

1.Maximum XOR of Two Non-Overlapping Subtrees

There is an undirected tree with n nodes labeled from 0 to $n - 1$. You are given the integer n and a 2D integer array `edges` of length $n - 1$, where `edges[i] = [ai, bi]` indicates that there is an edge between nodes a_i and b_i in the tree. The root of the tree is the node labeled 0. Each node has an associated value. You are given an array `values` of length n , where `values[i]` is the value of the i th node. Select any two non-overlapping subtrees. Your score is the bitwise XOR of the sum of the values within those subtrees. Return the maximum possible score you can achieve. If it is impossible to find two nonoverlapping subtrees, return 0. Note that:

- The subtree of a node is the tree consisting of that node and all of its descendants.
- Two subtrees are non-overlapping if they do not share any common node.

Example 1:

Input: $n = 6$, `edges = [[0,1],[0,2],[1,3],[1,4],[2,5]]`, `values = [2,8,3,6,2,5]`

Output: 24

Explanation: Node 1's subtree has sum of values 16, while node 2's subtree has sum of values 8, so choosing these nodes will yield a score of $16 \text{ XOR } 8 = 24$. It can be proved that is the maximum possible

code:

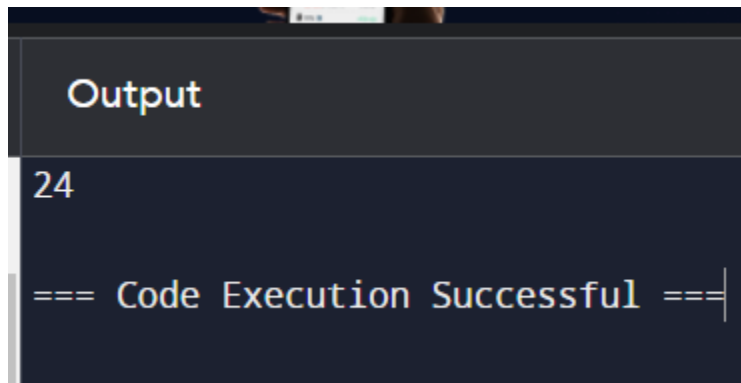
```
from collections import defaultdict
from typing import List
class Trie:
    def __init__(self):
        self.children = [None] * 2
    def insert(self, x):
        node = self
        for i in range(47, -1, -1):
            v = (x >> i) & 1
            if node.children[v] is None:
                node.children[v] = Trie()
            node = node.children[v]
    def search(self, x):
        node = self
        res = 0
        for i in range(47, -1, -1):
```

```

    v = (x >> i) & 1
    if node is None:
        return res
    if node.children[v ^ 1]:
        res = res << 1 | 1
        node = node.children[v ^ 1]
    else:
        res <<= 1
        node = node.children[v]
    return res
class Solution:
    def maxXor(self, n: int, edges: List[List[int]], values: List[int]) -> int:
        def dfs1(i, fa):
            t = values[i]
            for j in g[i]:
                if j != fa:
                    t += dfs1(j, i)
            s[i] = t
            return t
        def dfs2(i, fa):
            nonlocal ans
            ans = max(ans, tree.search(s[i]))
            for j in g[i]:
                if j != fa:
                    dfs2(j, i)
            tree.insert(s[i])
        g = defaultdict(list)
        for a, b in edges:
            g[a].append(b)
            g[b].append(a)
        s = [0] * n
        dfs1(0, -1)
        ans = 0
        tree = Trie()
        dfs2(0, -1)
        return ans
n = 5
edges = [[0, 1], [1, 2], [1, 3], [3, 4]]
values = [1, 2, 3, 4, 5]
solution = Solution()
result = solution.maxXor(n, edges, values)
print(result)

```

output:



```
Output
24
=== Code Execution Successful ===
```

2. Form a Chemical Bond SQL Schema \

Table: Elements +-----+-----+ | Column Name | Type |
+-----+-----+ | symbol | varchar | | type | enum | | electrons | int |
+-----+-----+ symbol is the primary key for this table. Each row of this table contains information of one element. type is an ENUM of type ('Metal', 'Nonmetal', 'Noble') - If type is Noble, electrons is 0. - If type is Metal, electrons is the number of electrons that one atom of this element can give. - If type is Nonmetal, electrons is the number of electrons that one atom of this element needs. Two elements can form a bond if one of them is 'Metal' and the other is 'Nonmetal'. Write an SQL query to find all the pairs of elements that can form a bond. Return the result table in any order. The query result format is in the following example.

Example 1:

Input: Elements table: +-----+-----+-----+ | symbol | type |
electrons | +-----+-----+-----+ | He | Noble | 0 | | Na | Metal | 1 | |
Ca | Metal | 2 | | La | Metal | 3 | | Cl | Nonmetal | 1 | | O | Nonmetal | 2 | |
N | Nonmetal | 3 | +-----+-----+-----+

Output: +-----+-----+ | metal | nonmetal | +-----+-----+ | La | Cl |
| Ca | Cl | | Na | Cl | | La | O | | Ca | O | | Na | O | | La | N | | Ca | N | | Na |
N | +-----+-----+

Explanation: Metal elements are La, Ca, and Na. Nonmeal elements are Cl, O, and N. Each Metal element pairs with a Nonmetal element in the output table. Accepted:173 Su

code:

```
import sqlite3
conn = sqlite3.connect(':memory:')
cursor = conn.cursor()
cursor.execute("""
    CREATE TABLE Elements (
        symbol TEXT PRIMARY KEY,
        type TEXT CHECK(type IN ('Metal', 'Nonmetal', 'Noble')),
        electrons INT
    )
""")
elements_data = [
    ('He', 'Noble', 0),
    ('Na', 'Metal', 1),
    ('Ca', 'Metal', 2),
    ('La', 'Metal', 3),
    ('Cl', 'Nonmetal', 1),
    ('O', 'Nonmetal', 2),
    ('N', 'Nonmetal', 3)
]
cursor.executemany("""
    INSERT INTO Elements (symbol, type, electrons)
    VALUES (?, ?, ?)
""", elements_data)
query = """
    SELECT e1.symbol AS metal, e2.symbol AS nonmetal
    FROM Elements e1, Elements e2
    WHERE e1.type = 'Metal' AND e2.type = 'Nonmetal'
"""
cursor.execute(query)
results = cursor.fetchall()
print("+-----+-----+")
print("| metal | nonmetal |")
print("+-----+-----+")
for row in results:
    print(f"| {row[0]:<5} | {row[1]:<8} |")
print("+-----+-----+")
conn.close()
```

```
Output
+-----+-----+
| metal | nonmetal |
+-----+-----+
| Na    | Cl      |
| Na    | N       |
| Na    | O       |
| Ca    | Cl      |
| Ca    | N       |
| Ca    | O       |
| La    | Cl      |
| La    | N       |
| La    | O       |
+-----+-----+

=== Code Execution Successful ===
```

3. Minimum Cuts to Divide a Circle

A valid cut in a circle can be: A cut that is represented by a straight line that touches two points on the edge of the circle and passes through its center, or A cut that is represented by a straight line that touches one point on the edge of the circle and its center. Some valid and invalid cuts are shown in the figures below. Given the integer n , return the minimum number of cuts needed to divide a circle into n equal slices.

Example 1:

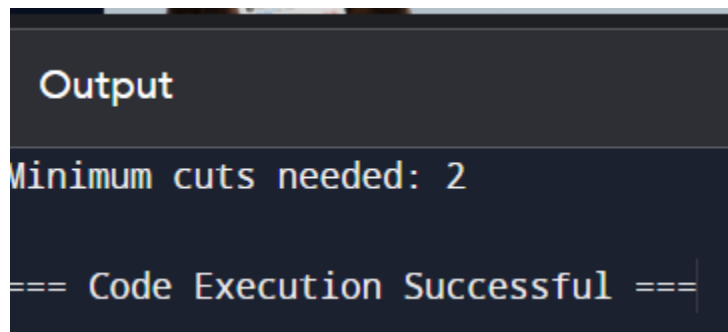
Input: $n = 4$

Output: 2

Explanation: The above figure shows how cutting the circle twice through the middle divides it into 4 equal slices.

Code:

```
def minimumCuts(n):  
    if n == 1:  
        return 0  
    elif n % 2 == 0:  
        return n // 2  
    else:  
        return (n // 2) + 1  
n = 4  
print("Minimum cuts needed:", minimumCuts(n))
```



The screenshot shows a dark-themed output window. At the top, the word "Output" is displayed in a light blue font. Below it, the text "Minimum cuts needed: 2" is shown in a light blue monospace font. At the bottom, the text "=== Code Execution Successful ===" is displayed in a light blue monospace font.

4. Difference Between Ones and Zeros in Row and Column

You are given the customer visit log of a shop represented by a 0-indexed string `customers` consisting only of characters 'N' and 'Y':

- if the i th character is 'Y', it means that customers come at the i th hour
- whereas 'N' indicates that no customers come at the i th hour. If the shop closes at the j th hour ($0 \leq j \leq n$), the penalty is calculated as follows:
- For every hour when the shop is open and no customers come, the penalty increases by 1.
- For every hour when the shop is closed and customers come, the penalty increases by 1.

Return the earliest hour at which the shop must be closed to incur a minimum penalty. Note that if a shop closes at the j th hour, it means the shop is closed at the hour j .

Example 1:

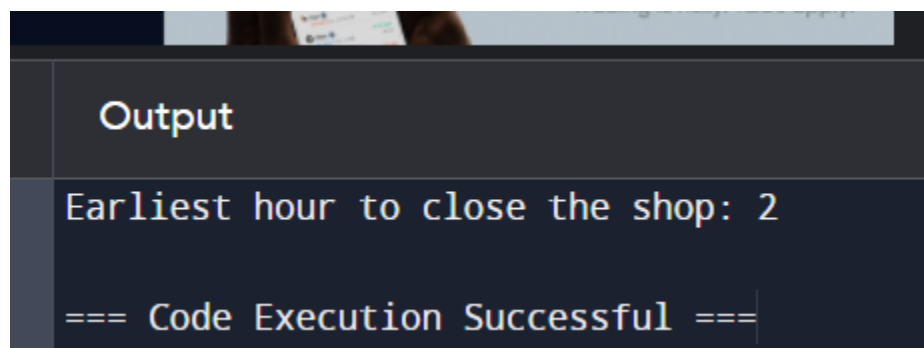
Input: `customers = "YNYN"`

Output: 2

Explanation: - Closing the shop at the 0th hour incurs in $1+1+0+1 = 3$ penalty. - Closing the shop at the 1st hour incurs in $0+1+0+1 = 2$ penalty. - Closing the shop at the 2nd hour incurs in $0+0+0+1 = 1$ penalty. - Closing the shop at the 3rd hour incurs in $0+0+1+1 = 2$ penalty. - Closing the shop at the 4th hour incurs in $0+0+1+0 = 1$ penalty. Closing the shop at 2nd or 4th hour gives a minimum penalty. Since 2 is earlier, the optimal closing time is 2.

code:

```
def minPenaltyClosingHour(customers: str) -> int:
    n = len(customers)
    min_penalty = float('inf')
    best_hour = 0
    y_count = [0] * (n + 1)
    n_count = [0] * (n + 1)
    for i in range(n):
        y_count[i + 1] = y_count[i] + (customers[i] == 'Y')
        n_count[i + 1] = n_count[i] + (customers[i] == 'N')
    for j in range(n + 1):
        penalty = n_count[j] + (y_count[n] - y_count[j])
        if penalty < min_penalty:
            min_penalty = penalty
            best_hour = j
    return best_hour
customers = "YYNY"
print("Earliest hour to close the shop:", minPenaltyClosingHour(customers))
```

A screenshot of a code execution environment. At the top, there's a header labeled "Output". Below it, the text "Earliest hour to close the shop: 2" is displayed. At the bottom, a status message reads "=== Code Execution Successful ===".

```
Output
Earliest hour to close the shop: 2
=== Code Execution Successful ===
```

5. Minimum Penalty for a Shop

You are given the customer visit log of a shop represented by a 0-indexed string customers consisting only of characters 'N' and 'Y': •

if the i th character is 'Y', it means that customers come at the i th hour

- whereas 'N' indicates that no customers come at the i th hour. If the shop closes at the j th hour ($0 \leq j \leq n$), the penalty is calculated as follows:
- For every hour when the shop is open and no customers come, the penalty increases by 1.
- For every hour when the shop is closed and customers come, the penalty increases by 1.

Return the earliest hour at which the shop must be closed to incur a minimum penalty. Note that if a shop closes at the j th hour, it means the shop is closed at the hour j .

Example 1:

Input: customers = "YYNY"

Output: 2

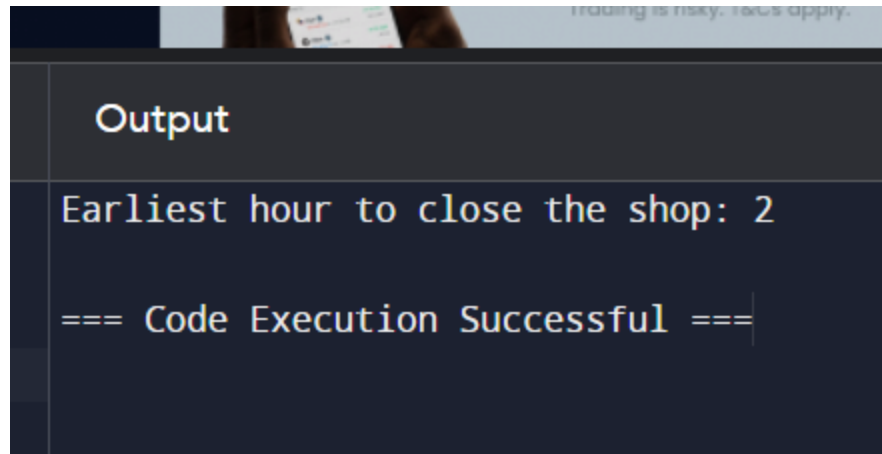
Explanation:

- Closing the shop at the 0th hour incurs in $1+1+0+1 = 3$ penalty.
- Closing the shop at the 1st hour incurs in $0+1+0+1 = 2$ penalty.
- Closing the shop at the 2nd hour incurs in $0+0+0+1 = 1$ penalty.
- Closing the shop at the 3rd hour incurs in $0+0+1+1 = 2$ penalty.
- Closing the shop at the 4th hour incurs in $0+0+1+0 = 1$ penalty.

Closing the shop at 2nd or 4th hour gives a minimum penalty. Since 2 is earlier, the optimal closing time is 2.

code:

```
def minPenaltyClosingHour(customers: str) -> int:
    n = len(customers)
    min_penalty = float('inf')
    best_hour = 0
    y_count = [0] * (n + 1)
    n_count = [0] * (n + 1)
    for i in range(n):
        y_count[i + 1] = y_count[i] + (customers[i] == 'Y')
        n_count[i + 1] = n_count[i] + (customers[i] == 'N')
    for j in range(n + 1):
        penalty = n_count[j] + (y_count[n] - y_count[j])
        if penalty < min_penalty:
            min_penalty = penalty
            best_hour = j
    return best_hour
customers = "YYNY"
print("Earliest hour to close the shop:", minPenaltyClosingHour(customers))
```

```
Output
Earliest hour to close the shop: 2
=== Code Execution Successful ===
```

6. Count Palindromic Subsequences

Given a string of digits s , return the number of palindromic subsequences of s having length 5. Since the answer may be very large, return it modulo $10^9 + 7$. Note: • A string is palindromic if it reads the same forward and backward. • A subsequence is a string that can be derived from another string by deleting some or no characters without changing the order of the remaining characters.

Example 1:

Input: $s = "103301"$

Output: 2

Code:

```
MOD = 10**9 + 7
def count_palindromic_subsequences(s):
    n = len(s)
    dp = [[0] * n for _ in range(n)]
    for i in range(n):
        dp[i][i] = 1
    for length in range(2, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if s[i] == s[j]:
                dp[i][j] = dp[i + 1][j - 1] + 2
            else:
                dp[i][j] = dp[i + 1][j] + dp[i][j - 1] - dp[i + 1][j - 1]
    return dp[0][n - 1] % MOD
s = "103301"
print(count_palindromic_subsequences(s))
```

```
Output
6
=== Code Execution Successful ===
```

7. Find the Pivot Integer

Given a positive integer n , find the pivot integer x such that: • The sum of all elements between 1 and x inclusively equals the sum of all elements between x and n inclusively. Return the pivot integer x . If no such integer exists, return -1. It is guaranteed that there will be at most one pivot index for the given input.

Example 1:

Input: $n = 8$

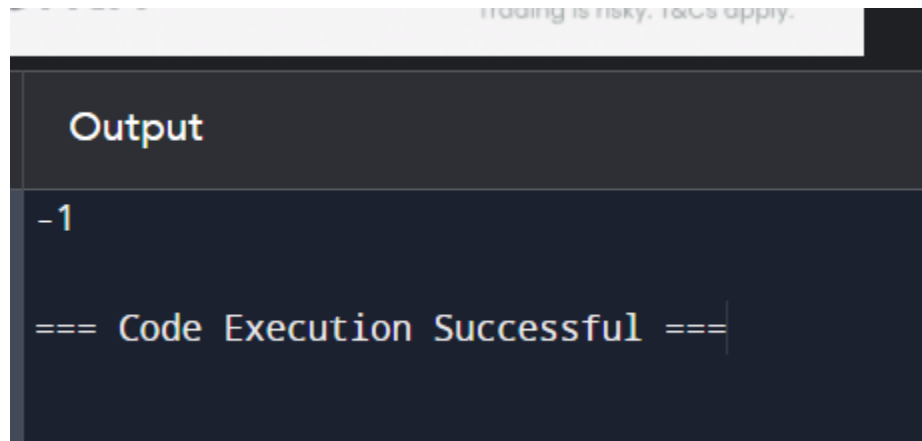
Output: 6

Explanation: 6 is the pivot integer since: $1 + 2 + 3 + 4 + 5 + 6 = 6 +$

code:

```
def find_pivot_integer(n):
    total_sum = n * (n + 1) // 2
    if total_sum % 2 != 0:
        return -1
    target_sum = total_sum // 2
    current_sum = 0
    for x in range(1, n + 1):
        current_sum += x
        if current_sum == target_sum:
            return x
        elif current_sum > target_sum:
            return -1
    return -1
n = 4
```

```
print(find_pivot_integer(n))
```



The screenshot shows a dark-themed interface with a header bar at the top. Below the header, the word "Output" is displayed in a light blue font. Underneath, the value "-1" is shown in a light blue font. At the bottom, a message "=== Code Execution Successful ===" is displayed in a light blue font, with a vertical cursor line positioned at the end of the text.

8. Append Characters to String to Make Subsequence

You are given two strings *s* and *t* consisting of only lowercase English letters. Return the minimum number of characters that need to be appended to the end of *s* so that *t* becomes a subsequence of *s*. A subsequence is a string that can be derived from another string by deleting some or no characters without changing the order of the remaining characters.

