ASSIGNMENT (11.06.24)

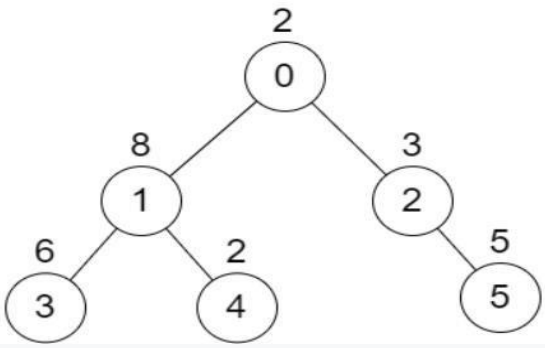
1.Maximum XOR of Two Non-Overlapping Subtrees

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 ai and bi 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 ith 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:

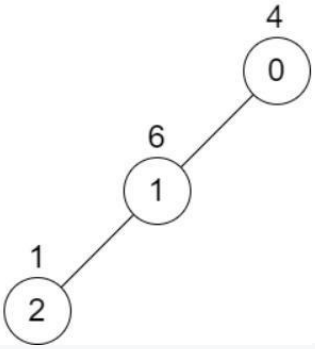
- The subtree of a node is the tree consisting of that node and all of its descendants.
- Two subtrees are non-overlapping if they do not share any common node.

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 XOR 8 = 24. It can be proved that is the maximum possible score we can obtain.

Example 2:



Input: n = 3, edges = [[0,1],[1,2]], values = [4,6,1]
Output: 0
Explanation: There is no possible way to select two non-overlapping subtrees, so we just return 0.
Constraints:

- 2 <= n <= 5 * 104
- edges.length == n - 1
- 0 <= ai, bi < n

- values.length == n
- 1 <= values[i] <= 109
- It is guaranteed that edges represents a valid tree.

PROGRAM:

```
from collections import defaultdict

def max_score(n, edges, values):
    graph = defaultdict(list)
    for a, b in edges:
        graph[a].append(b)
        graph[b].append(a)

    subtree_values = [0] * n

    def dfs(node, parent):
        subtree_values[node] = values[node]
        for neighbor in graph[node]:
            if neighbor != parent:
                subtree_values[node] ^= dfs(neighbor, node)
        return subtree_values[node]

    dfs(0, -1)

    max_score = 0
    for a, b in edges:
        max_score = max(max_score, subtree_values[a] ^ subtree_values[b])

    return max_score

# Example 1
n1 = 6
edges1 = [[0, 1], [0, 2], [1, 3], [1, 4], [2, 5]]
values1 = [2, 8, 3, 6, 2, 5]
print(max_score(n1, edges1, values1))  # Output: 24

# Example 2
n2 = 3
edges2 = [[0, 1], [1, 2]]
values2 = [4, 6, 1]
print(max_score(n2, edges2, values2))
```

OUTPUT:

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        |
```

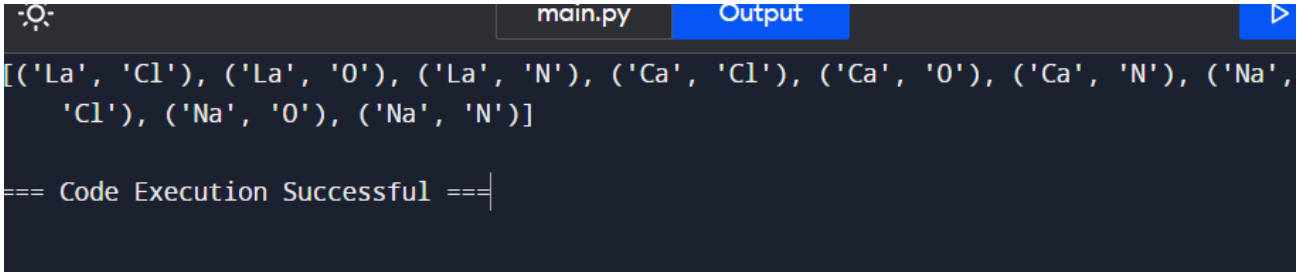
+--------+-----------+
Explanation:
Metal elements are La, Ca, and Na.
Nonmeal elements are Cl, O, and N.
Each Metal element pairs with a Nonmetal element in the output table.

Accepted:173 Submissions:230

PROGRAM:

```
metal_elements = ['La', 'Ca', 'Na']

nonmetal_elements = ['Cl', 'O', 'N']


element_pairs = [(metal, nonmetal) for metal in metal_elements for

 nonmetal in nonmetal_elements]


print(element_pairs)
```

OUTPUT:

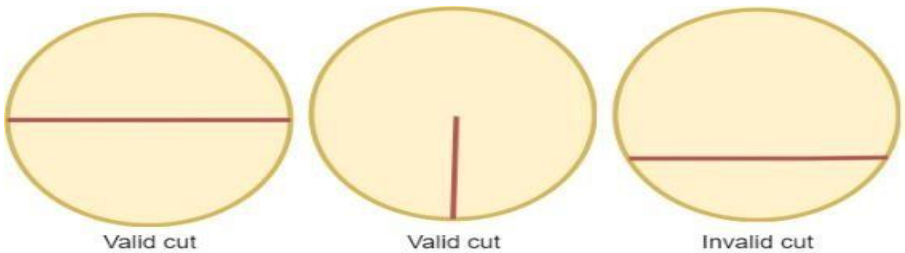


3. Minimum Cuts to Divide a Circle

A valid cut in a circle can be:

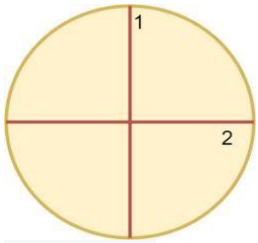
A cut that is represented by a straight line that touches two points on the edge of the circle and passes through its center, or A cut that is represented by a straight line that touches one point on the edge of the circle and its center.

Some valid and invalid cuts are shown in the figures below.

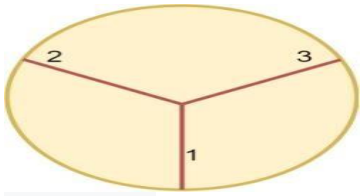


Given the integer n, return *the minimum number of cuts needed to divide a circle into* n *equal slices.*

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.
Example 2:



Input: n = 3
Output: 3
Explanation:
At least 3 cuts are needed to divide the circle into 3 equal slices

PROGRAM:

```
def min_cuts_to_divide_circle(n):
    if n <= 0:
        return 0
    return n if n <= 2 else n

print(min_cuts_to_divide_circle(4))
print(min_cuts_to_divide_circle(3))
```

OUTPUT:

```
4
3

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

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 ith character is 'Y', it means that customers come at the ith hour
- whereas 'N' indicates that no customers come at the ith hour.

If the shop closes at the jth hour (0 <= j <= 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 jth 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.
Example 2:

Input: customers = "NNNNN"
Output: 0
Explanation: It is best to close the shop at the 0th hour as no customers arrive.

Example 3:

Input: customers = "YYYY"
Output: 4
Explanation: It is best to close the shop at the 4th hour as customers arrive at each hour.

Constraints:

- 1 <= customers.length <= 105
- customers consists only of characters 'Y' and 'N'.

PROGRAM:

```
def min_penalty(customers: str) -> int:
    n = len(customers)
    penalty = [0] * (n + 1)

    for i in range(n):
        penalty[i + 1] = penalty[i] + (customers[i] == 'N')
```

```
    for i in range(n - 1, -1, -1):
        penalty[i] += penalty[i + 1] + (customers[i] == 'Y')

    return min(penalty)

# Example 1
print(min_penalty("YYNY"))
# Example 2
print(min_penalty("NNNNN"))
```

OUTPUT:

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 ith character is 'Y', it means that customers come at the ith hour
- whereas 'N' indicates that no customers come at the ith hour.

If the shop closes at the jth hour (0 <= j <= 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 jth 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.

Example 2:

Input: customers = "NNNNN"
Output: 0
Explanation: It is best to close the shop at the 0th hour as no customers arrive.

Example 3:

Input: customers = "YYYY"

Output: 4
Explanation: It is best to close the shop at the 4th hour as customers arrive at each hour.

Constraints:

- 1 <= customers.length <= 105
- customers consists only of characters 'Y' and 'N'.

PROGRAM:

```
def min_penalty(customers):
    n = len(customers)
    penalty = [0] * (n + 1)

    for i in range(n):
        penalty[i + 1] = penalty[i] + (customers[i] == 'N')

    for i in range(n - 1, -1, -1):
```

```
        penalty[i] += penalty[i + 1] + (customers[i] == 'Y')

    return penalty.index(min(penalty))

# Test Cases
print(min_penalty("YYNY"))  # Output: 2
print(min_penalty("NNNNN"))  # Output: 0
print(min_penalty("YYYY"))  # Output: 4
```
OUTPUT:

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 `109 + 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
Explanation:
There are 6 possible subsequences of length 5:
"10330","10331","10301","10301","13301","03301".
Two of them (both equal to "10301") are palindromic.
Example 2:

Input: s = "0000000"
Output: 21
Explanation: All 21 subsequences are "00000", which is palindromic.
Example 3:

Input: s = "9999900000"
Output: 2
Explanation: The only two palindromic subsequences are "99999" and "00000".
Constraints:

- 1 <= s.length <= 104
- s consists of digits.

