LAB DAY-6(11-06-2024)

1.MaximumXORofTwoNon-OverlappingSubtrees

Thereisanundirectedtreewithnnodeslabeledfrom0ton-1.Youaregiventheintegernanda 2Dintegerarrayedgesoflengthn-1,whereedges[i]=[ai,bi]indicatesthatthereisanedge betweennodesaiandbiinthetree.Therootofthetreeisthenodelabeled0.Eachnodehasan associatedvalue.Youaregivenanarrayvaluesoflengthn,wherevalues[i]isthevalueoftheith node.Selectanytwonon-overlappingsubtrees.YourscoreisthebitwiseXORofthesumofthe valueswithinthosesubtrees.Returnthemaximumpossiblescoreyoucanachieve.Ifitis impossibletofindtwononoverlappingsubtrees,return0.

Notethat:

- Thesubtreeofanodeisthetreeconsistingofthatnodeandallofitsdescendants.
- Twosubtreesarenon-overlappingiftheydonotshareanycommonnode.

Example1:

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

Output:24

Explanation:Node1'ssubtreehassumofvalues16,whilenode2'ssubtreehassumofvalues8, sochoosingthesenodeswillyieldascoreof16XOR8=24.Itcanbeprovedthatisthe maximumpossiblescorewecanobtain.

Example2:

Input:n=3,edges=[[0,1],[1,2]],values=[4,6,1]

Output:0

Explanation:Thereisnopossiblewaytoselecttwonon-overlappingsubtrees,sowejustreturn0.

Constraints:

- 2<=n<=5*104
- edges.length==n-1
- 0<=ai,bi<n
- values.length==n
- 1<=values[i]<=109
- Itisguaranteedthatedgesrepresentsavalidtree.

PROGRAM:

```
def maximum(n, edges, values):
    tree = defaultdict(list)
    for a, b in edges:
        tree[a].append(b)
        tree[b].append(a)
        subsum = [0] * n
    visited = [False] * n

    def dfs(node):
        visited[node] = True
        currentsum = values[node]
        for neighbor in tree[node]:
            if not visited[neighbor]:
                currentsum += dfs(neighbor)
        subsum[node] = currentsum
        return currentsum

    dfs(0)

    allsums = set(subsum)

    maxxor = 0
    allsums = list(allsums)
    for i in range(len(allsums)):
        for j in range(i + 1, len(allsums)):
            maxxor = max(maxxor, allsums[i] ^ allsums[j])

    return maxxor

n1 = 6
edges1 = [[0, 1], [0, 2], [1, 3], [1, 4], [2, 5]]
```

```
values1 = [2, 8, 3, 6, 2, 5]

print(maximum (n1, edges1, values1))
```

OUTPUT: 24

2.FormaChemicalBond

SQLSchema

Table:Elements

```
+-------------+---------+

|ColumnName|Type |

+-------------+---------+

|symbol |varchar|

| type |enum |

|electrons | int |

+-------------+---------+
```

symbolistheprimarykeyforthistable.

Eachrowofthistablecontainsinformationofoneelement.

typeisanENUMoftype('Metal', 'Nonmetal', 'Noble')-IftypeisNoble,electronsis0.-IftypeisMetal,electronsisthenumberofelectronsthatoneatomofthiselementcangive.-IftypeisNonmetal,electronsisthenumberofelectronsthatoneatomofthiselement

needs.

Twoelementscanformabondifoneofthemis'Metal'andtheotheris'Nonmetal'.WriteanSQL

querytofindallthepairsofelementsthatcanformabond.Returntheresulttableinany

order.Thequeryresultformatisinthefollowingexample.

Example1:

Input:

Elementstable:

```
+--------+----------+-----------+

|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:

```
SELECT
   m.symbol AS metal,
   n.symbol AS nonmetal
FROM
   Elements m
```

```
JOIN
   Elements n
ON
   m.type = 'Metal' AND n.type = 'Nonmetal';
```

OUTPUT:

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

3. Minimum Cuts to Divide a Circle

Avalid cut in a circle can be:

Acut 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.

It can be shown that less than 3 cuts cannot result in 3 slices of equal size and shape.

Also note that the first cut will not divide the circle into distinct parts.

Constraints:

- 1 <=n<=100

PROGRAM:

```
def min (n):
   if n == 1:
       return 0
   elif n % 2 == 0:
       return n // 2
   else:
       return n


print(min(4))
```

Output: 2

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 (customers):
   n = len(customers)


   totalY = customers.count('Y')
```

```
    prefixN = [0] * (n + 1)
    prefixY = [0] * (n + 1)

    for i in range(n):
        prefixN[i + 1] = prefixN[i] + (1 if customers[i] == 'N' else 0)
        prefixY[i + 1] = prefixY[i] + (1 if customers[i] == 'Y' else 0)

    minpenalty = float('inf')
    besthour = 0

    for j in range(n + 1):
        openpenalty = prefixN[j]
        closedpenalty = prefixY[n] - prefixY[j]
        totalpenalty = openpenalty + closedpenalty

        if totalpenalty < minpenalty:
            minpenalty = totalpenalty
            besthour = j

    return besthour

print(min_penalty_closing_hour("YYNY"))
```

OUTPUT: 2

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(customers):
    n = len(customers)

    totalY = customers.count('Y')

    prefixN = [0] * (n + 1)

    prefixY = [0] * (n + 1)


    for i in range(n):

        prefixN[i + 1] = prefixN[i] + (1 if customers[i] == 'N' else 0)

        prefixY[i + 1] = prefixY[i] + (1 if customers[i] == 'Y' else 0)


    minpenalty = float('inf')

    besthour = 0


    for j in range(n + 1):

        openpenalty = prefixN[j]

        closedpenalty = totalY - prefixY[j]

        totalpenalty = openpenalty + closedpenalty


        if totalpenalty < minpenalty:

            minpenalty = totalpenalty

            besthour = j


    return besthour


print(min ("YYNY"))
```

OUTPUT: 2

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:

● Astring is palindromic if it reads the same forward and backward.

- Asubsequence 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.

PROGRAM:

```
def count (s):
   MOD = 10**9 + 7
   n = len(s)
   count = 0

   if n < 5:
      return 0
```

```
    dp3 = [[0] * 10 for _ in range(n)]
    for i in range(n):
        countleft = [0] * 10
        for j in range(i + 1, n):
            if s[i] == s[j]:
                for k in range(10):
                    dp3[j][k] = (dp3[j][k] + countleft[k]) % MOD
            count_left[int(s[j])] += 1

    count_left = [[0] * 10 for _ in range(n)]
    count_right = [[0] * 10 for _ in range(n)]

    for i in range(n):
        for j in range(10):
            if i > 0:
                countleft[i][j] = countleft[i - 1][j]
            countleft[i][int(s[i])] += 1

    for i in range(n - 1, -1, -1):
        for j in range(10):
            if i < n - 1:
                countright[i][j] = countright[i + 1][j]
            countright[i][int(s[i])] += 1

    for i in range(1, n - 1):
        for j in range(10):
            count = (count + dp3[i][j] * countright[i + 1][j]) % MOD

    return count

print(count_palindromic_subsequences("103301"))
```

OUTPUT: 2

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:

```
import math


def pivot(n):
   discriminant = 2 * n * n + 2 * n + 1
   sqrt_discriminant = math.isqrt(discriminant)


   if sqrt_discriminant * sqrt_discriminant == discriminant:
      x = (sqrt_discriminant - 1) // 2
```

```
        if 1 <= x <= n:
            return x

    return -1

print(pivot (8))
```

Output: 6

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.

Asubsequence 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 ("co

achingding

").

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 ("a

Example 3:

Input: s = "z", t = "abcde"

Output: 5

bcde").

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_appends_to_make_subsequence(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

print(min ("coaching", "coding"))
```

Output: 4

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 remove (head):
    if not head or not head.next:
        return head


    dummy = ListNode(0)
    dummy.next = head
    current = dummy
    maxval = float('-inf')


    while current.next:
        if current.next.val > maxval:
            maxval = current.next.val
            current.next = current.next.next
        else:
            current = current.next


    return dummy.next
```

```
def print_linked_list(head):
    while head:
        print(head.val, end=" -> ")
        head = head.next
    print("None")

head1 = ListNode(5)
head1.next = ListNode(2)
head1.next.next = ListNode(13)
head1.next.next.next = ListNode(3)
head1.next.next.next.next = ListNode(8)

print("Original Linked List:")
printlinkedList(head1)

modifiedhead1 = remove (head1)
print("\nModified Linked List:")
printLinkedlist(modifiedhead1)

head2 = ListNode(1)
head2.next = ListNode(1)
head2.next.next = ListNode(1)
head2.next.next.next = ListNode(1)

print("\nOriginal Linked List:")
printlinkedlist(head2)

modifiedhead2 = remove(head2)
print("\nModified Linked List:")
printlinkedlist(modifiedhead2)
```

OUTPUT:

Original Linked List:

5 -> 2 -> 13 -> 3 -> 8 -> None

Modified Linked List:

13 -> 8 -> None

Original Linked List:

1 -> 1 -> 1 -> 1 -> None

Modified Linked List:

1 -> 1 -> 1 -> 1 -> None

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:

● Themedian 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.

● Asubarray 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

PROGRAM:

```
def countarray(nums, k):
    count = 0
    n = len(nums)

    for i in range(n):
        left = right = i
        while left >= 0 and right < n:
            while right + 1 < n and nums[right + 1] < nums[i]:
                right += 1
            while left - 1 >= 0 and nums[left - 1] < nums[i]:
                left -= 1

            if nums[left:right + 1].count(k) >= (right - left + 1) // 2 + 1:
                count += 1

            right += 1

    return count

nums1 = [3, 2, 1, 4, 5]
k1 = 4
print(count (nums1, k1))
```

Output: 3