# 1.Maximum XOR of Two Non-Overlapping Subtree

In [5]:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def get_subtree_xors(node):
    if not node:
        return 0, []
    left_xor, left_subtree_xors = get_subtree_xors(node.left)
    right_xor, right_subtree_xors = get_subtree_xors(node.right)
    current_xor = node.val ^ left_xor ^ right_xor
    all_xors = left_subtree_xors + right_subtree_xors + [current_xor]
    return current_xor, all_xors
def find_max_xor_of_two_subtrees(root):
    _, all_subtree_xors = get_subtree_xors(root)
    max_xor = 0
    n = len(all_subtree_xors)
    for i in range(n):
        for j in range(i + 1, n):
            max_xor = max(max_xor, all_subtree_xors[i] ^ all_subtree_xors[j
    return max_xor
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
root.right.left = TreeNode(6)
root.right.right = TreeNode(7)
print(find_max_xor_of_two_subtrees(root))
```

```
7
```

# 2. Form a Chemical

In [20]:

```python
from tabulate import tabulate
chemical_elements = {
    "H": {"name": "Hydrogen"},
    "He": {"name": "Helium"},
    "Li": {"name": "Lithium"},
    "Be": {"name": "Beryllium"},
    "B": {"name": "Boron"},
}
def create_element_table(elements):
    table = [["Symbol", "Name"]]
    for symbol, info in elements.items():
        table.append([symbol, info["name"]])
    return table
print(tabulate(create_element_table(chemical_elements), headers="firstrow",
```

```
+----------+-----------+
| Symbol   | Name      |
+==========+===========+
| H        | Hydrogen  |
+----------+-----------+
| He       | Helium    |
+----------+-----------+
| Li       | Lithium   |
+----------+-----------+
| Be       | Beryllium |
+----------+-----------+
| B        | Boron     |
+----------+-----------+
```

# 3. Minimum Cuts to Divide a Circle

In [9]:

```python
def min_cuts_to_divide_circle(n):
    if n <= 0:
        return "Invalid input: No of parts should be greater than 0"
    return n - 1
n = int(input("Enter the no of equal parts to divide the circle into: "))
print("Mini cuts requi:", min_cuts_to_divide_circle(n))
```

```
Enter the no of equal parts to divide the circle into: 6
Mini cuts requi: 5
```

# 4. Difference Between Ones and Zeros in Row and Columns

In [10]:
```
def calculate_diffe(matrix):
    row_differences = []
    col_differences = []
    for row in matrix:
        ones_count = row.count(1)
        zeros_count = row.count(0)
        row_differences.append(abs(ones_count - zeros_count))
    num_cols = len(matrix[0])
    for j in range(num_cols):
        ones_count = sum(matrix[i][j] == 1 for i in range(len(matrix)))
        zeros_count = sum(matrix[i][j] == 0 for i in range(len(matrix)))
        col_differences.append(abs(ones_count - zeros_count))

    return row_differences, col_differences
matrix = [
    [1, 0, 1],
    [0, 1, 0],
    [1, 1, 1]
]
row_diff, col_diff = calculate_diffe(matrix)
print("Difference in rows:", row_diff)
print("Difference in columns:", col_diff)
```

```
Difference in rows: [1, 1, 3]
Difference in columns: [1, 1, 1]
```

# 5. Minimum Penalty for a Shop

In [21]:
```
def min_penalty(graph, start, end):
    num_shops = len(graph)
    INF = float('inf')
    dist = [[INF] * num_shops for _ in range(num_shops)]
    for i in range(num_shops):
        for j in range(num_shops):
            if i == j:
                dist[i][j] = 0
            elif graph[i][j] != -1:
                dist[i][j] = graph[i][j]
    for k in range(num_shops):
        for i in range(num_shops):
            for j in range(num_shops):
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
    return dist[start][end]
graph = [
    [-1, 2, 5, 1],
    [2, -1, 3, 2],
    [5, 3, -1, 1],
    [1, 2, 1, -1]
]
start = 0
end = 3
print("Mini penalty for the shop:", min_penalty(graph, start, end))
```

```
Mini penalty for the shop: 1
```

# 6. Count Palindromic Subsequence

In [22]:

```
def count_palindrom_subseq(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] + dp[i][j - 1] + 1
            else:
                dp[i][j] = dp[i + 1][j] + dp[i][j - 1] - dp[i + 1][j - 1]
    return dp[0][n - 1]
s = "aab"
print("Number of palindrom subseq:", count_palindrom_subseq(s))
```

```
Number of palindrom subseq: 4
```

# 7.pivot integer

In [13]:

```
def find_pivot_integer(nums):
    total_sum = sum(nums)
    left_sum = 0
    for i, num in enumerate(nums):
        if left_sum == total_sum - left_sum - num:
            return num
        left_sum += num
    return -1
nums = [1, 7, 3, 6, 5, 6]
print("Pivot Integer:", find_pivot_integer(nums))
```

```
Pivot Integer: 6
```

# 8.Append Characters to String to Make Subsequence

In [23]:

```
def append_to_make_subseq(s, target):
    i = 0
    result = ""
    for char in target:
        while i < len(s) and s[i] != char:
            i += 1
        if i == len(s):
            result += char
        else:
            result += s[i]
            i += 1
    return result
s = "abcde"
target = "ace"
print("Characters:", append_to_make_subseq(s, target))
```

```
Characters: ace
```

# 9.Remove Nodes From Linked List

In [17]:

```
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next
def removeNodes(head, val):
    dummy = ListNode(0)
    dummy.next = head
    current = dummy
    while current.next:
        if current.next.value == val:
            current.next = current.next.next
        else:
            current = current.next
    return dummy.next
def printList(head):
    while head:
        print(head.value, end=" -> ")
        head = head.next
    print("None")
def createLinkedList(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head
values = [1, 2, 6, 3, 4, 5, 6]
head = createLinkedList(values)
print("Original list:")
printList(head)
head = removeNodes(head, 6)
print("List after removing 6:")
printList(head)
```

```
Original list:
1 -> 2 -> 6 -> 3 -> 4 -> 5 -> 6 -> None
List after removing 6:
1 -> 2 -> 3 -> 4 -> 5 -> None
```

# 10.Count Subarrays With Median k

In [24]:

```
def countSubarray(nums, k):
    k_index = nums.index(k)
    prefix_sum = 0
    prefix_count = {0: 1}
    result = 0
    for i in range(len(nums)):
        if nums[i] < k:
            prefix_sum -= 1
        elif nums[i] > k:
            prefix_sum += 1
        if i < k_index:
            prefix_count[prefix_sum] = prefix_count.get(prefix_sum, 0) + 1
        else:
            result += prefix_count.get(prefix_sum, 0) + prefix_count.get(pre
    return result
nums = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
k = 5
print(countSubarray(nums, k))
```

```
10
```

In [ ]: