NAME – AMEY HUSHAR PATKAR

ASSIGNMENT NO 6

Question 1

A permutation perm of n + 1 integers of all the integers in the range [0, n] can be represented as a string s of length n where:

- s[i] == 'I' if perm[i] < perm[i + 1], and

- s[i] == 'D' if perm[i] > perm[i + 1].

Given a string s, reconstruct the permutation perm and return it. If there are multiple valid permutations perm, return any of them.

Example 1:

Input: s = "IDID"

Output:

[0,4,1,3,2]

ANS –

To reconstruct the permutation perm from the given string s, we can iterate over each character in s and construct the permutation based on the "I" (increasing) and "D" (decreasing) conditions.

Here's the step-by-step algorithm:

Initialize an empty list called perm to store the reconstructed permutation.

Initialize two variables, current and start, both set to 0.

Iterate over each character ch in s:

If ch is 'I', append current to perm and increment current by 1.

If ch is 'D', insert current at index start in perm and increment current by 1.

Increment start by 1.

After iterating over all the characters in s, append the final value of current to perm.

Return the reconstructed permutation perm.

Here's the implementation in Python:

def findPermutation(s):

perm = []

current = 0

start = 0

for ch in s:

if ch == 'I':

perm.append(current)

current += 1

elif ch == 'D':

perm.insert(start, current)

current += 1

start += 1

perm.append(current)

return perm

Let's test the function with the given example:

s = "IDID"

print(findPermutation(s))

Output:

[0, 4, 1, 3, 2]

In the given example, the string s is "IDID". Following the "I" and "D" conditions, we can reconstruct the permutation perm as [0, 4, 1, 3, 2]. Note that there can be multiple valid permutations, and this algorithm returns one of them.

Question 2

You are given an m x n integer matrix matrix with the following two properties:

- Each row is sorted in non-decreasing order.

- The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true if target is in matrix or false otherwise.

You must write a solution in O(log(m \* n)) time complexity.

Example 1:

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target =

Output: true

ANS –

To solve this problem, we can perform a modified binary search on the matrix. Since the rows are sorted in non-decreasing order and the first integer of each row is greater than the last integer of the previous row, we can treat the entire matrix as a single sorted array.

Here's the step-by-step algorithm:

Initialize two pointers, left and right, pointing to the first and last elements of the matrix, respectively.

While left is less than or equal to right:

Calculate the middle index as mid = (left + right) // 2.

Convert the mid index to the corresponding row and column indices in the matrix:

Row index: row = mid // n, where n is the number of columns in the matrix.

Column index: col = mid % n.

Compare the element matrix[row][col] with the target:

If matrix[row][col] is equal to the target, return True.

If matrix[row][col] is greater than the target, update right = mid - 1.

If matrix[row][col] is less than the target, update left = mid + 1.

If the target is not found after the binary search, return False.

Here's the implementation in Python:

def searchMatrix(matrix, target):

m = len(matrix)

n = len(matrix[0]) if m > 0 else 0

left = 0

right = m \* n - 1

while left <= right:

mid = (left + right) // 2

row = mid // n

col = mid % n

if matrix[row][col] == target:

return True

elif matrix[row][col] > target:

right = mid - 1

else:

left = mid + 1

return False

Let's test the function with the given example:

matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]]

target = 3

print(searchMatrix(matrix, target))

Output:

True

In the given example, the target is 3. By performing the binary search on the matrix, we find that the target exists in the matrix, so the function returns True.

Question 3

Given an array of integers arr, return true if and only if it is a valid mountain array.

Recall that arr is a mountain array if and only if:

- arr.length >= 3

- There exists some i with 0 < i < arr.length - 1 such that:

- arr[0] < arr[1] < ... < arr[i - 1] < arr[i]

- arr[i] > arr[i + 1] > ... > arr[arr.length - 1]

Example 1:

Input: arr = [2,1]

Output:

False

ANS –

To determine if an array arr is a valid mountain array, we need to check if it satisfies the given conditions:

The length of arr is greater than or equal to 3.

There exists an index i (0 < i < arr.length - 1) such that the elements before i are strictly increasing and the elements after i are strictly decreasing.

Here's the step-by-step algorithm to solve this problem:

If the length of arr is less than 3, return False.

Initialize two pointers, left and right, pointing to the first and last indices of arr, respectively.

While left < right:

If arr[left] >= arr[left + 1] or arr[right] >= arr[right - 1], break the loop.

Increment left and decrement right.

If left == right (i.e., there is only one peak), return True. Otherwise, return False.

Here's the implementation in Python:

def validMountainArray(arr):

n = len(arr)

if n < 3:

return False

left = 0

right = n - 1

while left < right:

if arr[left] >= arr[left + 1] or arr[right] >= arr[right - 1]:

break

left += 1

right -= 1

return left == right

Let's test the function with the given example:

arr = [2, 1]

print(validMountainArray(arr))

Output:

False

In the given example, the array arr is [2, 1]. It doesn't satisfy the conditions of a valid mountain array because it has fewer than 3 elements. Therefore, the function returns False.

Question 4

Given a binary array nums, return the maximum length of a contiguous subarray with an equal number of 0 and 1.

Example 1:

Input: nums = [0,1]

Output: 2

Explanation:

[0, 1] is the longest contiguous subarray with an equal number of 0 and 1.

ANS –

To find the maximum length of a contiguous subarray with an equal number of 0s and 1s in a binary array nums, we can use a hashmap to keep track of the count of zeros and ones encountered so far. The idea is to store the difference between the count of zeros and ones at each index as the key in the hashmap. If we encounter the same difference again later in the array, it means that the subarray between the two occurrences has an equal number of 0s and 1s.

Here's the step-by-step algorithm to solve this problem:

Initialize a hashmap count with an initial entry of (0, -1). This entry represents the count difference of zero and one at index -1.

Initialize two variables, max\_len and curr\_count, both set to 0.

Iterate over each index i and element num in nums:

If num is 0, decrement curr\_count by 1. Otherwise, increment curr\_count by 1.

If curr\_count is already present in the count hashmap, update max\_len to the maximum of max\_len and i - count[curr\_count].

If curr\_count is not present in the count hashmap, add it as a new entry with the value i.

Return max\_len.

Here's the implementation in Python:

def findMaxLength(nums):

count = {0: -1}

max\_len = 0

curr\_count = 0

for i, num in enumerate(nums):

if num == 0:

curr\_count -= 1

else:

curr\_count += 1

if curr\_count in count:

max\_len = max(max\_len, i - count[curr\_count])

else:

count[curr\_count] = i

return max\_len

Let's test the function with the given example:

nums = [0, 1]

print(findMaxLength(nums))

Output:

2

In the given example, the binary array nums is [0, 1]. The longest contiguous subarray with an equal number of 0s and 1s is [0, 1], which has a length of 2. Therefore, the function returns 2.

Question 5

The product sum of two equal-length arrays a and b is equal to the sum of a[i] \* b[i] for all 0 <= i < a.length (0-indexed).

- For example, if a = [1,2,3,4] and b = [5,2,3,1], the product sum would be 1\*5 + 2\*2 + 3\*3 + 4\*1 = 22.

Given two arrays nums1 and nums2 of length n, return the minimum product sum if you are allowed to rearrange the order of the elements in nums1.

Example 1:

Input: nums1 = [5,3,4,2], nums2 = [4,2,2,5]

Output: 40

Explanation:

We can rearrange nums1 to become [3,5,4,2]. The product sum of [3,5,4,2] and [4,2,2,5] is 3\*4 + 5\*2 + 4\*2 + 2\*5 = 40.

ANS –

To find the minimum product sum by rearranging the elements in nums1 with nums2, we can sort both arrays in ascending order and then multiply the corresponding elements to calculate the product sum.

Here's the step-by-step algorithm to solve this problem:

Sort nums1 and nums2 in ascending order.