Example 1:

Input: *s* = "coaching", *t* = "coding"

Output: 4

Explanation: Append the characters "ding" to the end of *s* so that *s* = "coachingding". Now, *t* is a subsequence of *s* ("coachingding"). It can be shown that appending any

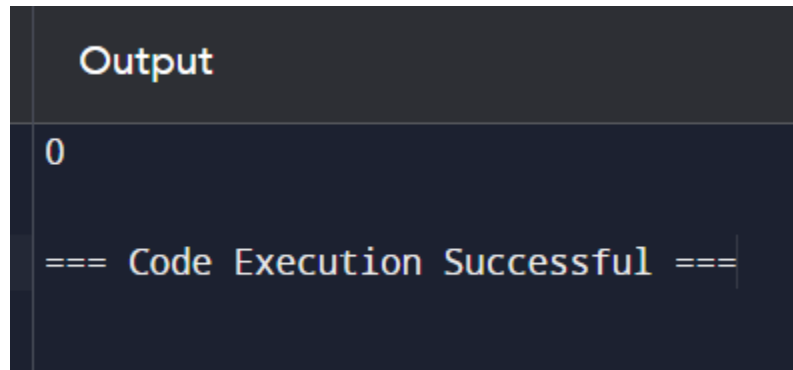
Code:

```
def min_appends_to_make_subsequence(s, t):
    s_ptr, t_ptr = 0, 0
    appends = 0
    while t_ptr < len(t):
        if s_ptr == len(s):
            appends += len(t) - t_ptr
            break
        if s[s_ptr] == t[t_ptr]:
            t_ptr += 1
        else:
            appends += 1
```

```

        s_ptr += 1
    return appends
s = "abcde"
t = "a"
print(min_appends_to_make_subsequence(s, t))

```



```

Output

0

=== Code Execution Successful ===

```

9. Remove Nodes From Linked List

You are given the head of a linked list. Remove every node which has a node with a strictly greater value anywhere to the right side of it. Return the head of the modified linked list.

Example 1:

Input: head = [5,2,13,3,8]

Output: [13,8]

Explanation: The nodes that should be removed are 5, 2 and 3. - Node 13 is to the right of node 5. - Node 13 is to the right of node 2. - Node 8 is to the right of node 3.

code:

```

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def remove_greater_nodes(head):
    if not head:
        return None
    dummy = ListNode(0)
    dummy.next = head
    prev = dummy
    current = head
    while current and current.next:

```

```

        if current.val < current.next.val:
            prev.next = current.next
            current = prev.next
        else:
            prev = prev.next
            current = current.next
    return dummy.next
def print_linked_list(head):
    while head:
        print(head.val, end=" ")
        head = head.next
    print()
head = ListNode(5)
head.next = ListNode(2)
head.next.next = ListNode(13)
head.next.next.next = ListNode(3)
head.next.next.next.next = ListNode(8)
print("Original linked list:")
print_linked_list(head)
new_head = remove_greater_nodes(head)
print("Modified linked list:")
print_linked_list(new_head)

```

Output

Original linked list:

5 2 13 3 8

Modified linked list:

5 13 8

=== Code Execution Successful ===

10. Count Subarrays With Median K

You are given an array `nums` of size `n` consisting of distinct integers from 1 to `n` and a positive integer `k`. Return the number of non-empty subarrays in `nums` that have a median equal to `k`. Note: • The median of an array is the middle element after sorting the array in ascending order. If the array is of even length, the median is the left middle element. ○

For example, the median of `[2,3,1,4]` is 2, and the median of `[8,4,3,5,1]` is 4. • A subarray is a contiguous part of an array.

Example 1:

Input: `nums = [3,2,1,4,5]`, `k = 4`

Output: 3

Explanation: The subarrays that have a median equal to 4 are: `[4]`, `[4,5]` and `[1,4,5]`

code:

```
def count_subarrays_with_median(nums, k):
    n = len(nums)
    count = 0
    for i in range(n):
        for j in range(i, n):
            subarray = nums[i:j+1]
            subarray.sort()
            median_index = (j - i) // 2
            if subarray[median_index] == k:
                count += 1
    return count
nums = [3, 2, 1, 4, 5]
k = 4
print(count_subarrays_with_median(nums, k))
```

Output

3

=== Code Execution Successful ===