Questions/Answers

BlackRock

Description | Accepted | Note | Editorial | Solutions | Submissions

3004. Maximum Subtree of the Same Color

Solved

Premium

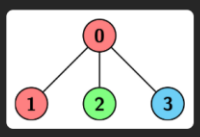
Medium | Topics | Companies | Hint

You are given a 2D integer array `edges` representing a tree with `n` nodes, numbered from `0` to `n - 1`, rooted at node `0`, where `edges[i] = [u_i, v_i]` means there is an edge between the nodes `v_i` and `u_i`.

You are also given a **0-indexed** integer array `colors` of size `n`, where `colors[i]` is the color assigned to node `i`.

We want to find a node `v` such that every node in the subtree of `v` has the **same** color.

Return *the size of such subtree with the **maximum** number of nodes possible.*



Example 1:

```
Input: edges = [[0,1],[0,2],[0,3]], colors = [1,1,2,3]
Output: 1
Explanation: Each color is represented as: 1 -> Red, 2 -> Green, 3 -> Blue. We can see that the subtree rooted at node 0 has children with different colors. Any other subtree is of the same color and has a size of 1. Hence, we return 1.
```

Example 2:

22 | 4 | 1 Online

Submit

Code

Python3 • Auto

```python
class Solution:
    def maximumSubtreeSize(self, edges: List[List[int]], colors: List[int]) -> int:
        n = len(colors)
        # Step 1: Build the tree as an adjacency list
        graph = defaultdict(list)
        for u, v in edges:
            graph[u].append(v)
            graph[v].append(u)

        self.max_size = 1
        # Step 2: DFS to traverse the tree and check subtrees
        def dfs(node, parent):
            size = 1  # size of the current subtree
            is_same = True  # flag to check if all nodes in this subtree have the same color
            for child in graph[node]:
                if child == parent:
                    continue  # avoid revisiting the parent
                child_size, child_same = dfs(child, node)
                # Step 3: Check if the child's subtree is the same color as the current node
                if not child_same or colors[child] != colors[node]:
                    is_same = False
                size += child_size
            # Step 4: Update the maximum size if this subtree is valid
            if is_same:
                self.max_size = max(self.max_size, size)
            return size, is_same
        # Step 5: Start DFS from the root node (0)
        dfs(0, -1)
        # Step 6: Return the largest valid subtree size
        return self.max_size

# if __name__ == "__main__":
#     import sys
#     input = sys.stdin.read
#     data = input().splitlines()
#     n = int(data[0])
```

Saved

Ln 31, Col 1

Testcase | Test Result

```python
# if __name__ == "__main__":
#     import sys
#     input = sys.stdin.read
#     data = input().splitlines()
#     n = int(data[0])
#     edges = [list(map(int, line.split())) for line in data[1:n]]
#     colors = list(map(int, data[n].split()))
#     sol = Solution()
#     print(sol.maxSubtreeWithSameColor(edges, colors))
```

BlackRock

Description | Note | Editorial | Solutions | Submissions

2291. Maximum Profit From Trading Stocks

Premium

Solved

Medium | Topics | Companies | Hint

You are given two **0-indexed** integer arrays of the same length `present` and `future` where `present[i]` is the current price of the ith stock and `future[i]` is the price of the ith stock a year in the future. You may buy each stock at most **once**. You are also given an integer `budget` representing the amount of money you currently have.

Return *the maximum amount of profit you can make.*

Example 1:

```
Input: present = [5,4,6,2,3], future = [8,5,4,3,5], budget = 10
Output: 6
Explanation: One possible way to maximize your profit is to:
Buy the 0th, 3rd, and 4th stocks for a total of 5 + 2 + 3 = 10.
Next year, sell all three stocks for a total of 8 + 3 + 5 = 16.
The profit you made is 16 - 10 = 6.
It can be shown that the maximum profit you can make is 6.
```

Example 2:

```
Input: present = [2,2,5], future = [3,4,10], budget = 6
Output: 5
Explanation: The only possible way to maximize your profit is to:
Buy the 2nd stock, and make a profit of 10 - 5 = 5.
It can be shown that the maximum profit you can make is 5.
```

182 | 9 | 5 Online

Submit

Code

Python3 • Auto

```python
# this is a variation of knapsack problem
class Solution:
    def maximumProfit2(self, present: List[int], future: List[int], budget: int) -> int:
        n = len(present)
        def dfs(i, remaining):
            if i == n or remaining <= 0:
                return 0
            profit = future[i] - present[i]
            # Option 1: skip this stock
            res = dfs(i + 1, remaining)
            # Option 2: buy this stock (if profitable and affordable)
            if profit > 0 and present[i] <= remaining:
                res = max(res, profit + dfs(i + 1, remaining - present[i]))
            return res
        return dfs(0, budget)

    def maximumProfit(self, present: List[int], future: List[int], budget: int) -> int:
        n = len(present)
        dp = [0] * (budget + 1)
        for i in range(n):
            print(dp)
            profit = future[i] - present[i]
            if profit <= 0:
                continue  # skip stocks with no profit
            cost = present[i]
            for b in range(budget, cost - 1, -1):
                dp[b] = max(dp[b], dp[b - cost] + profit)
        print(dp)
        return max(dp)
```

Saved

Ln 1, Col 1

Testcase | Test Result

BlackRock

Description | Note | Editorial | Solutions | Submissions

399. Evaluate Division

Solved

Medium | Topics | Companies | Hint

You are given an array of variable pairs `equations` and an array of real numbers `values`, where `equations[i] = [$A_i$, $B_i$]` and `values[i]` represent the equation `$A_i$ / $B_i$ = values[i]`. Each `$A_i$` or `$B_i$` is a string that represents a single variable.

You are also given some `queries`, where `queries[j] = [$C_j$, $D_j$]` represents the `$j^{th}$` query where you must find the answer for `$C_j$ / $D_j$ = ?`.

Return *the answers to all queries.* If a single answer cannot be determined, return `-1.0`.

Note: The input is always valid. You may assume that evaluating the queries will not result in division by zero and that there is no contradiction.

Note: The variables that do not occur in the list of equations are undefined, so the answer cannot be determined for them.

Example 1:

```
Input: equations = [["a","b"],["b","c"]], values = [2.0,3.0],
queries = [["a","c"],["b","a"],["a","e"],["a","a"],["x","x"]]
Output: [6.00000,0.50000,-1.00000,1.00000,-1.00000]
Explanation:
Given: a / b = 2.0, b / c = 3.0
queries are: a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ?
return: [6.0, 0.5, -1.0, 1.0, -1.0 ]
note: x is undefined => -1.0
```

Example 2:

9.8K | 239 | 124 Online

Submit

Code

Python3 • Auto

```python
from collections import defaultdict
class Solution:
    def calcEquation(self, equations: List[List[str]], values: List[float], queries: List[List[str]]) -> List[float]:
        # Step 1: Build the graph
        graph = defaultdict(list)
        for i in range(len(equations)):
            # Add both directions: a/b = k and b/a = 1/k
            graph[equations[i][0]].append((values[i], equations[i][1]))
            graph[equations[i][1]].append((1/values[i], equations[i][0]))

        # Step 2: DFS to find the result for each query
        def dfs(start, end, seen):
            if start == end:
                return 1.0  # base case: division by itself
            seen.add(start)
            for val, node in graph[start]:
                if node not in seen:
                    res = dfs(node, end, seen)
                    if res != -1.0:
                        return val * res  # multiply the path value
            return -1.0  # no path found

        res = []
        # Step 3: Process each query
        for start, end in queries:
            if not (start in graph and end in graph):
                res.append(-1.0)  # variable not in graph
            else:
                seen = set()
                result = dfs(start, end, seen)
                res.append(result)
        return res
```

Saved

Ln 32, Col 19

Testcase | Test Result

BlackRock

Submit

Description | Note | Editorial | Solutions | Submissions

322. Coin Change

Solved

Medium | Topics | Companies

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] <= 2^31 - 1`

20K | 183 | 326 Online

Code

Python3 • Auto

```python
class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        # Use a priority queue to perform BFS: (steps taken, remaining amount)
        que = [(0, amount)]
        seen = set()  # To avoid revisiting the same amount

        while que:
            # Pop the state with the fewest steps so far
            step, node = hp.heappop(que)
            if node == 0:
                return step  # Found a solution
            if node < 0:
                continue  # Invalid state, skip
            if node in seen:
                continue  # Already visited this amount, skip
            seen.add(node)

            # Try all coin denominations
            for coin in coins:
                if node - coin not in seen:
                    hp.heappush(que, (step + 1, node - coin))
        return -1  # No solution found

# if __name__ == "__main__":
#     import sys
#     input = sys.stdin.read
#     data = input().splitlines()
#     # Input format:
#     # First line: space-separated coin denominations
#     # Second line: amount
#     coins = list(map(int, data[0].split()))
#     amount = int(data[1])
#     sol = Solution()
#     print(sol.coinChange(coins, amount))
```

Saved | Ln 29, Col 29

Testcase | Test Result

BlackRock

Submit

Description | Note | Editorial | Solutions | Submissions

1514. Path with Maximum Probability

Solved

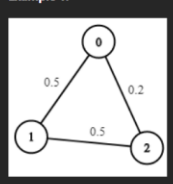
Medium | Topics | Companies | Hint

You are given an undirected weighted graph of `n` nodes (0-indexed), represented by an edge list where `edges[i] = [a, b]` is an undirected edge connecting the nodes `a` and `b` with a probability of success of traversing that edge `succProb[i]`.

Given two nodes `start` and `end`, find the path with the maximum probability of success to go from `start` to `end` and return its success probability.

If there is no path from `start` to `end`, **return 0**. Your answer will be accepted if it differs from the correct answer by at most **1e-5**.

Example 1:




```
Input: n = 3, edges = [[0,1],[1,2],[0,2]], succProb = [0.5,0.5,0.2], start = 0, end = 2
Output: 0.25000
Explanation: There are two paths from start to end, one having a probability of success = 0.2 and the other has 0.5 * 0.5 = 0.25.
```

Example 2:

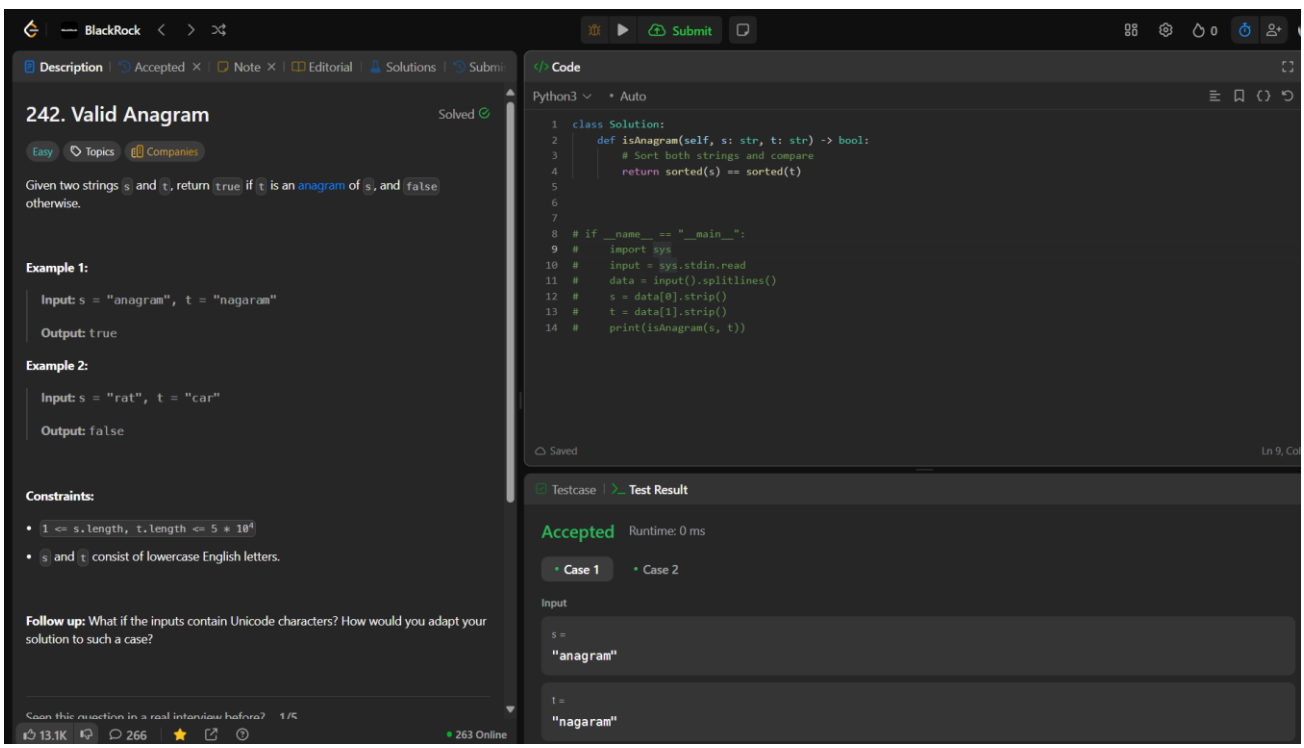
Code

Python3 • Auto

```python
import heapq as hp
from collections import defaultdict
class Solution:
    def maxProbability(self, n: int, edges: List[List[int]], succProb: List[float], start_node: int, end_node: int) -> fl
        # I will use BFS with priotity queue

        # build graph
        graph = defaultdict(list)
        for edge, score in zip(edges, succProb):
            graph[edge[0]].append((-score, edge[1]))
            graph[edge[1]].append((-score, edge[0]))

        # print(graph)
        # BFS with priotity queue
        que = [(-1.0,start_node)]
        seen = set()
        max_prob = 0

        while que:
            # print(que)
            score, node = hp.heappop(que)
            if node == end_node:
                return -score
            if node in seen:
                continue
            seen.add(node)
            for i in range(len(graph[node])):
                if graph[node][i][1] not in seen:
                    hp.heappush(que, (-(-score * -graph[node][i][0]), graph[node][i][1]))

        return 0.0
```

Saved | Ln 1, Col 1

```python
# if __name__ == "__main__":
#     import sys
#     input = sys.stdin.read
#     data = input().splitlines()
#     n = int(data[0])
#     m = int(data[1])
#     edges = [list(map(int, line.split())) for line in data[2:2 + m]]
#     succProb = list(map(float, data[2 + m].split()))
#     start_node = int(data[3 + m])
#     end_node = int(data[4 + m])
#     sol = Solution()
#     print(sol.maxProbability(n, edges, succProb, start_node, end_node))

# Inputs:
# 3
# 3
# 0 1
# 1 2
# 0 2
# 0.5 0.5 0.2
# 0
# 2
```



BlackRock

Description | Note | Editorial | Solutions | Submissions

202. Happy Number

Solved

Easy | Topics | Companies

Write an algorithm to determine if a number `n` is happy.

A **happy number** is a number defined by the following process:

- Starting with any positive integer, replace the number by the sum of the squares of its digits.
- Repeat the process until the number equals 1 (where it will stay), or it **loops endlessly in a cycle** which does not include 1.
- Those numbers for which this process **ends in 1** are happy.

Return `true` *if* `n` *is a happy number, and* `false` *if not.*

Example 1:

```
Input: n = 19
Output: true
Explanation:
1² + 9² = 82
8² + 2² = 68
6² + 8² = 100
1² + 0² + 0² = 1
```

Example 2:

```
Input: n = 2
Output: false
```

11.2K | 303 | 113 Online

Submit

Code

Python3 • Auto

```python
class Solution:
    def isHappy(self, n: int) -> bool:
        # since we might loop infinitely, Let's break the loop if we see an element twice
        sett = set()

        while n != 1:
            # if we already saw n
            if n in sett:
                return False

            sett.add(n)
            st = str(n)
            total = 0
            for digit in st:
                total += (int(digit))**2

            n = total


        return True


```

Saved | Ln 1, Col 1

Testcase | Test Result

You must run your code first

BlackRock

Description | Note | Editorial | Solutions | Submissions

121. Best Time to Buy and Sell Stock

Solved

Easy | Topics | Companies

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 <= 10^5`

Submit

Code

Python3 • Auto

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        profit = 0
        max_profit = 0
        min_p = prices[0]
        for i in range(len(prices)):
            if prices[i]>min_p:
                max_profit = max(max_profit, prices[i]-min_p)
            else:
                min_p = prices[i]

        return max_profit

```

Saved | Ln 1, Col 1

Testcase | Test Result

You must run your code first

BlackRock

Description | Accepted | Note | Editorial | Solutions | Submi

2592. Maximize Greatness of an Array

Solved

Medium | Topics | Companies | Hint

You are given a 0-indexed integer array `nums`. You are allowed to permute `nums` into a new array `perm` of your choosing.

We define the **greatness** of `nums` be the number of indices `0 <= i < nums.length` for which `perm[i] > nums[i]`.

Return *the **maximum** possible greatness you can achieve after permuting* `nums`.

Example 1:

```
Input: nums = [1,3,5,2,1,3,1]
Output: 4
Explanation: One of the optimal rearrangements is perm =
[2,5,1,3,3,1,1].
At indices = 0, 1, 3, and 4, perm[i] > nums[i]. Hence, we
return 4.
```

Example 2:

```
Input: nums = [1,2,3,4]
Output: 3
Explanation: We can prove the optimal perm is [2,3,4,1].
At indices = 0, 1, and 2, perm[i] > nums[i]. Hence, we return
3.
```

Constraints:

- `1 <= nums.length <= 10^5`

477 | 13 | 4 Online

Submit

Code

Python3 • Auto

```python
class Solution:
    def maximizeGreatness(self, nums: List[int]) -> int:
        # sorting perm and nums
        perm = nums[:]
        perm.sort()
        nums.sort()
        count = 0
        # setting pointers
        j = i = 0
        n = len(nums)
        # Method 1 : Brute Force O(N^2) + O(NlogN)
        # while i < n-1:
        #     j = i
        #     while j<n:
        #         if nums[j] > perm[i]:
        #             count+=1
        #             tmp = perm[i]
        #             perm[i] = nums[j]
        #             nums[j] = tmp
        #             break
        #         j+=1
        #     i+= 1

        # Method 2 : fast and slow pointer O(NlogN) + O(N)
        slow = i
        fast = j
        while slow<n and fast<n:
            if perm[fast] > nums[slow]:
                slow += 1
                count+=1
            fast += 1

        return count
```

Saved | Ln 15, Col 40

Testcase | Test Result

BlackRock

Description | Note | Editorial | Solutions | Submissions

49. Group Anagrams

Solved

Medium | Topics | Companies

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"]]`

20.6K | 303 | 365 Online

Submit

Code

Python3 • Auto

```python
from collections import defaultdict
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        dicto = defaultdict(list)

        for i in range(len(strs)):
            dicto["".join(sorted(list(strs[i])))].append(strs[i])

        res = []
        items = list(dicto.items())
        for i in range(len(items)):
            res.append(items[i][1])

