ARRAY PRACTICE PROBLEM - 01

14 February 2024 20:15

Created By -
Shivam Barekar

HOMEWORK PROBLEMS - I

14 February 2024 19:15

Problem Statement – 01 (GeeksForGeeks) : Given an unsorted array Arr of N positive and negative numbers. Your task is to create an array of alternate positive and negative numbers without changing the relative order of positive and negative numbers.

Note: Array should start with a positive number and 0 (zero) should be considered a positive element.

Link : https://www.geeksforgeeks.org/problems/array-of-alternate-ve-and-ve-nos1401/1

Example 1:

Input:
N = 9
Arr[] = {9, 4, -2, -1, 5, 0, -5, -3, 2}

Output:
9 -2 4 -1 5 -5 0 -3 2

Explanation :

Positive elements : 9,4,5,0,2
Negative elements : -2,-1,-5,-3
As we need to maintain the relative order of
postive elements and negative elements we will pick
each element from the positive and negative and will
store them. If any of the positive and negative numbers
are completed. we will continue with the remaining signed
elements.The output is 9,-2,4,-1,5,-5,0,-3,2.

Example 2:

Input:
N = 10
Arr[] = {-5, -2, 5, 2, 4, 7, 1, 8, 0, -8}

Output:
5 -5 2 -2 4 -8 7 1 8 0

Explanation :

Positive elements : 5,2,4,7,1,8,0
Negative elements : -5,-2,-8
As we need to maintain the relative order of
postive elements and negative elements we will pick
each element from the positive and negative and will

store them. If any of the positive and negative numbers are completed. we will continue with the remaining signed elements.The output is 5,-5,2,-2,4,-8,7,1,8,0.

Your Task:

You don't need to read input or print anything. Your task is to complete the function rearrange() which takes the array of integers arr[] and n as parameters. You need to modify the array itself.

Expected Time Complexity: O(N)
Expected Auxiliary Space: O(N)

Constraints:

$1 \leq N \leq 10^7$

$-10^6 \leq Arr[i] \leq 10^7$

```java
/*
- Take 2-vectors one for storing positive elements and another one for negative elements.
- Take 2 pointers one is pointing to positive list indexes and another one is pointing to negative list
indexes and initialize it with zero.
- Run a loop from 0 to N-1 and Check
- If the index is even and positive elements are present in the list or there are no negative elements left in the
negative list then put the positive value at that index into the array and increase the positive pointer.
- Else put the Negative value at that index into the array and increase the Negative pointer.
*/

class Solution{
    void rearrange(int arr[], int n) {

        //create two ArrayLists to store negative and positive numbers separately
        ArrayList<Integer> neg = new ArrayList<Integer>();
        ArrayList<Integer> pos = new ArrayList<Integer>();

        //iterate through the array and add numbers to the appropriate ArrayList
        for (int i = 0; i < n; i++) {
            //if number is negative, add it to the neg ArrayList
            if (arr[i] < 0)
                neg.add(arr[i]);
            //if number is positive, add it to the pos ArrayList
            else
                pos.add(arr[i]);
        }

        //initialize variables for navigating through the neg and pos ArrayLists
        int i = 0, j = 0, k = 0;
        //rearrange the array by alternating between positive and negative numbers
        while (i < neg.size() && j < pos.size()) {
            arr[k++] = pos.get(j++); //add positive number to array
            arr[k++] = neg.get(i++); //add negative number to array
        }
        //add any remaining positive numbers to the array
        while (j < pos.size()) { arr[k++] = pos.get(j++); }

        //add any remaining negative numbers to the array
        while (i < neg.size()) { arr[k++] = neg.get(i++); }
    }
};
```

Problem Statement – 02 (GeeksForGeeks) : Given three arrays sorted in increasing order. Find the elements that are common in all three arrays.

Note: can you take care of the duplicates without using any additional Data Structure?

Link : https://www.geeksforgeeks.org/problems/common-elements1132/1

Example 1:

Input:
n1 = 6; A = {1, 5, 10, 20, 40, 80}
n2 = 5; B = {6, 7, 20, 80, 100}
n3 = 8; C = {3, 4, 15, 20, 30, 70, 80, 120}

Output:
20 80

Explanation:

20 and 80 are the only
common elements in A, B and C.

Your Task:

You don't need to read input or print anything. Your task is to complete the function commonElements() which take the 3 arrays A[], B[], C[] and their respective sizes n1, n2 and n3 as inputs and returns an array containing the common element present in all the 3 arrays in sorted order.
If there are no such elements return an empty array. In this case the output will be printed as -1.

Expected Time Complexity: O(n1 + n2 + n3)
Expected Auxiliary Space: O(n1 + n2 + n3)

Constraints:

$1 <= n1, n2, n3 <= 10^5$

The array elements can be both positive or negative integers.

```java
/*
- Three pointers (i, j, and k) are initialized to point to the first element of arrays A, B, and C, respectively.
- An ArrayList named res is created to store the common elements found in the three arrays.
- A variable last is initialized with Integer.MIN_VALUE. This variable is used to keep track of the last checked common
element to avoid duplicates.
- A while loop is employed to iterate through the arrays until any of the three arrays (A, B, C) reaches its end.
- Inside the loop, there is a conditional check to
- Determine if the current elements of all three arrays are equal and not the same as the last checked element
(A[i] == B[j] && A[i] == C[k] && A[i] != last).
- If the condition is true:
    The common element (A[i]) is added to the result ArrayList (res).
    The last variable is updated to the current common element.
    Pointers (i, j, k) are incremented to move to the next element in each array.
    If the current element of array A is the smallest among the three, the pointer for array A (i) is incremented.
    If the current element of array B is the smallest among the three, the pointer for array B (j) is incremented.
    If the current element of array C is the smallest among the three, the pointer for array C (k) is incremented.
    Once the loop ends, the function returns the ArrayList (res) containing the common elements found in all three
    arrays.
*/

class Solution
{
    // Function to find common elements in three arrays
    ArrayList<Integer> commonElements(int A[], int B[], int C[], int n1, int n2, int n3)
    {
        // Initialize three pointers for each array
        int i = 0, j = 0, k = 0;

        // Initialize an arraylist to store the common elements
        ArrayList<Integer> res = new ArrayList<Integer>();

        // Initialize a variable to store the last checked element
        int last = Integer.MIN_VALUE;

        // Loop until any of the three arrays reaches its end
        while (i < n1 && j < n2 && k < n3)
        {
            // If the current elements of all three arrays are equal and not the same as the last checked element
            if (A[i] == B[j] && A[i] == C[k] && A[i] != last)
            {
                // Add the common element to the result arraylist
                res.add (A[i]);

                // Update the last checked element
                last = A[i];

                // Move to the next element in each array
                i++;
                j++;
                k++;
            }
            // If the current element of array A is the smallest, move to the next element in array A
            else if (Math.min (A[i], Math.min(B[j], C[k])) == A[i]) i++;
            // If the current element of array B is the smallest, move to the next element in array B
            else if (Math.min (A[i], Math.min(B[j], C[k])) == B[j]) j++;
            // If the current element of array C is the smallest, move to the next element in array C
            else k++;
        }

        // Return the arraylist of common elements
        return res;
    }
}
```

Problem Statement – 03 (Leetcode) : Given an array nums of n integers where nums[i] is in the range [1, n], return an array of all the integers in the range [1, n] that do not appear in nums.