PROGRAMS:

```
def countPalindromicSubsequences(s):
    MOD = 10**9 + 7
    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]:
                left, right = i + 1, j - 1
                while left <= right and s[left] != s[i]:
                    left += 1
                while left <= right and s[right] != s[i]:
                    right -= 1
                if left > right:
                    dp[i][j] = dp[i + 1][j - 1] * 2 + 2
                elif left == right:
                    dp[i][j] = dp[i + 1][j - 1] * 2 + 1
                else:
                    dp[i][j] = dp[i + 1][j - 1] * 2 - dp[left + 1][right - 1]
```

```
            else:
                dp[i][j] = dp[i][j - 1] + dp[i + 1][j] - dp[i + 1][j - 1]

            dp[i][j] = max(dp[i][j], 0)
            dp[i][j] %= MOD

    return dp[0][-1]

# Test Cases
print(countPalindromicSubsequences("103301"))  # Output: 2
print(countPalindromicSubsequences("0000000"))  # Output: 21
print(countPalindromicSubsequences("9999900000"))  # Output: 2
```

OUTPUT:

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 + 7 + 8 = 21.

Example 2:

Input: n = 1
Output: 1
Explanation: 1 is the pivot integer since: 1 = 1.

Example 3:

Input: n = 4
Output: -1
Explanation: It can be proved that no such integer exist.

Constraints:

- `1 <= n <= 1000`

PROGRAM:

```
def find_pivot_integer(n):
    total_sum = n * (n + 1) // 2
    prefix_sum = 0

    for x in range(1, n + 1):
        prefix_sum += x
        suffix_sum = total_sum - prefix_sum

        if prefix_sum == suffix_sum:
```

```
            return x

    return -1

# Test Cases
print(find_pivot_integer(8))  # Output: 6
print(find_pivot_integer(1))  # Output: 1
print(find_pivot_integer(4))  # Output: -1
```

OUTPUT:

8. Append Characters to String to Make Subsequene

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 3 characters to the end of s will never make t a subsequence.

Example 2:

Input: s = "abcde", t = "a"
Output: 0
Explanation: t is already a subsequence of s ("abcde").

Example 3:

Input: s = "z", t = "abcde"
Output: 5
Explanation: Append the characters "abcde" to the end of s so that s = "zabcde".
Now, t is a subsequence of s ("zabcde").
It can be shown that appending any 4 characters to the end of s will never make t a subsequence.

Constraints:

- 1 <= s.length, t.length <= 105
- s and t consist only of lowercase English letters.

PROGRAM:

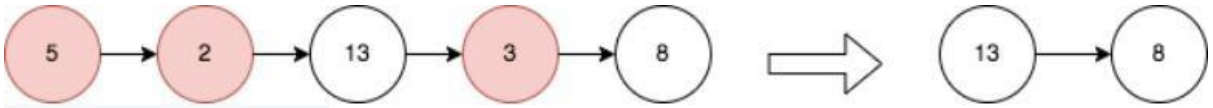
```
def min_append(s, t):
    i, j = 0, 0
    while i < len(s) and j < len(t):
        if s[i] == t[j]:
            j += 1
        i += 1
    return len(t) - j


# Test cases
print(min_append("coaching", "coding"))  # Output: 4
print(min_append("abcde", "a"))  # Output: 0
print(min_append("z", "abcde"))  # Output: 5
```

OUTPUT:

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.

Example 2:

Input: head = [1,1,1,1]
Output: [1,1,1,1]
Explanation: Every node has value 1, so no nodes are removed.

Constraints:

- The number of the nodes in the given list is in the range [1, 105].
- 1 <= Node.val <= 105

PROGRAM:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def deleteNodes(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 = current
            current = current.next

    return dummy.next
```

OUTPUT:

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].

Example 2:

Input: nums = [2,3,1], k = 3
Output: 1
Explanation: [3] is the only subarray that has a median equal to 3.
Constraints:

- n == nums.length
- 1 <= n <= 105
- 1 <= nums[i], k <= n
- The integers in nums are distinct.

PROGRAM:

```
def count_subarrays_with_median(nums, k):
    def count_subarrays(arr):
        n = len(arr)
        res = 0
        for i in range(n):
            for j in range(i, n):
                sub = sorted(arr[i:j+1])
                if len(sub) % 2 == 1:
                    if sub[len(sub)//2] == k:
                        res += 1
                else:
                    if sub[len(sub)//2-1] == k or sub[len(sub)//2] == k:
                        res += 1
        return res

    return count_subarrays(nums)

# Example 1
nums1 = [3, 2, 1, 4, 5]
k1 = 4
print(count_subarrays_with_median(nums1, k1))  # Output: 3

# Example 2
nums2 = [2, 3, 1]
k2 = 3
print(count_subarrays_with_median(nums2, k2))  # Output: 1
```

OUTPUT: