Input: arr = [4, 5, 8, 3]
- Output: 2
- Explanation: arr[2] is a peak element because it's greater than or equal to arr[1], and greater than or equal to arr[3]

Example 2:

- Input: arr = [1, 3, 4, 7, 8]
- Output: 4
- Explanation: arr[4] is a peak element because it's greater than or equal to arr[3], which is it's only neighbor

Example 3:

- Input: arr = [1, 5, 2, 6, 6, 3]
- Output: 3
- Explanation: arr[3] is a peak element because it's greater than or equal to arr[2] and greater than or equal to arr[4] (other valid outputs would be 1 and 4, because arr[1] and arr[4] are also peak elements)

11- Palindrome linked list

Description

Given a linked list of integers `list`, create a boolean function that checks if it's a palindrome linked list in constant space complexity.

Example 1:

- Input: `list = 1 -> 4 -> 6 -> 5 -> 6 -> 4 -> 1 -> null`
- Output: `true`

Example 2:

- Input: `list = 8 -> 3 -> 1 -> 8 -> null`
- Output: `false`

Example 3:

- Input: `list = 6 -> null`
- Output: `true`

12- Longest possible palindrome

Description

Given a string `str` made of alphabetical letters only, create a function that returns the length of the longest palindrome string that can be made from letters of `str`

Example 1:

- Input: `str = "abbaba"`
- Output: `5`
- Explanation: one of the longest palindromes that can be made is "baaab", its length is 5

Example 2:

- Input: `str = "abbaaa"`
- Output: `6`
- Explanation: one of the longest palindromes that can be made is "baaaab", its length is 6

Example 3:

- Input: `str = "abbabab"`
- Output: `7`
- Explanation: one of the longest palindromes that can be made is "bbaaabb", its length is 7

Example 4:

- Input: `str = "abdccdceeebebc"`
- Output: `13`
- Explanation: one of the longest palindromes that can be made is "eedccbabccdee", its length is 13

13- Get substring index

Description

Given two strings str1 and str2, create a function that returns the first index where we can find str2 in str1. If we cannot find str2 in str1, the function must return -1.
Try to solve the problem without using the built-in .indexOf() method.

Example 1:

- Input: str1 = "inside", str2 = "side"
- Output: 2
- Explanation: we can find "side" in "inside" by starting from the index 2

Example 2:

- Input: str1 = "inside", str2 = "in"
- Output: 0
- Explanation: we can find "in" in "inside" by starting from the index 0

Example 3:

- Input: str1 = "inside", str2 = "code"
- Output: -1
- Explanation: we can't find "code" in "inside"

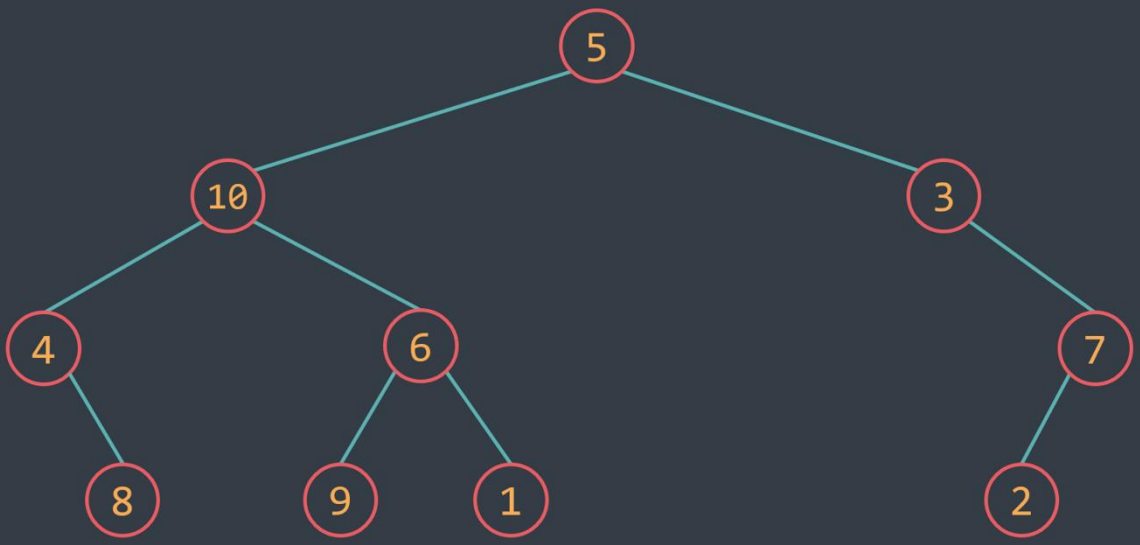
14- Tree breadth first search

Description

Given a binary tree `root`, create a function that prints its nodes in level order (level by level).

Example 1:

- Input: `root = [5, 10, 3, 4, 6, null, 7, null, 8, 9, 1, 2]`
- Output: `5 10 3 4 6 7 8 9 1 2`



15- Sort linked list

Description

Given a linked list of integers `list`, create a function that sorts it.
Note that the function returns nothing, the linked list must be sorted in-place.

Example 1:

- Input: `list = 4 -> 8 -> 1 -> 6 -> 2 -> 5 -> null`
- Output: `1 -> 2 -> 4 -> 5 -> 6 -> 8 -> null`

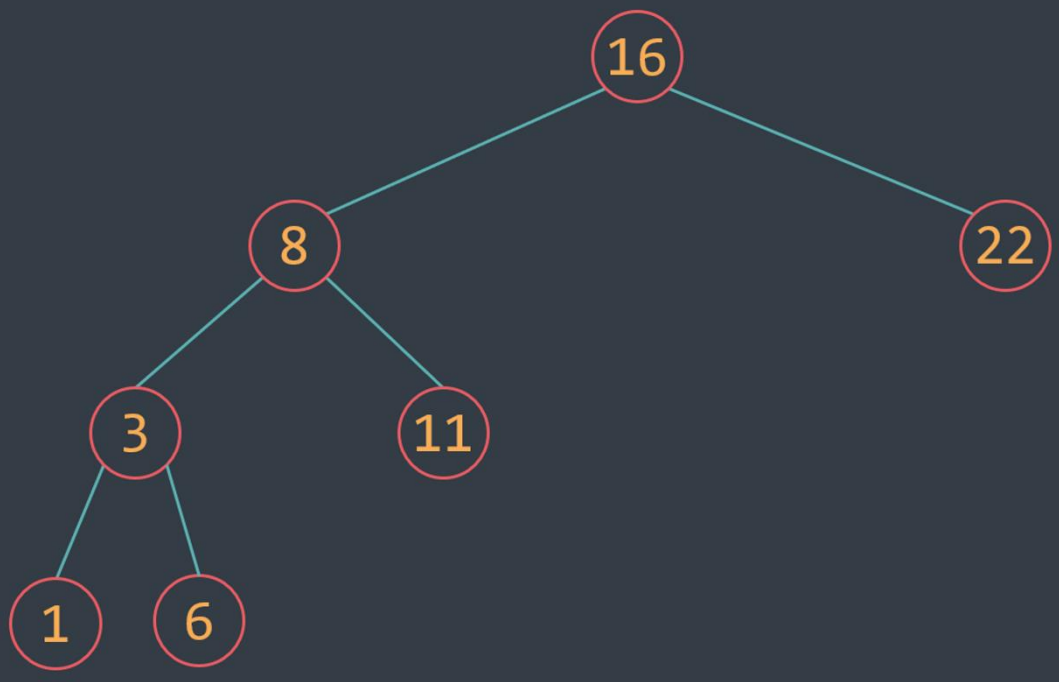
16- Valid binary search tree

Description

Given a binary tree `root`, create a boolean function that checks if it's a binary search tree. Note that in a binary search tree, every node must be strictly greater than all nodes on its left subtree, and strictly smaller than all nodes on its right subtree.

