PW SKILLS

Lesson Plan

Hash Maps - 2

PW SKILLS

Q1. Leetcode 2094

You are given an integer array digits, where each element is a digit. The array may contain duplicates.

You need to find all the unique integers that follow the given requirements:

The integer consists of the concatenation of three elements from digits in any arbitrary order.
The integer does not have leading zeros.
The integer is even.
For example, if the given digits were [1, 2, 3], integers 132 and 312 follow the requirements.

Return a sorted array of the unique integers.

Example 1:

Input: digits = [2,1,3,0]
Output: [102,120,130,132,210,230,302,310,312,320]

Explanation: All the possible integers that follow the requirements are in the output array.
Notice that there are no odd integers or integers with leading zeros.

Example 2:

Input: digits = [2,2,8,8,2]
Output: [222,228,282,288,822,828,882]

Explanation: The same digit can be used as many times as it appears in digits.
In this example, the digit 8 is used twice each time in 288, 828, and 882.

Example 3:

Input: digits = [3,7,5]
Output: []

Explanation: No even integers can be formed using the given digits.

Constraints:

- 3 <= digits.length <= 100
- 0 <= digits[i] <= 9

/*
As we know that we want unique numbers of 3 digits only that too only even. so first we
gather the frequency of all the digits we have, then we iterate from 100 to 999 (all possible 3 digits numbers,
100,102,104...
all possible even 3 digit numbers). for ex we are iterating and we are
at 104 so we will see that if we have digits
1,0,4 in our database if yes then we can make this number from our
available digits given to us.

Time complexity : O(digits.length) // due to making of frequency map
Space Complexity : O(1) //fixed map array space for digits 0 to 9 */

```
class Solution {
    public int[] findEvenNumbers(int[] digits) {
        int [] map = new int[10]; // for freq of 0 to 9 (digits
are fixed)

        for(int i = 0;i<digits.length;i++){ //make a frequency
map of digits
            map[digits[i]]++;
        }

        List<Integer> arr = new ArrayList<>();

        for(int i = 100;i<=999;i = i + 2){ //will always runs
from 100 to 999
            int num = i;
            int [] freq = new int[10];
            while(num > 0){  // will always run 3 times
                int rem = num % 10;
                freq[rem]++;
                num = num/10;
            }

            boolean res = findans(freq,map);
            if(res) arr.add(i);
        }

        int [] ans = new int[arr.size()]; //logic for arraylist
to array conversion
        for(int i = 0;i<arr.size();i++){ // at max we can have
all num from 100 to 998 only
            ans[i] = arr.get(i);
        }

        return ans;
    }

    private boolean findans(int [] currentNum,int [] database){

        for(int i = 0;i<10;i++){  //it will always run for at max
10 times
            if(currentNum[i] > database[i]) return false;
        }
        return true;
    }
}
```

Q2. Leetcode 1814

You are given an array nums that consists of non-negative integers. Let us define rev(x) as the reverse of the non-negative integer x. For example, rev(123) = 321, and rev(120) = 21. A pair of indices (i, j) is nice if it satisfies all of the following conditions:

0 <= i < j < nums.length
nums[i] + rev(nums[j]) == nums[j] + rev(nums[i])
Return the number of nice pairs of indices. Since that number can be too large, return it modulo 109 + 7.

Example 1:

Input: nums = [42,11,1,97]
Output: 2

Explanation: The two pairs are:
- (0,3) : 42 + rev(97) = 42 + 79 = 121, 97 + rev(42) = 97 + 24 = 121.
- (1,2) : 11 + rev(1) = 11 + 1 = 12, 1 + rev(11) = 1 + 11 = 12.

Example 2:

Input: nums = [13,10,35,24,76]
Output: 4

Constraints:
1 <= nums.length <= 1e5
0 <= nums[i] <= 1e9

```
import java.util.HashMap;
import java.util.Collection;
import java.util.Map;

class Solution {
    public int countNicePairs(int[] nums) {
        /// Create a HashMap to store occurrences of temporary
numbers
        Map<Integer, Integer> numbers = new HashMap<>();

        /// Iterate through the array
        for (int num : nums) {
            /// Calculate the temporary number
            int temporary_number = num - reverse(num);

            /// Update the count in the HashMap
            if (numbers.containsKey(temporary_number)) {
                numbers.put(temporary_number,
numbers.get(temporary_number) + 1);
            } else {
                numbers.put(temporary_number, 1);
            }
        }
```

```
        /// Calculate the total number of nice pairs
        long result = 0;
        Collection<Integer> values = numbers.values();
        int mod = 1000000007;
        for (int value : values) {
            result = (result % mod + ((long) value * ((long)
value - 1) / 2)) % mod;
        }

        // Return the result as an integer
        return (int) result;
    }

    /// Helper function to reverse a number
    private int reverse(int number) {
        int reversed_number = 0;
        while (number > 0) {
            reversed_number = reversed_number * 10 + number % 10;
            number /= 10;
        }
        return reversed_number;
    }
}
```

Q3. Top view of binary tree

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.TreeMap;

class TreeNode {
    int data;
    TreeNode left, right;

    public TreeNode(int data) {
        this.data = data;
        this.left = this.right = null;
    }
}

public class TopViewBinaryTree {
    public static void topView(TreeNode root) {
        if (root == null) {
            return;
        }

        class QueueItem {
            TreeNode node;
            int horizontalDistance;
```

```
            public QueueItem(TreeNode node, int
horizontalDistance) {
                this.node = node;
                this.horizontalDistance = horizontalDistance;
            }
        }

        TreeMap<Integer, Integer> topViewMap = new TreeMap<>();
        Queue<QueueItem> queue = new LinkedList<>();
        queue.add(new QueueItem(root, 0));

        while (!queue.isEmpty()) {
            QueueItem item = queue.poll();
            TreeNode node = item.node;
            int horizontalDistance = item.horizontalDistance;

            if (!topViewMap.containsKey(horizontalDistance)) {
                topViewMap.put(horizontalDistance, node.data);
            }

            if (node.left != null) {
                queue.add(new QueueItem(node.left,
horizontalDistance - 1));
            }
            if (node.right != null) {
                queue.add(new QueueItem(node.right,
horizontalDistance + 1));
            }
        }

        // Print the top view
        for (int value : topViewMap.values()) {
            System.out.print(value + " ");
        }
    }

    public static void main(String[] args) {
        /*
         * Example Binary Tree:
         *        1
         *       / \
         *      2   3
         *       \
         *        4
         *         \
         *          5
         */
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.right = new TreeNode(4);
        root.left.right.right = new TreeNode(5);
```

```
        System.out.println("Top view of the binary tree:");
        topView(root);
    }
}
```

Q4. Leetcode 138

A linked list of length n is given such that each node contains an additional random pointer, which could point to any node in the list, or null.

Construct a deep copy of the list. The deep copy should consist of exactly n brand new nodes, where each new node has its value set to the value of its corresponding original node. Both the next and random pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state. None of the pointers in the new list should point to nodes in the original list.

For example, if there are two nodes X and Y in the original list, where X.random --> Y, then for the corresponding two nodes x and y in the copied list, x.random --> y.

Return the head of the copied linked list.

The linked list is represented in the input/output as a list of n nodes. Each node is represented as a pair of [val, random_index] where:

val: an integer representing Node.val
random_index: the index of the node (range from 0 to n-1) that the random pointer points to, or null if it does not point to any node.
Your code will only be given the head of the original linked list.

Example 1:

Input: head = [[7,null],[13,0],[11,4],[10,2],[1,0]]
Output: [[7,null],[13,0],[11,4],[10,2],[1,0]]

Example 2:

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

Example 3:

Input: head = [[3,null],[3,0],[3,null]]
Output: [[3,null],[3,0],[3,null]]

Constraints:

0 <= n <= 1000
-104 <= Node.val <= 104
Node.random is null or is pointing to some node in the linked list.

```
public class Solution {
    public Node copyRandomList(Node head) {
        if (head == null) return null;

        HashMap<Node, Node> oldToNew = new HashMap<>();

        Node curr = head;
        while (curr != null) {
            oldToNew.put(curr, new Node(curr.val));
            curr = curr.next;
        }

        curr = head;
        while (curr != null) {
            oldToNew.get(curr).next = oldToNew.get(curr.next);
            oldToNew.get(curr).random =
oldToNew.get(curr.random);
            curr = curr.next;
        }

        return oldToNew.get(head);
    }
}
```

Q5. Leetcode 560

Given an array of integers nums and an integer k, return the total number of subarrays whose sum equals to k.

A subarray is a contiguous non-empty sequence of elements within an array.

Example 1:

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

Example 2:

Input: nums = [1,2,3], k = 3
Output: 2

Constraints:

1 <= nums.length <= 2 * 1e4
-1000 <= nums[i] <= 1000
-107 <= k <= 1e7

```
class Solution {
    public int subarraySum(int[] nums, int k) {
        int sum = 0;
        int ans = 0;
        HashMap<Integer,Integer> map = new HashMap<>();
        map.put(0,1);
        for(int j=0;j<nums.length;j++){
            sum += nums[j];
            if(map.containsKey(sum -k)){
                ans += map.get(sum-k);
            }
            map.put(sum,map.getOrDefault(sum,0)+1);
        }
        return ans;
    }
}
```

Q6. Leetcode 3

Given a string s, find the length of the longest substring without repeating characters.

Example 1:

Input: s = "abcabcbb"
Output: 3

Explanation: The answer is "abc", with the length of 3.

Example 2:

Input: s = "bbbbb"
Output: 1

Explanation: The answer is "b", with the length of 1.

Example 3:

Input: s = "pwwkew"
Output: 3

Explanation: The answer is "wke", with the length of 3.

Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Constraints:

0 <= s.length <= 5 * 104
s consists of English letters, digits, symbols and spaces.

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n = s.length();
        int maxLength = 0;
        Set<Character> charSet = new HashSet<>();
        int left = 0;

        for (int right = 0; right < n; right++) {
            if (!charSet.contains(s.charAt(right))) {
                charSet.add(s.charAt(right));
                maxLength = Math.max(maxLength, right - left +
1);
            } else {
                while (charSet.contains(s.charAt(right))) {
                    charSet.remove(s.charAt(left));
                    left++;
                }
                charSet.add(s.charAt(right));
            }
        }

        return maxLength;
    }
}
```

PW SKILLS

THANK YOU !

PW SKILLS