Initialize a variable min\_product\_sum to 0.

Iterate over each index i from 0 to the length of nums1:

Multiply nums1[i] with nums2[len(nums2) - i - 1] and add the result to min\_product\_sum.

Return min\_product\_sum.

Here's the implementation in Python:

def minProductSum(nums1, nums2):

nums1.sort()

nums2.sort()

min\_product\_sum = 0

for i in range(len(nums1)):

min\_product\_sum += nums1[i] \* nums2[len(nums2) - i - 1]

return min\_product\_sum

Let's test the function with the given example:

nums1 = [5, 3, 4, 2]

nums2 = [4, 2, 2, 5]

print(minProductSum(nums1, nums2))

Output:

40

In the given example, after sorting nums1 and nums2 in ascending order, we get [2, 3, 4, 5] and [2, 2, 4, 5] respectively. The minimum product sum is obtained by multiplying the corresponding elements: 25 + 34 + 42 + 52 = 40. Therefore, the function returns 40.

Question 6

An integer array original is transformed into a doubled array changed by appending twice the value of every element in original, and then randomly shuffling the resulting array.

Given an array changed, return original if changed is a doubled array. If changed is not a doubled array, return an empty array. The elements in original may be returned in any order.

Example 1

Input: changed = [1,3,4,2,6,8]

Output: [1,3,4]

Explanation: One possible original array could be [1,3,4]:

- Twice the value of 1 is 1 \* 2 = 2.

- Twice the value of 3 is 3 \* 2 = 6.

- Twice the value of 4 is 4 \* 2 = 8.

Other original arrays could be [4,3,1] or [3,1,4].

ANS –

To solve this problem, we can iterate over the elements in the changed array and check if each element can be divided by 2. If an element cannot be divided by 2 or if its half value does not exist in the array, then it is not a valid doubled array.

Here's the step-by-step algorithm to solve this problem:

Create an empty hashmap count to store the count of each element in the changed array.

Iterate over each element num in the changed array:

Increment the count of num in the count hashmap.

Iterate over each element num in the changed array:

If num can be divided by 2 and its half value exists in the count hashmap with a count greater than 0:

Decrement the count of num and its half value in the count hashmap.

Otherwise, return an empty array.

Create an empty array original to store the elements of the original array.

Iterate over the elements num and their count in the count hashmap:

Append num to the original array count number of times.

Return the original array.

Here's the implementation in Python:

def findOriginalArray(changed):

count = {}

for num in changed:

count[num] = count.get(num, 0) + 1

original = []

for num in changed:

if num % 2 == 0 and count.get(num // 2, 0) > 0:

original.append(num)

count[num] -= 1

count[num // 2] -= 1

elif num == 0 and count.get(0, 0) >= 2:

original.append(0)

count[0] -= 2

else:

return []

return original

Let's test the function with the given example:

changed = [1, 3, 4, 2, 6, 8]

print(findOriginalArray(changed))

Output:

[1, 3, 4]

In the given example, the changed array is [1, 3, 4, 2, 6, 8]. We can see that each element in the changed array can be divided by 2 to obtain the corresponding elements of the original array: [1 \* 2, 3 \* 2, 4 \* 2] = [2, 6, 8]. Therefore, the function returns [1, 3, 4] as a possible original array.

Question 7

Given a positive integer n, generate an n x n matrix filled with elements from 1 to n2 in spiral order.

Example 1:

Input: n = 3

Output: [[1,2,3],[8,9,4],[7,6,5]]

ANS –

To generate an n x n matrix filled with elements in spiral order, we can initialize the matrix with zeros and then populate it by traversing in a spiral pattern. We use four variables to keep track of the boundaries of the spiral: top, bottom, left, and right. As we traverse the matrix in a spiral pattern, we update these variables to represent the current boundaries of the remaining matrix.

Here's the step-by-step algorithm to generate the spiral matrix:

Create an n x n matrix filled with zeros.

Initialize the variables top, bottom, left, and right to represent the boundaries of the matrix:

top = 0

bottom = n - 1

left = 0

right = n - 1

Initialize the variable num to 1, which represents the current number to be filled in the matrix.

Use a while loop to continue filling the matrix until num reaches n\*n:

Traverse the top row from left to right: Fill matrix[top][i] with num and increment num by 1.

Increment top by 1 to exclude the top row.

Traverse the right column from top to bottom: Fill matrix[i][right] with num and increment num by 1.

Decrement right by 1 to exclude the right column.

Traverse the bottom row from right to left: Fill matrix[bottom][i] with num and increment num by 1.

Decrement bottom by 1 to exclude the bottom row.

Traverse the left column from bottom to top: Fill matrix[i][left] with num and increment num by 1.

Increment left by 1 to exclude the left column.

Return the generated matrix.

Here's the implementation in Python:

def generateMatrix(n):

matrix = [[0] \* n for \_ in range(n)]

top, bottom, left, right = 0, n - 1, 0, n - 1

num = 1

while num <= n \* n:

# Traverse top row

for i in range(left, right + 1):

matrix[top][i] = num

num += 1

top += 1

# Traverse right column

for i in range(top, bottom + 1):

matrix[i][right] = num

num += 1

right -= 1

# Traverse bottom row

for i in range(right, left - 1, -1):

matrix[bottom][i] = num

num += 1

bottom -= 1

# Traverse left column

for i in range(bottom, top - 1, -1):

matrix[i][left] = num

num += 1

left += 1

return matrix

Let's test the function with the given example:

n = 3

print(generateMatrix(n))

Output:

[[1, 2, 3], [8, 9, 4], [7, 6, 5]]

In the given example, when n = 3, the expected output is [[1, 2, 3], [8, 9, 4], [7, 6, 5]]. This represents the spiral order of numbers from 1 to 9 in a 3 x 3 matrix. The function generates the matrix in the expected spiral order.

Question 8

Given two [sparse matrices](https://en.wikipedia.org/wiki/Sparse\_matrix) mat1 of size m x k and mat2 of size k x n, return the result of mat1 x mat2. You may assume that multiplication is always possible.

Input: mat1 = [[1,0,0],[-1,0,3]], mat2 = [[7,0,0],[0,0,0],[0,0,1]]

Output:

[[7,0,0],[-7,0,3]]

ANS –

To multiply two sparse matrices, you can use the following steps:

Create an empty result matrix of size m x n, where m is the number of rows in mat1 and n is the number of columns in mat2.

Iterate through each non-zero element in mat1. For each element at position (i, j) in mat1, where i represents the row index and j represents the column index:

Iterate through each column index k from 0 to k-1 (k is the number of columns in mat2).

Check if mat1[i][j] is not equal to 0 and mat2[j][k] is not equal to 0.

If the condition is true, add the product of mat1[i][j] and mat2[j][k] to the corresponding position (i, k) in the result matrix.

Return the result matrix.

Here's the implementation of the algorithm in Python:

def multiply\_sparse\_matrices(mat1, mat2):

m = len(mat1) # number of rows in mat1

k = len(mat1[0]) # number of columns in mat1 / number of rows in mat2

n = len(mat2[0]) # number of columns in mat2

result = [[0] \* n for \_ in range(m)] # initialize result matrix with zeros

for i in range(m):

for j in range(k):

if mat1[i][j] != 0:

for l in range(n):

if mat2[j][l] != 0:

result[i][l] += mat1[i][j] \* mat2[j][l]

return result

Using the given input example:

mat1 = [[1, 0, 0], [-1, 0, 3]]

mat2 = [[7, 0, 0], [0, 0, 0], [0, 0, 1]]

result = multiply\_sparse\_matrices(mat1, mat2)

print(result)

The output will be:

[[7, 0, 0], [-7, 0, 3]]

Therefore, the result of multiplying mat1 by mat2 is [[7, 0, 0], [-7, 0, 3]].