Link : https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/description/

Example 1:

Input: nums = [4,3,2,7,8,2,3,1]
Output: [5,6]

Example 2:

Input: nums = [1,1]
Output: [2]

Constraints:

n == nums.length
1 <= n <= 10^5
1 <= nums[i] <= n

```java
/*
We know that nums is of size is n and it contains only elements from [1, n].
We can map each element of the range [1, n] to the indices of nums from [0, n-1].

Thus, the above property can be used to mark if an element from range [1, n] is present in nums or not.

How?

- We can iterate over nums and for each element, we know it can be mapped to index nums[i]-1.
- We can therefore mark the element nums[i] as present in nums by making the element at index nums[i]-1 negative.
- Thus after iterating the array, we have - nums[i] < 0 or nums[i] is negative only if the element i+1 is present in
  the array.
- nums[i] > 0 or nums[i] is positive only if the element i+1 is not present in the array, We need to take care that
  some elements may already be negated.
- Thus, to avoid negative indexing or converting a negative element back to positive, we use abs() to get the absolute
  value of elements.
*/

import java.util.ArrayList;
import java.util.List;

class Solution {
    public List<Integer> findDisappearedNumbers(int[] nums) {
        List<Integer> ans = new ArrayList<>();
        for (int c : nums)
            nums[Math.abs(c) - 1] = -Math.abs(nums[Math.abs(c) - 1]); // mark c is present by negating nums[c-1]
        for (int i = 0; i < nums.length; i++)
            if (nums[i] > 0) ans.add(i + 1); // nums[i] > 0 means i+1 isn't present in nums
        return ans;
    }
}
```

Problem Statement – 04 (Codeforces) : Alex is solving a problem. He has n constraints on what the integer k can be. There are three types of constraints :
1- k must be greater than or equal to some integer x;
2- k must be less than or equal to some integer x;
3- k must be not equal to some integer x.

Help Alex find the number of integers k that satisfy all n constraints. It is guaranteed that the answer is finite (there exists at least one constraint of type 11 and at least one constraint of type 22). Also, it is guaranteed that no two constraints are the exact same.

Link : https://codeforces.com/contest/1920/problem/A

Input :

Each test consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 500$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($2 \le n \le 100$) — the number of constraints.

The following n lines describe the constraints. Each line contains two integers a and x ($a \in \{1,2,3\}, 1 \le x \le 10^9$). a denotes the type of constraint. If $a=1$, k must be greater than or equal to x. If $a=2$, k must be less than or equal to x. If $a=3$, k must be not equal to x.

It is guaranteed that there is a finite amount of integers satisfying all n constraints (there exists at least one constraint of type 1 and at least one constraint of type 2). It is also guaranteed that no two constraints are the exact same (in other words, all pairs (a,x) are distinct).

Output :

For each test case, output a single integer — the number of integers k that satisfy all n constraints.

Note :

In the first test case, $k \ge 3$ and $k \le 10$. Furthermore, $k \ne 1$ and $k \ne 5$. The possible integers k that satisfy the constraints are $3,4,6,7,8,9,10$. So the answer is 7.

In the second test case, $k \ge 5$ and $k \le 4$, which is impossible. So the answer is 0.

```java
/*
Let's first only consider the ≥ (type 1)and ≤(type 2) constraints. The integers satisfying those two constraints will be some contiguous interval [l,r]
To find [l,r] , for each ≥x constraint, we do l:=max(l,x) and for each ≤x constraint, we do r:=min(r,x).
Now, for each, ≠x (type 3) constraint, we check if x is in [l,r]. If so, we subtract one from the answer.
*/

import java.util.Scanner;
import java.util.Vector;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int t = scanner.nextInt();
        for (int w = 0; w < t; w++) {
            int n = scanner.nextInt();
            Vector<Integer> v = new Vector<>();
            int mn = 0, mx = 1000000004, ans = 0;
            for (int i = 0; i < n; i++) {
                int a = scanner.nextInt();
                int x = scanner.nextInt();
                if (a == 1) mn = Math.max(mn, x);
                else if (a == 2) mx = Math.min(mx, x);
                else v.add(x);
            }
            if (mx < mn) {
                System.out.println(0);
                continue;
            }
            int cnt = mx - mn + 1;
            for (int i = 0; i < v.size(); i++) {
                if (v.get(i) >= mn && v.get(i) <= mx) cnt--;
            }
            System.out.println(cnt);
        }
    }
}
```