**CSE-E**

**Assignment Set 1**

**Name:** Ravella Abhinav

**Roll No:** CB.EN.U4CSE19453

**(**Submission Due Date => March 13th 2022 – Monday Midnight **)**

**HackerRank Problems**

1. Maximum Palindromes
2. Game of 2 Stacks
3. Append and Delete
4. Cut the Tree
5. Construct the Array

**Leetcode Problems**

1. Implement Trie (Prefix Tree)
2. Add 2 Numbers
3. Swapping Nodes in a Linked List
4. Path Sum
5. Sum of Root to Leaf Binary Numbers
6. Deepest Leaves Sum
7. Check Completeness of a Binary Tree
8. Merge Intervals
9. K-diff Pairs in an Array

**Submission Instructions**

1. You must copy your code to a Word Document. Look at the template below
2. Name the file as <Your Roll Number>\_Assignment\_Set1.doc
3. Fill the Summary Page as shown in the template below

|  |  |  |  |
| --- | --- | --- | --- |
| **Problem** | **Number of Test Cases passed** | **Number of Test Cases failed** | **Total Number of Test Cases** |
| Maximum Palindromes | - | - | - |
| Game of 2 Stacks | 13 | 0 | 13 |
| Append and Delete | 13 | 0 | 13 |
| Cut the Tree | - | - | - |
| Construct the Array | 20 | 0 | 20 |
| Implement Trie (Prefix Tree) | - | - | - |
| Add 2 Numbers | ALL | 0 | ALL |
| Swapping Nodes in a Linked List | ALL | 0 | ALL |
| Path Sum | ALL | 0 | ALL |
| Sum of Root to Leaf Binary Numbers | ALL | 0 | ALL |
| Deepest Leaves Sum |  |  |  |
| Check Completeness of a Binary Tree | ALL | 0 | ALL |
| Merge Intervals | ALL | 0 | ALL |
| K-diff Pairs in an Array | ALL | 0 | ALL |

**Problem Number 2 =>** Game of Two Stacks

**import** math

**import** os

**import** random

**import** re

**import** sys

**import** sys

g = **int**(**input**().strip())

**for** a0 **in** **range**(g):

    n,m,x = **input**().strip().split(' ')

    n,m,x = [**int**(n),**int**(m),**int**(x)]

    a = **list**(**map**(**int**, **input**().strip().split(' ')))

    b = **list**(**map**(**int**, **input**().strip().split(' ')))

    i = 0

**while** i < **len**(a) **and** x >= a[i]:

        x = x - a[i]

        i = i + 1

    ans = i

    j = 0

**for** p **in** b:

        j = j + 1

        x = x - p

**while** x < 0 **and** i > 0:

            i = i - 1

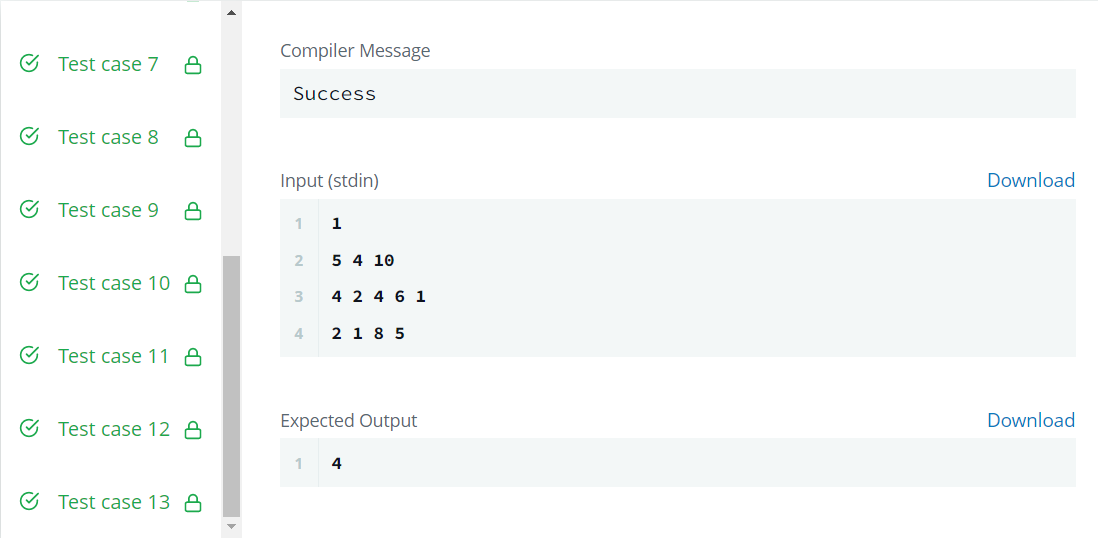
            x = x + a[i]

**if** x >= 0:

            ans = **max**(ans, j + i)

**print**(ans)

**Output:**



**Problem Number 3 =>** Append and Delete

**def** appendAndDelete(s, t, k):

*# Write your code here*

    assume=0

**for** (cs, ct) **in** **zip**(s, t):

**if** cs==ct:

            assume = assume + 1

**else**:

**break**

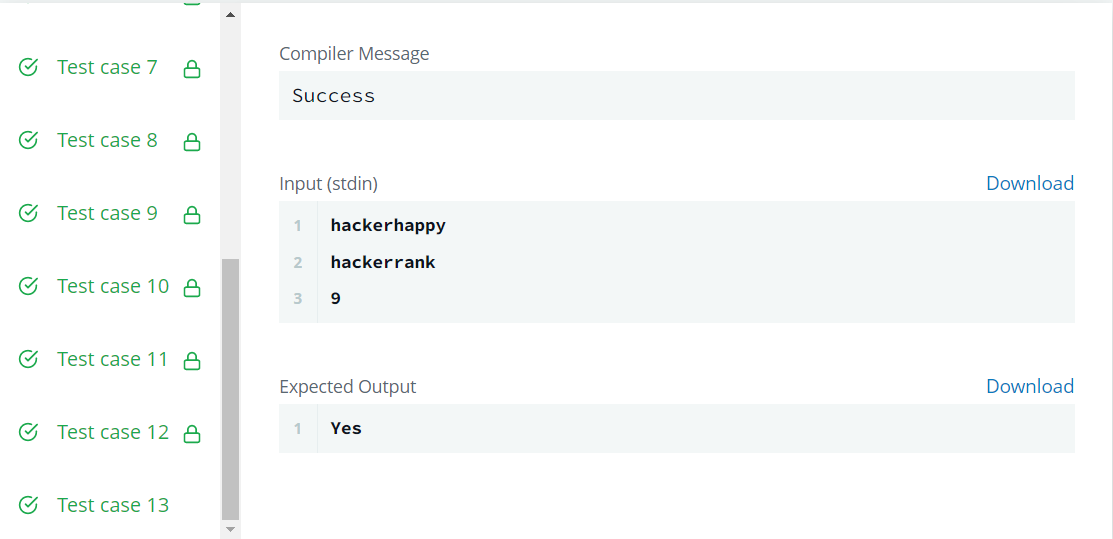
    sz = **len**(s) + **len**(t)

**if** (2\*assume+k >= sz **and** sz%2==k%2) **or** sz<k:

**return** "Yes"

**return** "No"

**Output:**



**Problem Number 5 =>** Construct the Array

**import** sys

**def** countArray(n, k, x):

    d, s = 1, 0

**for** i **in** **range**(2, n):

        d, s = (k - 2) \* d + s, (k - 1) \* d

**return** d **if** x != 1 **else** s

**if** \_\_name\_\_ == "\_\_main\_\_":

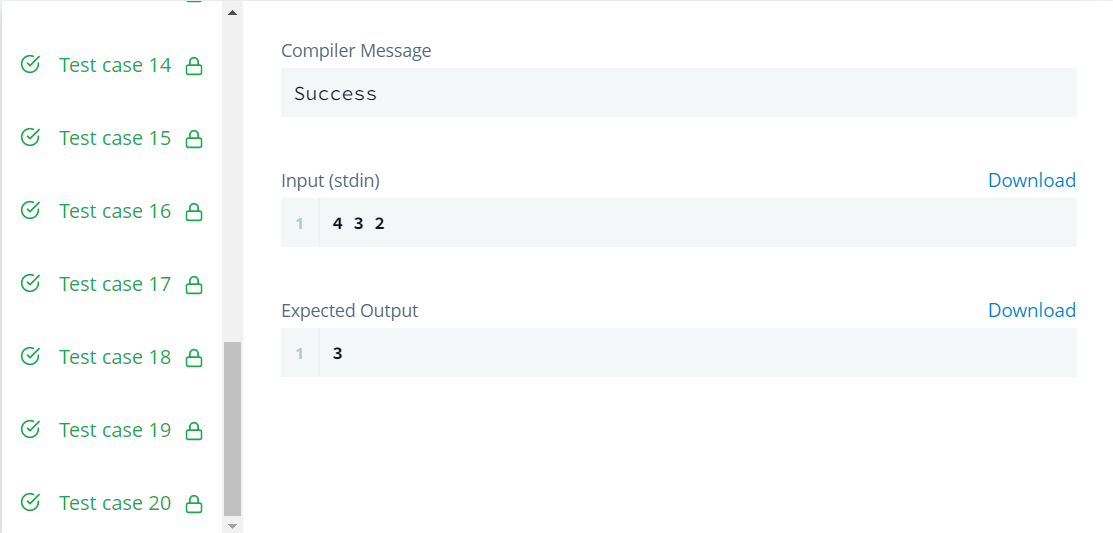
    n, k, x = **input**().strip().split(' ')

    n, k, x = [**int**(n), **int**(k), **int**(x)]

    answer = countArray(n, k, x)