        return res
```

Saved | Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

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

Output

[["eat","tea","ate"],["tan","nat"],["bat"]]

BlackRock

Submit

Description | Accepted | Note | Editorial | Solutions | Submi

545. Boundary of Binary Tree Premium

Solved

Medium | Topics | Companies

The **boundary** of a binary tree is the concatenation of the **root**, the **left boundary**, the **leaves** ordered from left-to-right, and the **reverse order** of the **right boundary**.

The **left boundary** is the set of nodes defined by the following:

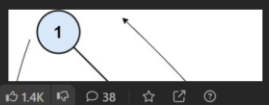
- The root node's left child is in the left boundary. If the root does not have a left child, then the left boundary is **empty**.
- If a node in the left boundary and has a left child, then the left child is in the left boundary.
- If a node is in the left boundary, has **no** left child, but has a right child, then the right child is in the left boundary.
- The leftmost leaf is **not** in the left boundary.

The **right boundary** is similar to the **left boundary**, except it is the right side of the root's right subtree. Again, the leaf is **not** part of the **right boundary**, and the **right boundary** is empty if the root does not have a right child.

The **leaves** are nodes that do not have any children. For this problem, the root is **not** a leaf.

Given the `root` of a binary tree, return *the values of its* ***boundary***.

Example 1:



1.4K 38 24 Online

Code

Python3 • Auto

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right


# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right

class Solution:
    def boundaryOfBinaryTree(self, root: Optional[TreeNode]) -> List[int]:
        if not root:
            return []


        # function
        def isleaf(node):
            return not node.left and not node.right

        # function
        def getLeftBoundary(node):
            while node:
                if not isleaf(node):
                    leftb.append(node.val)
                node = node.left if node.left else node.right

        # function
```

Saved Ln 1, Col 1

Testcase | Test Result

You must run your code first

BlackRock

Submit

Description | Accepted | Note | Editorial | Solutions | Submi

545. Boundary of Binary Tree Premium

Solved

Medium | Topics | Companies

The **boundary** of a binary tree is the concatenation of the **root**, the **left boundary**, the **leaves** ordered from left-to-right, and the **reverse order** of the **right boundary**.

The **left boundary** is the set of nodes defined by the following:

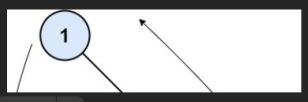
- The root node's left child is in the left boundary. If the root does not have a left child, then the left boundary is **empty**.
- If a node in the left boundary and has a left child, then the left child is in the left boundary.
- If a node is in the left boundary, has **no** left child, but has a right child, then the right child is in the left boundary.
- The leftmost leaf is **not** in the left boundary.

The **right boundary** is similar to the **left boundary**, except it is the right side of the root's right subtree. Again, the leaf is **not** part of the **right boundary**, and the **right boundary** is empty if the root does not have a right child.

The **leaves** are nodes that do not have any children. For this problem, the root is **not** a leaf.

Given the `root` of a binary tree, return *the values of its* ***boundary***.

Example 1:



1.4K 38 24 Online

Code

Python3 • Auto

```python
        # function
        def getLeftBoundary(node):
            while node:
                if not isleaf(node):
                    leftb.append(node.val)
                node = node.left if node.left else node.right

        # function
        # using DFS to keep the order of the leaves which BFS can not keep
        def getLeaves(node):
            if not node:
                return
            if isleaf(node):
                leaves.append(node.val)
            getLeaves(node.left)
            getLeaves(node.right)

        #function
        def getRightBoundary(node):
            tmp = []
            while node:
                if not isleaf(node):
                    tmp.append(node.val)
                node = node.right if node.right else node.left
            rightb.extend(tmp[::-1])


        # leftb, rightb, leaves
        leftb, rightb, leaves = [], [], []

        # check if root is leaf
        if isleaf(root):
            return [root.val]
```

Saved

Testcase | Test Result

You must run your code first

BlackRock

Submit

Description | Accepted | Note | Editorial | Solutions | Submis

545. Boundary of Binary Tree Premium

Solved

Medium | Topics | Companies

The **boundary** of a binary tree is the concatenation of the **root**, the **left boundary**, the **leaves** ordered from left-to-right, and the **reverse order** of the **right boundary**.

The **left boundary** is the set of nodes defined by the following:

- The root node's left child is in the left boundary. If the root does not have a left child, then the left boundary is **empty**.
- If a node in the left boundary and has a left child, then the left child is in the left boundary.
- If a node is in the left boundary, has **no** left child, but has a right child, then the right child is in the left boundary.
- The leftmost leaf is **not** in the left boundary.

