

ASSIGNMENT 3

1 . Counting Elements

CODE :

```
def countElements(arr,n):
    count = 0
    for i in range(n):
        x = arr[i]
        xPlusOne = x + 1
        found = False
        for j in range(i + 1,n,1):
            if (arr[j] == xPlusOne):
                found = True
                break
            k = i - 1
        while(found == False and k >= 0):
            if (arr[k] == xPlusOne):
                found = True
                break
            k -= 1
        if (found == True):
            count += 1

    return count

# Driver program
if __name__ == '__main__':
    arr = [1, 2, 3]
    n = len(arr)
```

OUTPUT :

2



2 . Perform String Shifts

CODE :

```
def stringShift(s, shift):
    val = 0
    for i in range(len(shift)):
```

```

val += -shift[i][1] if shift[i][0] == 0 else shift[i][1]
Len = len(s)
val = val % Len
result = ""
if (val > 0):
    result = s[Len - val:Len] + s[0: Len - val]
else:
    result = s[-val: Len] + s[0: -val]

print(result)

```

Driver Code

```
s = "abc"
```

```
shift = [
```

```
    [ 0, 1 ],
```

```
    [ 1, 2 ]
```

```
]
```

```
stringShift(s, shift)
```

OUTPUT :



3 . Leftmost Column with at Least a One

CODE :

```
import sys
```

```
N = 3
```

```
def search(mat, n, m):
```

```
    a = sys.maxsize
```

```
    for i in range (n):
```

```
        low = 0
```

```
        high = m - 1
```

```
        ans = sys.maxsize
```

```
        while (low <= high):
```

```
            mid = (low + high) // 2
```

```
            if (mat[i][mid] == 1):
```

```
                if (mid == 0):
```

```
                    ans = 0
```

```
                    break
```

```
            elif (mat[i][mid - 1] == 0):
```

```

                                ans = mid
                                break

        if (mat[i][mid] == 1):
            high = mid - 1
        else:
            low = mid + 1

    if (ans < a):
        a = ans
    if (a == sys.maxsize):
        return -1
    return a + 1

```

Driver Code

```
if __name__ == "__main__":
```

```

    mat = [[0, 0, 0],
            [0, 0, 1],
            [0, 1, 1]]
    print(search(mat, 3, 3))

```

OUTPUT :

2

2



4 . First Unique Number

CODE :

```
class FirstUnique:
```

```

    def __init__(self, nums: List[int]):
        self.cnt = Counter(nums)
        self.q = deque(nums)

```

```

    def showFirstUnique(self) -> int:
        while self.q and self.cnt[self.q[0]] != 1:
            self.q.popleft()
        return -1 if not self.q else self.q[0]

```

```

    def add(self, value: int) -> None:
        self.cnt[value] += 1
        self.q.append(value)

```

OUTPUT :

5 . Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree
Given a binary tree where each path going from the root to any leaf forms a valid sequence, check if a given string is a valid sequence in such a binary tree. We get the given string from the concatenation of an array of integers arr and the concatenation of all values of the nodes along a path results in a sequence in the given binary tree.

CODE :

```
class Node:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None

def existPathUtil(root, arr, n, index):
    if not root or index == n:
        if not root.left and not root.right:
            if root.val == arr[index] and index == n-1:
                return True
            return False

    can be either in left subtree or
    # right subtree.
    return ((index < n) and (root.val == arr[index]) and \
            (existPathUtil(root.left, arr, n, index+1) or \
             existPathUtil(root.right, arr, n, index+1)))

def existPath(root, arr, n, index):
    if not root:
        return (n == 0)

    return existPathUtil(root, arr, n, 0)

# Driver Code
if __name__ == "__main__":
    arr = [5, 8, 6, 7]
```

```

n = len(arr)
root = Node(5)
root.left = Node(3)
root.right = Node(8)
root.left.left = Node(2)
root.left.right = Node(4)
root.left.left.left = Node(1)
root.right.left = Node(6)
root.right.left.right = Node(7)

if existPath(root, arr, n, 0):
    print("Path Exists")
else:
    print("Path does not Exist")

```

OUTPUT :

Output

Path Exists

6 . Kids With the Greatest Number of Candies

CODE : `def kidsWithCandies(candies, extraCandies):`

`# Create an empty list to store the results`

`result = []`

`# Loop through the list of candies`

`for i in range(len(candies)):`

`# If the current candy + extraCandies is greater than or equal to the maximum candy in the list, append True to the result list. Otherwise, append False.`

`if candies[i] + extraCandies >= max(candies):`

`result.append(True)`

`else:`

`result.append(False)`

OUTPUT :

Output

[True , False , True , True , False]

7 . Max Difference You Can Get From Changing an Integer

CODE :

```
def morethanNbyK(arr, n, k):
    x = n // k
    freq = {}

    for i in range(n):
        if arr[i] in freq:
            freq[arr[i]] += 1
        else:
            freq[arr[i]] = 1
    for i in freq:
        if (freq[i] > x):
            print(i)

# Driver code
if __name__ == '__main__':
    arr = [1, 1, 2, 2, 3, 5, 4, 2, 2, 3, 1, 1, 1]
    n = len(arr)
    k = 4

    morethanNbyK(arr, n, k)
```

OUTPUT :

```
Output
1
2

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

8 . Check If a String Can Break Another String

CODE :

```
n1 = len(str1)
n2 = len(str2)
if (n1 != n2):
    return False

# Sort both strings
a = sorted(str1)
str1 = " ".join(a)
b = sorted(str2)
```

```

str2 = " ".join(b)

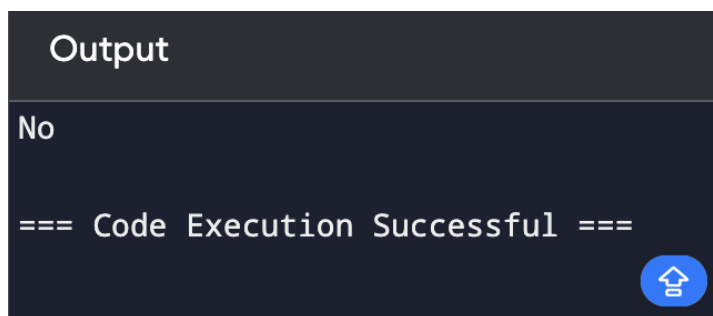
# Compare sorted strings
for i in range(0, n1, 1):
    if (str1[i] != str2[i]):
        return False

return True

# Driver Code
if __name__ == '__main__':
    str1 = "test"
    str2 = "ttew"
    if (arePermutation(str1, str2)):
        print("Yes")
    else:
        print("No")

```

OUTPUT :



9 . Number of Ways to Wear Different Hats to Each Other

CODE : class Solution:

```

def numberWays(self, hats: List[List[int]]) -> int:
    g = defaultdict(list)
    for i, h in enumerate(hats):
        for v in h:
            g[v].append(i)
    mod = 10**9 + 7
    n = len(hats)
    m = max(max(h) for h in hats)
    f = [[0] * (1 << n) for _ in range(m + 1)]
    f[0][0] = 1
    for i in range(1, m + 1):
        for j in range(1 << n):
            f[i][j] = f[i - 1][j]
            for k in g[i]:
                if j >> k & 1:
                    f[i][j] = (f[i][j] + f[i - 1][j ^ (1 << k)]) % mod

```

```
return f[m][-1]
```

OUTPUT :

```
Enter number of people 2
Enter number of hats and hats
3
3
5
1
Enter number of hats and hats
2
3
5
Number of ways to wear different hats to each other for N people 4
```

10 . Next Permutation

CODE :

```
def next_permutation(nums):
    # Find the first element from the right that is not in decreasing order
    i = len(nums) - 2
    while i >= 0 and nums[i] >= nums[i + 1]:
        i -= 1
    # If such an element is found, find the smallest element from the right that is greater than it
    if i >= 0:
        j = len(nums) - 1
        while nums[j] <= nums[i]:
            j -= 1
        # Swap the two elements
        nums[i], nums[j] = nums[j], nums[i]
    # Reverse the elements from i+1 to the end to get the next permutation
    nums[i + 1:] = reversed(nums[i + 1:])
```

```
nums = [3, 2, 1]
next_permutation(nums)
print(nums)
```

OUTPUT :

Output

[1, 2, 3]

=== Code Execution Successful ===