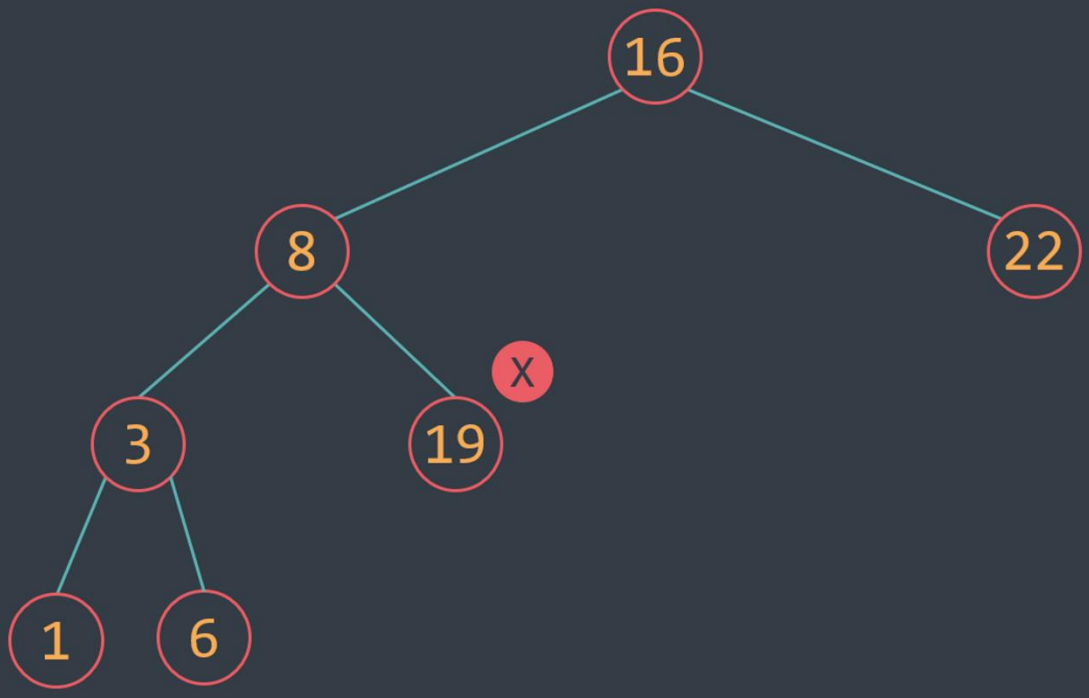
Example 1:

- Input: root = [16, 8, 22, 3, 11, null, null, 1, 6]
- Output: true
- Explanation: every node is strictly greater than all nodes on its left subtree, and strictly smaller than all nodes on its right subtree



Example 2:

- Input: root = [16, 8, 22, 3, 19, null, null, 1, 6]
- Output: false
- Explanation: it's not a valid binary search tree because 16 is smaller than 19



17- Minimum cost path in matrix

Description

Given a matrix of integers `matrix` of size n*m, where each element `matrix[i][j]` represents the cost of passing from that cell, create a function that returns the cost of the minimum cost path to go from the top left cell to the right bottom cell, knowing that you can only move to the right or to the bottom.

Example 1:

- Input: `matrix = [[3, 12, 4, 7, 10], [6, 8, 15, 11, 4], [19, 5, 14, 32, 21], [1, 20, 2, 9, 7]]`
- Output: `54`
- Explanation:

the orange cells represent the cheapest path

3 + 6 + 8 + 5 + 14 + 2 + 9 + 7 = 54

3	12	4	7	10
6	8	15	11	4
19	5	14	32	21
1	20	2	9	7

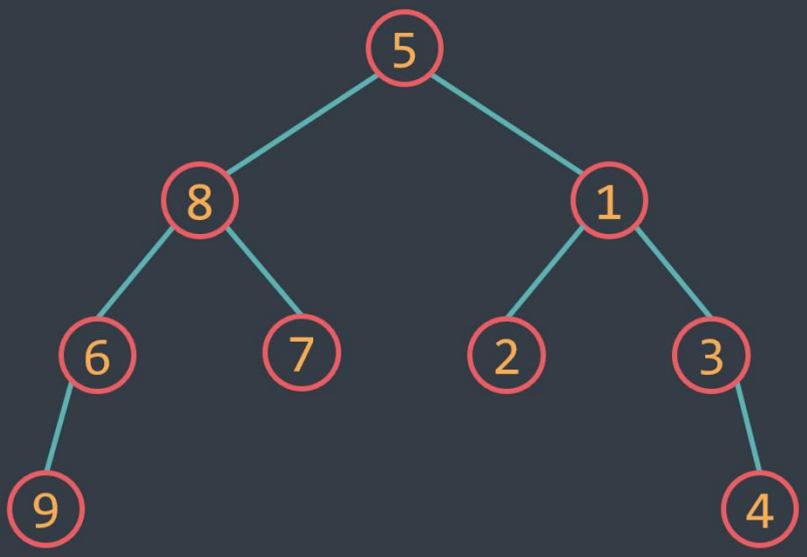
18- Balanced binary tree

Description

Given a binary tree if integers root, create a boolean function that checks if it's a balanced binary tree. A binary tree is considered as balanced if its left and right subtree are balanced, and the difference between their heights is at most 1

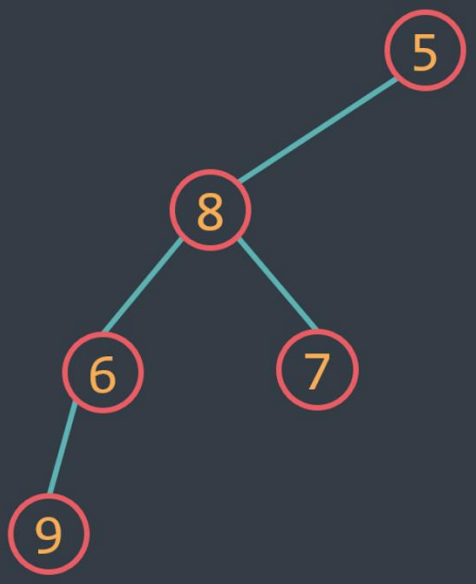
Example 1:

- Input: root = [5, 8, 1, 6, 7, 2, 3, 9, null, null, null, null, null, null, 4]
- Output: true



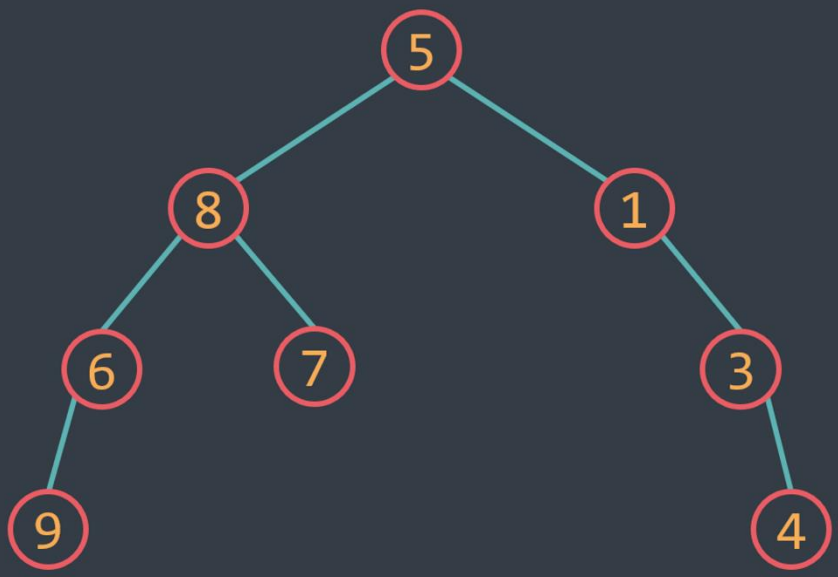
Example 2:

- Input: root = [5, 8, null, 6, 7, 9]
- Output: false
- Explanation: the height of the left subtree is 3, but the height of the right subtree is 0, the difference is greater than 1, we deduce that the tree is not balanced



Example 3:

- Input: root = [5, 8, 1, 6, 7, null, 3, 9, null, null, null, null, 4]
- Output: false
- Explanation: The height of the left subtree of the right subtree is 0, but the height of the right subtree of the right subtree is 2, the difference is greater than 1, we deduce that the right subtree is not balanced, it implies that the whole binary tree is not balanced



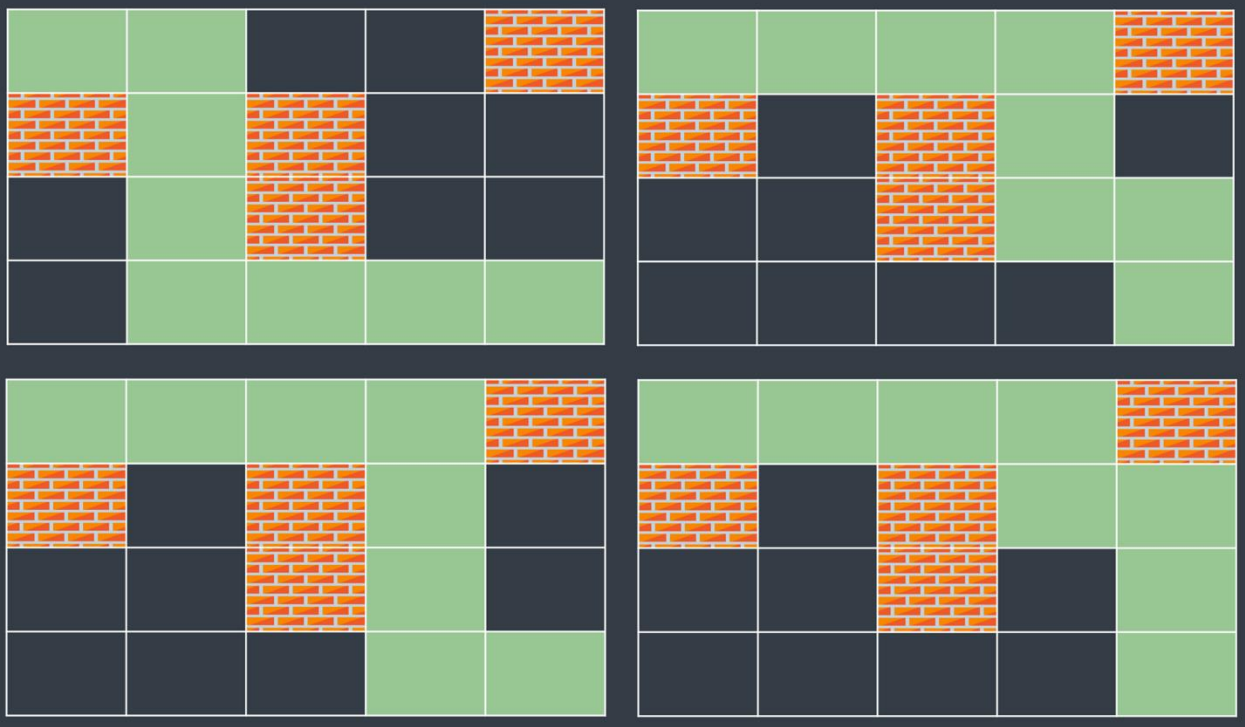
19- Paths in matrix

Description

Given a matrix of size n*m that contains only 0s and 1s, where 0 means that the cell is empty and 1 means that there is a wall in that cell, create a function that returns the number of paths that we can take to go from the top left cell to the right bottom cell, knowing that you can move to the right or to the bottom only.

Example 1:

- Input: matrix = [[0, 0, 0, 0, 1], [1, 0, 1, 0, 0], [0, 0, 1, 0, 0], [0, 0, 0, 0, 0]]
- Output: 4
- Explanation:



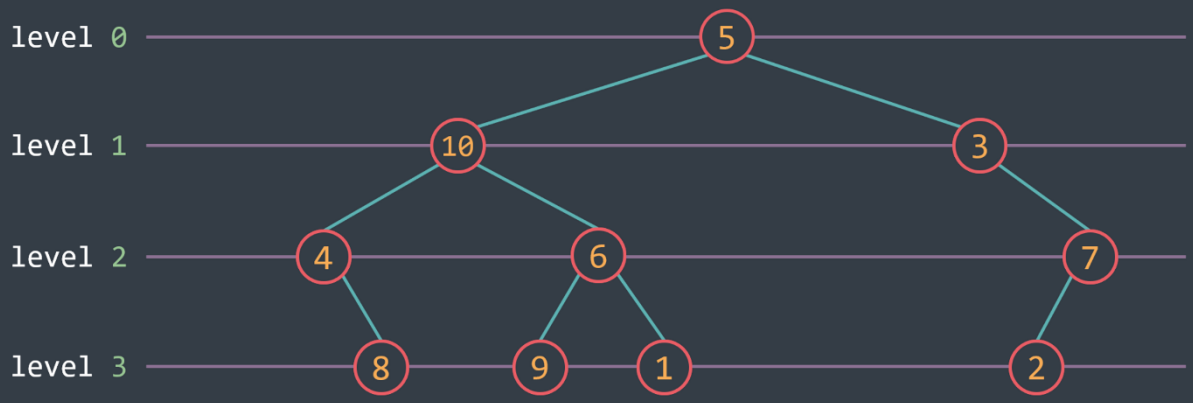
20- Tree breadth first search II

Description

Given a binary tree of integers `root`, create a function that returns an array where each element represents an array that contains the elements at the level i.

Example 1:

- Input: root = [5, 10, 3, 4, 6, null, 7, null, 8, 9, 1, 2]
- Output: [[5], [10, 3], [4, 6, 7], [8, 9, 1, 2]]
- Explanation:



21- Product of array except self

Description

Given an array of integers `arr` that has at least two elements, create a function that returns an array output where output[i] represents the product of all elements of `arr` except arr[i], and you are not allowed to use the division operator.

Example 1:

- Input: `arr = [2, 5, 3, 4]`
- Output: `[60, 24, 40, 30]`
- Explanation: output[0] = 5*3*4 = 60, output[1] = 2*3*4 = 24, output[2] = 2*5*4 = 40, output[3] = 2*5*3 = 30

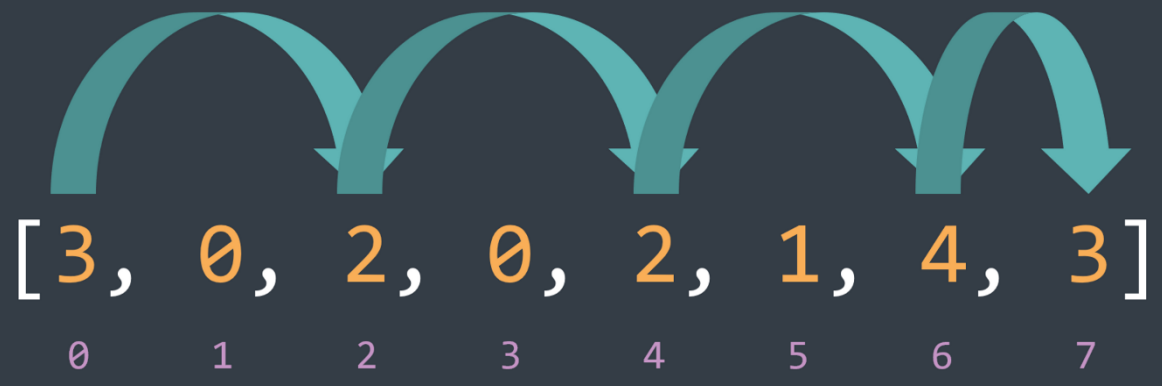
22- Jump to last index

Description

Given a non-empty array of non-negative integers `arr`, where each element represents the maximum jump that we can perform from that index, create a boolean function that checks if we can reach the last index starting from the first one.

Example 1:

- Input: `arr = [3, 0, 2, 0, 2, 1, 4, 3]`
- Output: `true`
- Explanation: we can for example jump from arr[0] to arr[2], then from arr[2] to arr[4], then from arr[4] to arr[6], then from arr[6] to arr[7] (the last index)



Example 2:

- Input: `arr = [5, 3, 2, 0, 1, 0, 4]`
- Output: `false`
- Explanation: we have no way to reach the last index

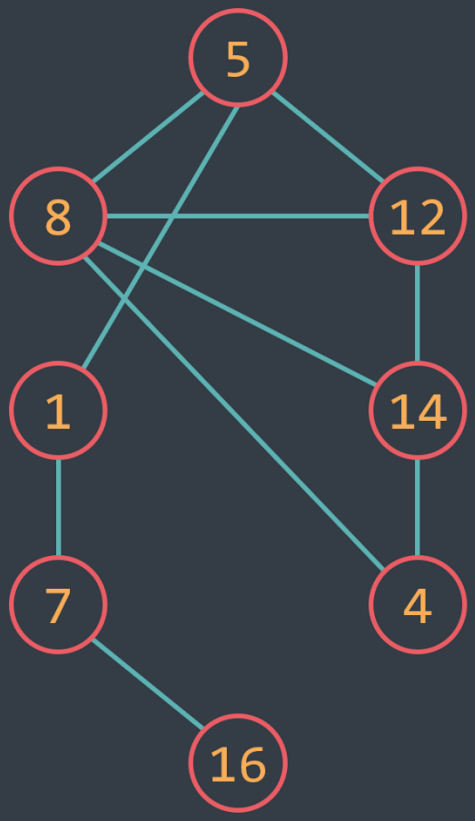
23- Graph depth first search

Description

Given an undirected graph of integers `graph`, represented by an adjacency list, and an integer `root`, create a function that prints its values in depth first search, by considering the vertex whose value is `root` as the arbitrary node.

Example 1:

- Input: `graph = {"5" : [8, 1, 12], "8" : [5, 12, 14, 4], "12" : [5, 8, 14], "14" : [8, 12, 4], "4" : [8, 14], "1" : [5, 7], "7" : [1, 16], "16" : [7]}`, `root = 5`
- Output: `5 8 12 14 4 1 7 16`.



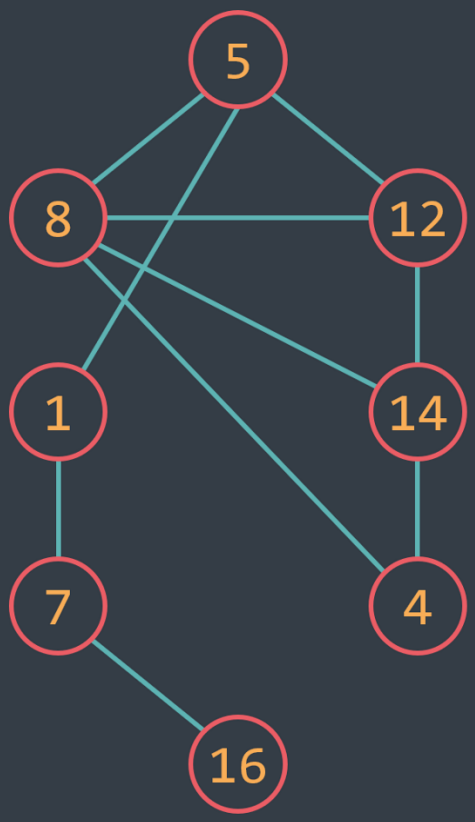
24- Graph breadth first search

Description

Given an undirected graph of integers graph, represented by an adjacency list, and an integer root, create a function that prints its values in breadth first search, by considering the vertex whose value is root as the arbitrary node.

Example 1:

- Input: graph = {"5" : [8, 1, 12], "8" : [5, 12, 14, 4], "12" : [5, 8, 14], "14" : [8, 12, 4], "4" : [8, 14], "1" : [5, 7], "7" : [1, 16], "16" : [7]}, root = 5
- Output: 5 8 1 12 14 4 7 16



25- String subsequences

Description

Given a string `str`, create a function that returns all its possible subsequences, the order doesn't matter.

Example 1:

- Input: `str = "abcd"`
- Output: `["abcd", "abc", "abd", "ab", "acd", "ac", "ad", "a", "bcd", "bc", "bd", "b", "cd", "c", "d", ""]`
- Explanation:

"abcd" "abcd" "abcd" "abcd"
"abcd" "abcd" "abcd" "abcd"
"abcd" "abcd" "abcd" "abcd"
"abcd" "abcd" "abcd" "abcd"

26- Valid brackets

Description

Given a string `str` made of `3` types of brackets only `(){}[]`, create a function that checks if the string is valid. The string is considered as valid if all opening brackets are closed with the same type of brackets, and in the correct order.

Example 1:

- Input: `str = "{()}"`
- Output: `true`

Example 2:

- Input: `str = "{(})"`
- Output: `false`

Example 3:

- Input: `str = "[}"`
- Output: `false`

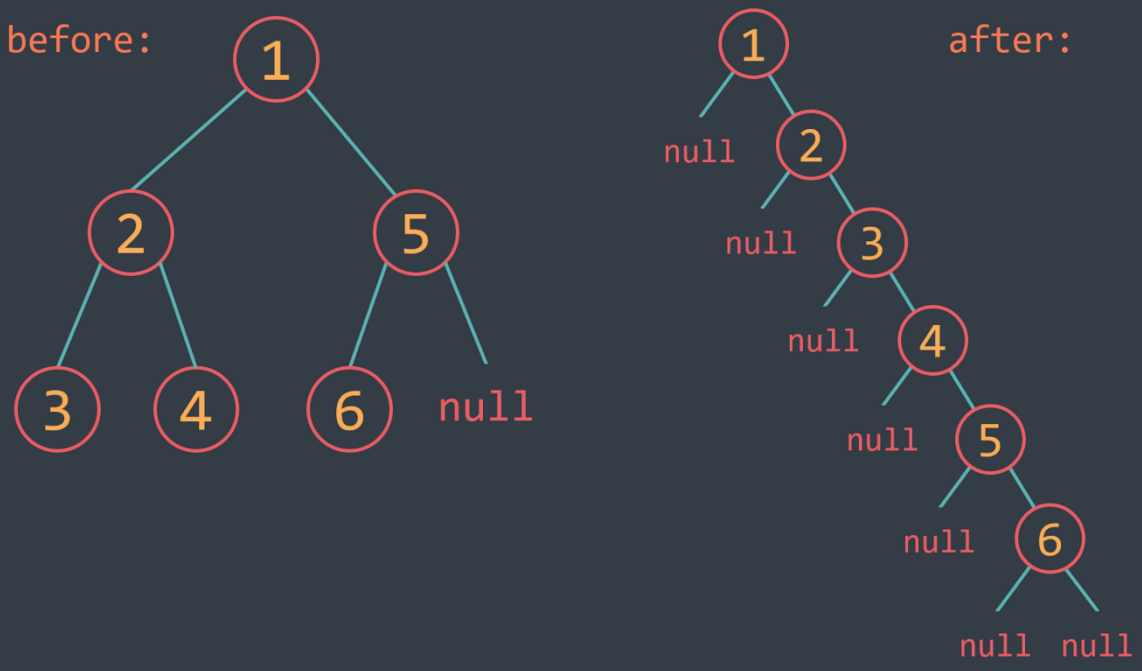
27- Flatten binary tree

Description

Given a binary tree root, create a function that flattens it to a linked list in-place by following the preorder traversal.

Example 1:

- Input: root = [1, 2, 5, 3, 4, 6]
- Output: [1, null, 2, null, 3, null, 4, null, 5, null, 6]
- Explanation:



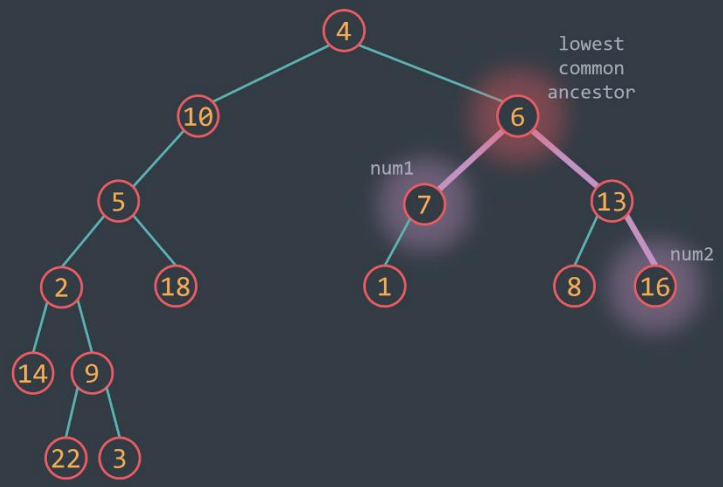
28- Lowest common ancestor

Description

Given a binary tree root and two integers num1 and num2, create a function that returns the lowest common ancestor of num1 and num2. The lowest common ancestor is the deepest node in root that has both num1 and num2 as descendants, and we consider a node as a descendant of itself.
Note that all values are unique and that num1 and num2 always exist in the tree.

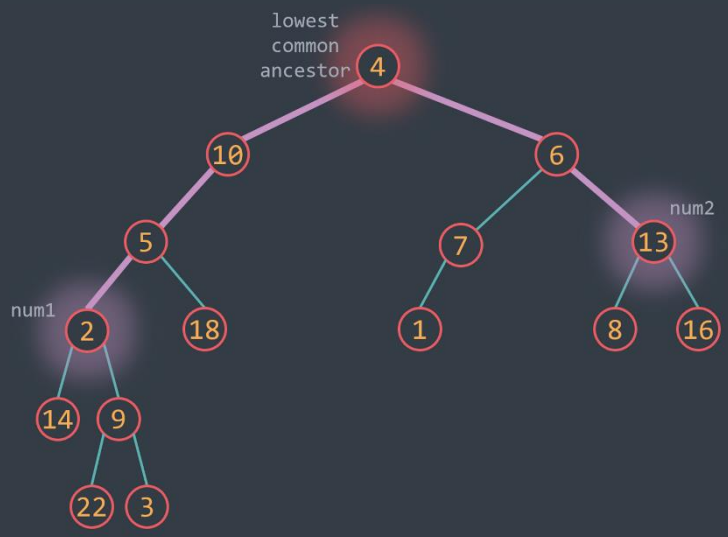
Example 1:

- Input: root = [4, 10, 6, 5, null, 7, 13, 2, 18, 1, null, 8, 16, 14, 9, null, null, null, null, null, null, null, null, null, null, 22, 3], num1 = 7, num2 = 16
- Output: 6
- Explanation:



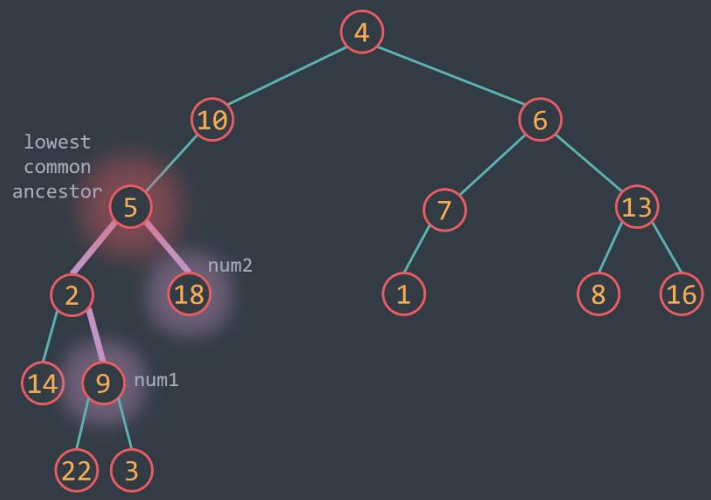
Example 2:

- Input: root = [4, 10, 6, 5, null, 7, 13, 2, 18, 1, null, 8, 16, 14, 9, null, null, null, null, null, null, null, null, null, null, 22, 3], num1 = 2, num2 = 13
- Output: 4
- Explanation:



Example 3:

- Input: root = [4, 10, 6, 5, null, 7, 13, 2, 18, 1, null, 8, 16, 14, 9, null, null, null, null, null, null, null, null, null, null, 22, 3], num1 = 9, num2 = 18
- Output: 5
- Explanation:



29- Minimum in rotated sorted array

Description

Given a non-empty rotated sorted array of integers `arr` that has no duplicates, create a function that returns the minimum.
Note that the array is sorted in ascending order, and that it's rotated by an unknown number of positions to the right.

Example 1:

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

Example 2:

- Input: `arr = [45, 66, 70, 71, 80, 89, -6, -2, -1, 0, 3, 5, 6, 8, 11, 15, 20, 23]`
- Output: `-6`

30- Add two linked lists

Description

Given two linked lists `list1` and `list2` that represent two positive integers, create a function that returns the linked list that represents their sum.
Note that the two integers don't contain any leading zero except the number `0` itself, and that each node contains one digit only, and that the digits are stored in reverse order.

Example 1:

- Input: `list1 = 3 -> 2 -> 1 -> null`, `list2 = 5 -> 9 -> 4 -> 3 -> null`
- Output: `8 -> 1 -> 6 -> 3 -> null`
- Explanation: 123 + 3495 = 3618

Example 2:

- Input: `list1 = 1 -> 6 -> 5 -> 4 -> null`, `list2 = 4 -> 8 -> 2 -> 7 -> 9 -> null`
- Output: `5 -> 4 -> 8 -> 1 -> 0 -> 1 -> null`
- Explanation: 4561 + 97284 + 101845

31- Ways to climb stairs

Description

Given a staircase of `n` steps, and a set of possible steps that we can climb at a time named `possibleSteps`, create a function that returns the number of ways that a person can take to reach the top of the staircase.

Example 1:

- Input: `n = 5, possibleSteps = {1, 2}`
- Output: `8`
- Explanation: possible ways to reach the top are: 1 1 1 1 1, 1 1 1 2, 1 1 2 1, 1 2 1 1, 2 1 1 1, 1 2 2, 2 1 2, 2 2 1

Example 2:

- Input: `n = 7, possibleSteps = {2, 3, 4}`
- Output: `5`
- Explanation: possible ways to reach the top are: 2 2 3, 2 3 2, 3 2 2, 3 4, 4 3

Example 3:

- Input: `n = 10, possibleSteps = {2, 4, 5, 8}`
- Output: `11`
- Explanation: possible ways to reach the top are: 2 2 2 2 2, 2 2 2 4, 2 2 4 2, 2 4 2 2, 4 2 2 2, 2 4 4, 4 2 4, 4 4 2, 5 5, 2 8, 8 2

32- Subsets that sum up to k

Description

Given an array `arr` of strictly positive integers, and an integer `k`, create a function that returns the number of subsets of `arr` that sum up to `k`.

Example 1:

- Input: `arr = [1, 2, 3, 1]`, `k = 4`
- Output: `3`
- Explanation: subsets that sum up to k are [1, 2, 1], [1, 3], and [3, 1]

Example 2:

- Input: `arr = [1, 2, 3, 1, 4]`, `k = 6`
- Output: `4`
- Explanation: subsets that sum up to k are [1, 2, 3], [1, 1, 4], [2, 3, 1], and [2, 4]

Example 3:

- Input: `arr = [2, 4, 2, 6, 8]`, `k = 7`
- Output: `0`
- Explanation: there are no subsets that sum up to k

33- Ways to decode

Description

Given a string `str` made of digits, create a function that returns with how many ways we can decode it, knowing that `1` is decoded as the letter A, `2` is decoded as the letter B, and so on until `26` that is decoded as the letter Z.

Example 1:

- Input: `str = "6324120129"`
- Output: `4`
- Explanation: we can decode the string as "FCBDATABI" (6 3 2 4 1 20 1 2 9), or "FCBDATLI" (6 3 2 4 1 20 12 9), or "FCXATABI" (6 3 24 1 20 1 2 9), or "FCXATLI" (6 3 24 1 20 12 9)

Example 2:

- Input: `str = "12"`
- Output: `2`
- Explanation: we can decode the string as "AB" (1 2), or "L" (12)

Example 3:

- Input: `str = "12345"`
- Output: `3`
- Explanation: we can decode the string as "ABCDE" (1 2 3 4 5), or "AWDE" (1 23 4 5), or "LCDE" (12 3 4 5)

34- Remove node from binary search tree

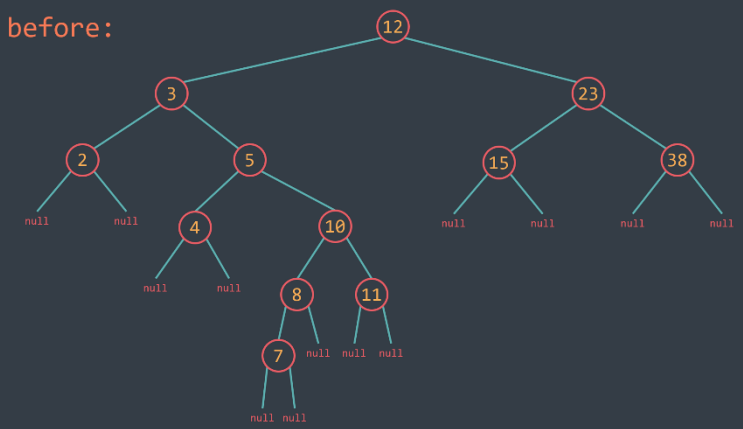
Description

Given a binary search tree `root`, and an integer `num`, create a function that deletes the node that contains `num` then returns `root`.

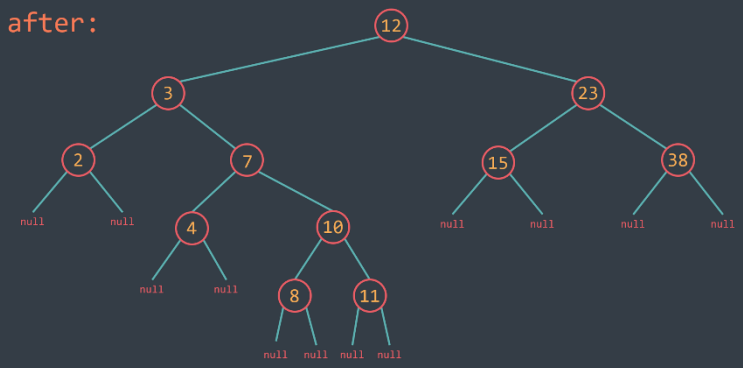
Example 1:

- Input: root = [12, 3, 23, 2, 5, 15, 38, null, null, 4, 10, null, null, null, null, null, null, 8, 11, 7, null, null, null], num = 5
- Output: [12, 3, 23, 2, 7, 15, 38, null, null, 4, 10, null, null, null, null, null, 8, 11]
- Explanation:

before:



after:



35- Array permutations

Description

Given an array of integers `arr`, create a function that returns all its possible permutations without duplicates, the order doesn't matter.

Example 1:

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

Example 2:

- Input: `arr = [1, 2, 2]`
- Output: `[[1, 2, 2], [2, 1, 2], [2, 2, 1]]`

36- Longest common subsequence

Description

Given two strings str1 and str2, create a function that returns the length of their longest common subsequence, in other words, the subsequence that is present in both of them.

Example 1:

- Input: str1 = "abdacbab", str2 = "acebfca"
- Output: 4
- Explanation: the length of the longest common subsequence of str1 and str2 is 4, this one for example: "abca"

Example 2:

- Input: str1 = "cbebaff", str2 = "aeddbggf"
- Output: 3
- Explanation: the length of the longest common subsequence of str1 and str2 is 3, this one for example: "ebf"

Example 3:

- Input: str1 = "abebafba", str2 = "cddghcd"
- Output: 0
- Explanation: the only common subsequence is "", the empty string, its length is 0

37- Longest consecutive sequence

Description

Given an array of integers `arr`, create a function that returns the length of the longest consecutive sequence that can we found in `arr`.

Example 1:

- Input: `arr = [14, 1, 8, 4, 0, 13, 6, 9, 2, 7]`
- Output: `4`
- Explanation: the longest consecutive sequence is 6 7 8 9

Example 2:

- Input: `arr = [4, 5, 2, 6, 5, 4, 1, -5, 0, 4]`
- Output: `3`
- Explanation: the longest consecutive sequence is 4 5 6

Example 3:

- Input: `arr = [5, 10]`
- Output: `1`
- Explanation: the longest consecutive sequence is 5 (also 10)

38- Edit distance

Description

Given two words as strings word1 and word2, create a function that returns the minimum number of operations required to convert word1 to word2. Note that we have 3 possible operations, we can either insert a character, or remove a character, or replace a character.

Example 1:

- Input: str1 = "inside", str2 = "index"
- Output: 3
- Explanation: to convert "inside" to "index", we need 3 operations, we remove the 's', we remove the 'i', and we insert 'x' (inside -> inide -> inde -> index)

Example 2:

- Input: str1 = "eagles", str2 = "algiers"
- Output: 4
- Explanation: to convert "eagles" to "algiers", we need 4 operations, we remove the 'e', we insert 'l', we replace the second 'l' by 'i', and we insert 'r' (eagles -> agles -> algles -> algies -> algiers)

39- Longest common substring

Description

Given two strings `str1` and `str2` made of alphabetical letters only, create a function that returns the length of their longest common substring.

Example 1:

- Input: `str1 = "opposite", str2 = "position"`
- Output: `5`
- Explanation: the longest common substring of str1 and str2 is "posit"

Example 2:

- Input: `str1 = "printer", str2 = "external"`
- Output: `3`
- Explanation: the longest common substring of str1 and str2 is "ter"

Example 3:

- Input: `str1 = "table", str2 = "dog"`
- Output: `0`
- Explanation: the longest common substring of str1 and str2 is ""

40- Smallest number after removing k digits

Description

Given a positive integer k and string num that represents a positive integer, create a function that returns as a string, the smallest number that can be made by removing k digits from num. Note that both input and output don't contain leading zeroes, except for the number 0 itself.

Example 1:

- Input: num = "825563", k = 2
- Output: "2553"

Example 2:

- Input: num = "83866", k = 3
- Output: "36"

Example 3:

- Input: num = "20050", k = 1
- Output: "50"

41- Insert interval

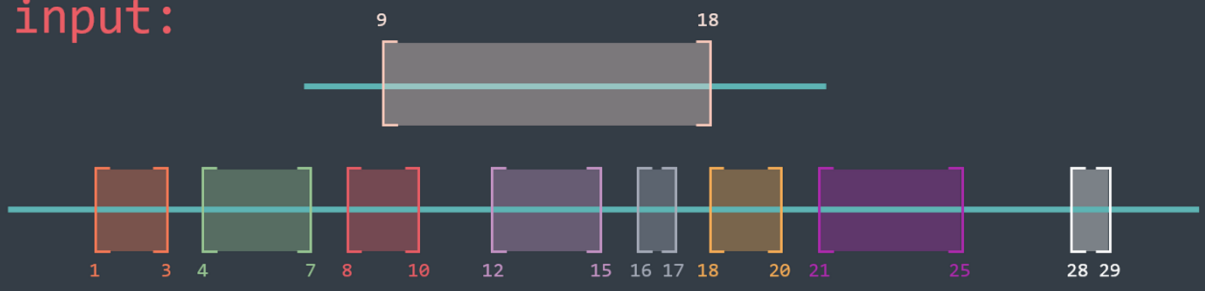
Description

Given an interval `newInterval` and an array of `intervals`, create a function that inserts that `newInterval` in the array, and to merge if necessary. Note that the `intervals` in the array are non-overlapping, and are sorted according to their starting point.

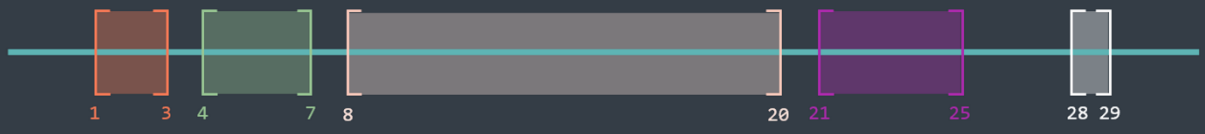
Example 1:

- Input: `intervals = [[1, 3], [4, 7], [8, 10], [12, 15], [16, 17], [18, 20], [21, 25], [28, 29]]`, `newInterval = [9, 18]`
- Output: `[[1, 3], [4, 7], [8, 20], [21, 25], [28, 29]]`
- Explanation:

input:



output:



42- Merge intervals

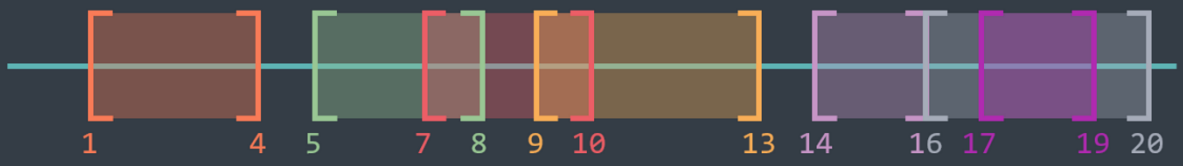
Description

Given an array of `intervals`, create a function that returns an array where overlapping intervals are merged.

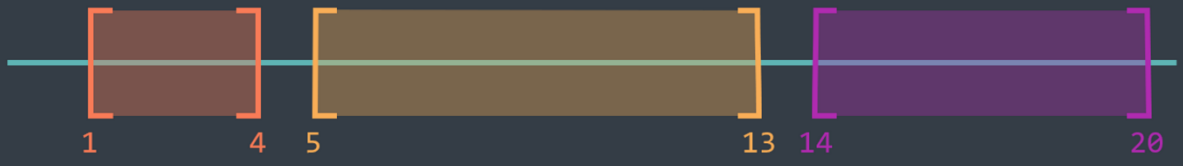
Example 1:

- Input: `intervals = [[1, 4], [5, 8], [7, 10], [9, 13], [14, 16], [16, 20], [17, 19]]`
- Output: `[[1, 4], [5, 13], [14, 20]]`
- Explanation:

input:



output:



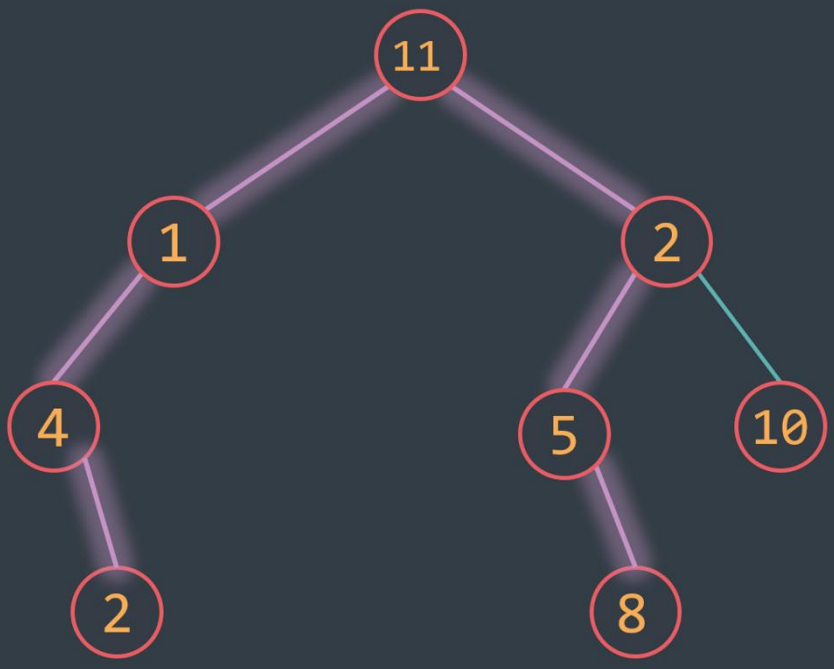
43- Maximum path sum

Description

Given a non-empty binary tree `root`, create a function that returns the maximum path sum. Note that for this problem, a path goes from a node to another one by traversing edges between them, and that the path must have at least one node, and it does not have to pass by the `root`.

Example 1:

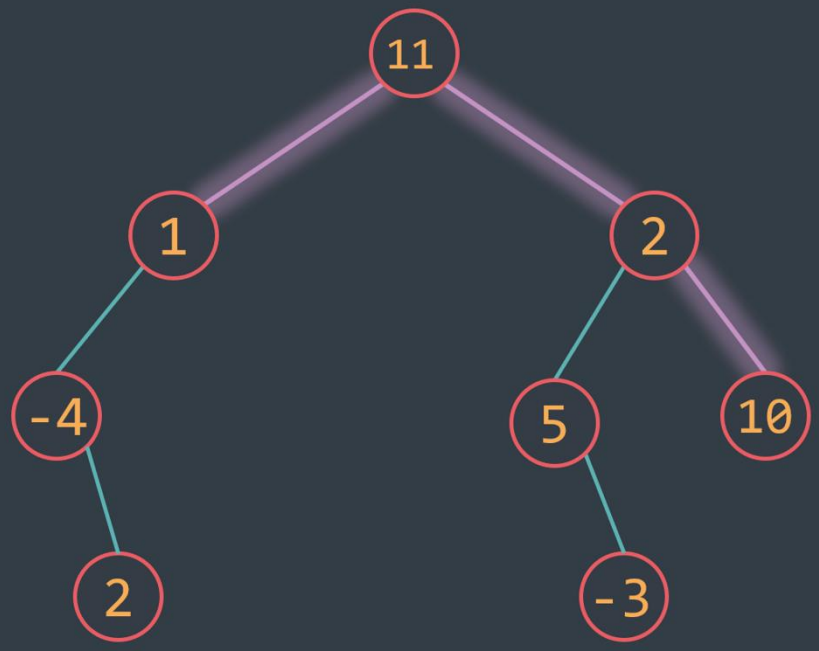
- Input: root = [11, 1, 2, 4, null, 5, 10, null, 2, null, 8]
- Output: 33
- Explanation:



Example 2:

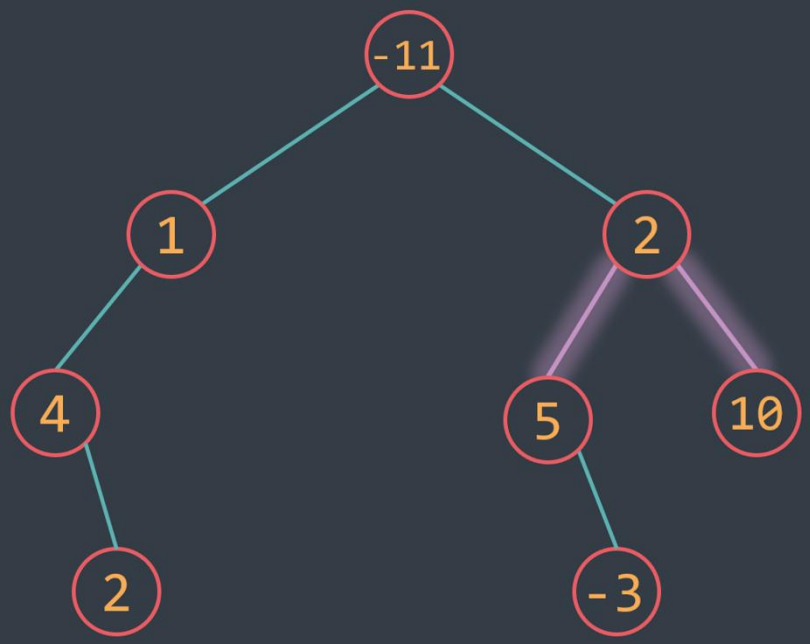
- Input: root = [11, 1, 2, -4, null, 5, 10, null, 2, null, -3]
- Output: 24

- Explanation:



Example 3:

- Input: root = [-11, 1, 2, 4, null, 5, 10, null, 2, null, -3]
- Output: 17
- Explanation:



44- 0-1 Knapsack

Description

Given values and weights of n items, we want to put them in a knapsack in a way to have the greatest possible value but without exceeding the maxWeight, so you are asked to create a function that returns that greatest possible value.

Example 1:

- Input: values = [20, 15, 25, 10], weights = [6, 5, 10, 3], maxWeight = 10
- Output: 30
- Explanation: we can get 30 by taking the elements at index 0 (value: 20, weight: 6) and index 3 (value: 10, weight: 3), the sum of weights is 9kg, it doesn't exceed the maxWeight

45- Shortest palindrome

Description

Given a string `str`, create a function that returns the shortest palindrome that we can get by adding characters in front of `str`. Note that a palindrome is a string that can be read in the same way from the left and from the right.

Example 1:

- Input: `str = "acbcabcbb"`
- Output: `"bbcbacbcabcbb"`

Example 2:

- Input: `str = "bcaacb"`
- Output: `"bcaacb"`

Example 3:

- Input: `str = "abcd"`
- Output: `"dcbabcd"`

46- Coin change

Description

Given an amount of money, and a set of possible coins, create a function that returns the minimum number of coins needed to make that amount. Note that if there exists no combination to do so, the function must return -1.

Example 1:

- Input: amount = 15, coins = {2, 3, 7}
- Output: 4
- Explanation: we can make 15 by taking a coin of value 7, two coins of value 3, a coin of value 2.

Example 2:

- Input: amount = 34, coins = {5, 13}
- Output: -1
- Explanation: there is no combination of coins to make 34.

47- Word search

Description

Given a board of characters and a string word, create a boolean function that checks if we can find the word in the board. Note that the word must be made with adjacent cells (horizontal and vertical neighbours), and that the same cell can be used only once.

Example 1:

- Input: board = [['K','I','N','T'], ['B','I','N','S'], ['G','N','Y','I'], ['U','O','E','D'], ['D','I','B','V'], ['H','I','R','T']], word = "INSIDE"
- Output: true
- Explanation:

K	I	N	T
B	I	N	S
G	N	Y	I
U	O	E	D
D	I	B	V
H	I	R	T

Example 2:

- Input: board = [['K','I','N','T'], ['B','I','N','S'], ['G','N','Y','I'], ['U','O','E','D'], ['D','I','B','V'], ['H','I','R','T']], word = "CODE"
- Output: false
- Explanation: the string "CODE" doesn't exist in the board

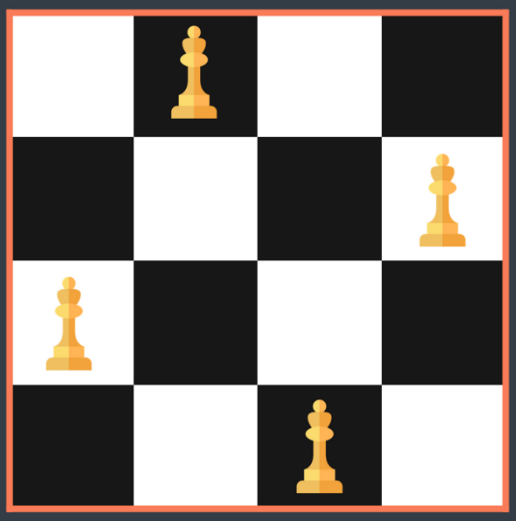
48- N-queens

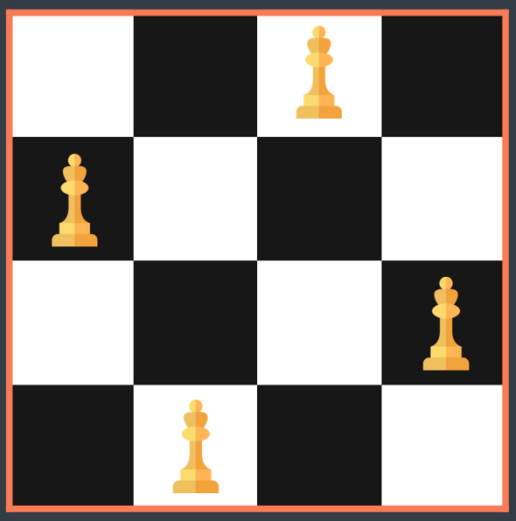
Description

Given a positive integer n, create a function that returns the number of possible ways to put n queens in an n*n chessboard such that no two queens attack each other.
Note that two queens attack each other when they are on the same row, column, or diagonal.

Example 1:

- Input: n = 4
- Output: 2
- Explanation:





49- Word ladder

Description

Given two words beginWord and endWord, and a list of words wordList, create a function that returns the length of the shortest transformation sequence from beginWord to endWord.
Note that:
-Only one letter can be changed at a time.
-Each intermediate word in the sequence must be in the wordList.
-The function returns 0 if there is no possible transformation sequence.
-All words have the same length.
-All words contain lowercase alphabetic characters.
-There are no duplicates in the wordList.
-beginWord and endWord are non-empty, and they are different.

Example 1:

- Input: beginWord = "hit", endWord = "cog", wordList = ["hot", "dot", "dog", "lot", "log", "cog"]
- Output: 5
- Explanation: one short possible transformation sequence is: "hit" -> "hot" -> "dot" -> "dog" -> "cog"

Example 2:

- Input: beginWord = "hit", endWord = "cog", wordList = ["dot", "dog", "lot", "log", "cog"]
- Output: 0
- Explanation: there is no way to go from "hit" to "cog"

50- Longest increasing subsequence

Description

Given an array of integers arr, create a function that returns the length of the longest increasing subsequence. Note that elements of that sequence must be strictly increasing and are not obliged to be contiguous.

Example 1:

- Input: arr = [9, 5, 1, 8, 6, 3, 0, 15, 5, 12, 4]
- Output: 4
- Explanation: the longest increasing subsequence is 1 3 5 12, its length is 4

Example 2:

- Input: arr = [7, 5, 2, 4, 7, 2, 3, 6, 4, 5, 12, 1, 7]
- Output: 5
- Explanation: the longest increasing subsequence is 2 3 4 5 12, its length is 5

Example 3:

- Input: arr = [5, 5, 5, 5]
- Output: 1
- Explanation: the subsequence must be strictly increasing, so the longest one here is 5, its length is 1