The **right boundary** is similar to the **left boundary**, except it is the right side of the root's right subtree. Again, the leaf is **not** part of the **right boundary**, and the **right boundary** is empty if the root does not have a right child.

The **leaves** are nodes that do not have any children. For this problem, the root is **not** a leaf.

Given the `root` of a binary tree, return *the values of its* ***boundary***.

Example 1:

Code

Python3 • Auto

```python
                tmp.append(node.val)
            node = node.right if node.right else node.left
        rightb.extend(tmp[::-1])


    # leftb, rightb, leaves
    leftb, rightb, leaves = [], [], []

    # check if root is leaf
    if isleaf(root):
        return [root.val]

    # get the leftb
    leftb.append(root.val)
    if root.left:
        getLeftBoundary(root.left)

    #get the leaves
    getLeaves(root)

    #get the right boundary
    if root.right:
        getRightBoundary(root.right)

    # print(leftb)
    # print(leaves)
    # print(rightb)
    return leftb+leaves+rightb
```

Saved

Testcase | Test Result

You must run your code first

```python
if __name__ == "__main__":
    import sys
    import json
    input = sys.stdin.read
    data = input().splitlines()
    # Input format: a single line with a list, e.g. [1,null,2,3,4]
    arr = json.loads(data[0])

    # Helper to build the tree from level order input
    from collections import deque
    class TreeNode:
        def __init__(self, val=0, left=None, right=None):
            self.val = val
            self.left = left
            self.right = right

    def build_tree(arr):
        if not arr or arr[0] is None:
            return None
        root = TreeNode(arr[0])
        queue = deque([root])
        i = 1
        while queue and i < len(arr):
            node = queue.popleft()
            if i < len(arr) and arr[i] is not None:
                node.left = TreeNode(arr[i])
                queue.append(node.left)
            i += 1
            if i < len(arr) and arr[i] is not None:
                node.right = TreeNode(arr[i])
                queue.append(node.right)
            i += 1
        return root

    root = build_tree(arr)
    sol = Solution()
    print(sol.boundaryOfBinaryTree(root))
```

BlackRock - LeetCode | Maximum Profit From Trading St | Pairs of Songs With Total Duratio | www.google.com | ChatGPT

https://leetcode.com/problems/pairs-of-songs-with-total-durations-divisible-by-60/?envType=company&envId=blackrock&favoriteSlug=blackrock-all

BlackRock | Submit

Description | Note | Editorial | Solutions | Submissions

1010. Pairs of Songs With Total Durations Divisible by 60

Solved

Medium | Topics | Companies | Hint

You are given a list of songs where the ith song has a duration of `time[i]` seconds.

Return *the number of pairs of songs for which their total duration in seconds is divisible by* `60`. Formally, we want the number of indices `i`, `j` such that `i < j` with `(time[i] + time[j]) % 60 == 0`.

Example 1:

```
Input: time = [30,20,150,100,40]
Output: 3
Explanation: Three pairs have a total duration divisible by 60:
(time[0] = 30, time[2] = 150): total duration 180
(time[1] = 20, time[3] = 100): total duration 120
(time[1] = 20, time[4] = 40): total duration 60
```

Example 2:

```
Input: time = [60,60,60]
Output: 3
Explanation: All three pairs have a total duration of 120, which is divisible by 60.
```

4.3K | 30 | 15 Online

Code

Python3 • Auto

```python
from collections import defaultdict
class Solution:
    def numPairsDivisibleBy60(self, time: List[int]) -> int:
        # brute force O(n^2)
        # n = len(time)
        # count = 0
        # factor = 60

        # for i in range(n-1):
        #     for j in range(i+1,n):
        #         if (time[i] + time[j])%factor == 0:
        #             count+=1

        # return count

        # Using a hashtable O(N):
        n = len(time)
        count = 0
        target = 60
        table = defaultdict(int) # I could also use an array size 60 to replace the defaultdict

        for i in range(n):
            remainder = time[i] % target
            complement = (target-remainder)%target # in case complement is equal to target
            count+=table[complement]
            table[remainder] += 1

        return count
```

Saved | Ln 19, Col 20

Testcase | Test Result

BlackRock | Submit

Description | Note | Editorial | Solutions | Submissions

22. Generate Parentheses

Solved

Medium | Topics | Companies

Given `n` pairs of parentheses, write a function to *generate all combinations of well-formed parentheses.*

Example 1:

```
Input: n = 3
Output: ["((()))","(()())","(())()","()(())","()()()"]
```

Example 2:

```
Input: n = 1
Output: ["()"]
```

Constraints:

- `1 <= n <= 8`

Seen this question in a real interview before? 1/5

Yes | No

Accepted **2,408,386** /3.1M | Acceptance Rate **77.2** %

Topics

Code

Python3 • Auto

```python
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        #use recursion
        # list to results
        res = []
        # stack = ""
        stack = []

        #recursive funtion
        def helper(op, cl, st):   #  []
            print("stack : ", st)
            #base case:
            if op == cl == n:
                res.append("".join(st))
                return

            # add op bracket if number of open is less than n
            if op < n:
                st.append("(")        # ["("]
                new_st = st.copy()
                helper(op+1, cl, new_st)
                st.pop()

            # add cl if op > close
            if op>cl:
                st.append(")")
                new_st = st.copy()
                helper(op, cl+1,new_st)
                st.pop()

            return

        helper(0,0,stack)
        return res
```

Saved

BlackRock

Submit

Description | Note | Editorial | Solutions | Submissions

20. Valid Parentheses

Solved

Easy | Topics | Companies | Hint

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 = "([)"

Code

Python3 • Auto

```python
class Solution:
    def isValid(self, s: str) -> bool:
        stack = []
        for i in range(len(s)):
            if s[i] == ")":
                if stack and stack[-1] == "(":
                    stack.pop()
                else:
                    return False
            elif s[i] == "]":
                if stack and stack[-1] == "[":
                    stack.pop()
                else:
                    return False
            elif s[i] == "}":
                if stack and stack[-1] == "{":
                    stack.pop()
                else:
                    return False
            else:
                stack.append(s[i])

        if len(stack)==0:
            return True
        else:
            return False

'''
'''
```

Saved

BlackRock

Submit

Description | Accepted | Note | Editorial | Solutions | Submis

5. Longest Palindromic Substring

Solved

Medium | Topics | Companies | Hint

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.

Seen this question in a real interview before? 1/5

Yes | No

Accepted 3,925,574/10.9M | Acceptance Rate 35.9%

Code

Python3 • Auto

```python
class Solution:
    def longestPalindrome(self, s: str) -> str:
        #Approach Two
        self.res = ""

        #Function
        def findPalindrome(l, r):
            while l>-1 and r<len(s):
                if s[l] == s[r]:
                    if len(self.res) < (r-l+1):
                        self.res = s[l:r+1]
                    l-=1
                    r+=1
                else:
                    break
            return self.res

        # for odd
        # O(N*2)
        for i in range(len(s)):
            findPalindrome(i, i)
        # for even
        # O(N*2)
        for i in range(1,len(s)):
            findPalindrome(i-1,i)

        return self.res




'''
ex1
s = babad
```

BlackRock

Submit

Description | Note | Editorial | Solutions | Submissions

200. Number of Islands

Solved

Medium | Topics | Companies

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
```

Code

Python3 • Auto

```python
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        total = 0
        def helper(x, y):
            if x< 0 or y< 0 or x>=len(grid) or y>=len(grid[0]) or grid[x][y]== "0":
                return

            grid[x][y] ="0"
            helper(x+1,y)
            helper(x, y+1)
            helper(x, y-1)
            helper(x-1, y)

            return

        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == "1":
                    total += 1
                    helper(i,j)
        print(grid)
        return total
```