**print**(answer % (10 \*\* 9 + 7))

**Output:**

****

**Problem Number 7 =>** Add Two Numbers

class Solution:

    def addTwoNumbers(self, l1, l2, carry=0):

        if not l1 and not l2 and not carry:

return None

        csum = carry

        if l1:

            csum += l1.val

            l1=l1.next

        if l2:

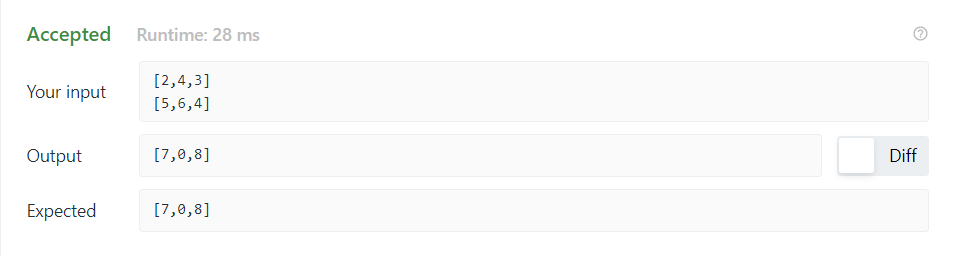
            csum+=l2.val

            l2=l2.next

        head = ListNode(csum%10, self.addTwoNumbers(l1,l2,csum>9))

        return head

**Output:**

****

**Problem Number 8 =>** Swapping Nodes in a Linked List

class Solution:

    def isValid(self, s: str) -> bool:

        stack, hm = [], {'(': ')', '{': '}', '[': ']'}

        for ch in s:

            if ch in hm:

                stack.append(ch)

            else:

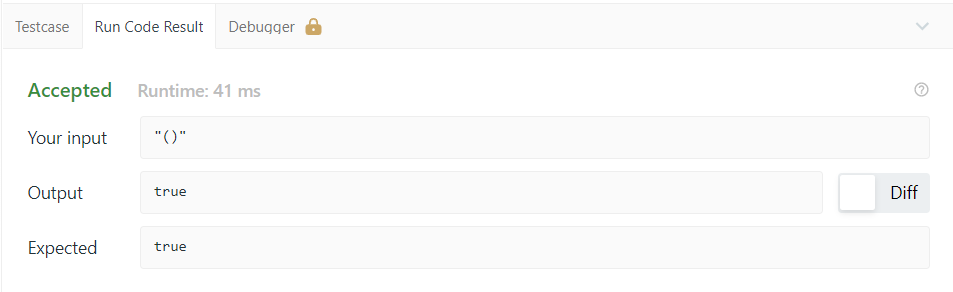
                if not stack or hm[stack[-1]] != ch:

                    return False

                stack.pop()

        return not stack

**Output:**

****

**Problem Number 9 =>** Path Sum

class Solution:

    def hasPathSum(self, root: Optional[TreeNode], targetSum: int) -> bool:

        if root is None: return False

        stack = deque()

        currSum = 0

        while stack or root:

            while root:

                currSum += root.val

                stack.append((root, currSum))

                root = root.left

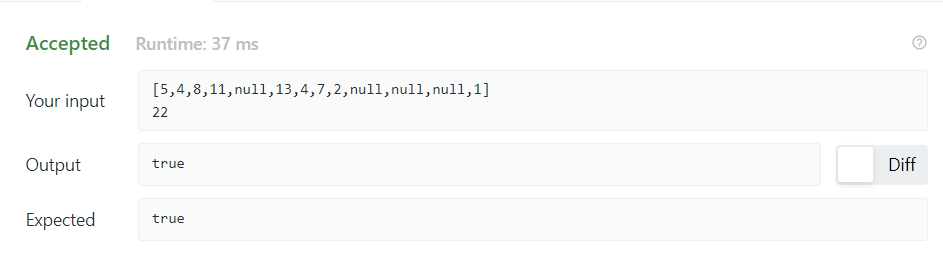
            root, currSum = stack.pop()

            if root.left is None and root.right is None and currSum == targetSum: return True

            root = root.right

        return False

**Output:**

****

**Problem Number 10 =>** Sum of Root to Leaf Binary Numbers

class Solution:

    def sumRootToLeaf(self, root: Optional[TreeNode]) -> int:

        self.ans = list()

        self.dfs(root, '')

        return sum(self.ans)

    def dfs(self, node, path):

        if not node:

            return

        path += str(node.val)

        self.dfs(node.left, path)

        self.dfs(node.right, path)

        if not node.left and not node.right:

            self.ans.append(int(path, 2))

**Output:**

****

**Problem Number 12 =>**Check Completeness of a Binary Tree

class Solution:

    def isCompleteTree(self, root: Optional[TreeNode]) -> bool:

        queue = [root]

        end = False

        while queue:

            node = queue.pop(0)

            if not node:

                end = True

            else:

                if end:

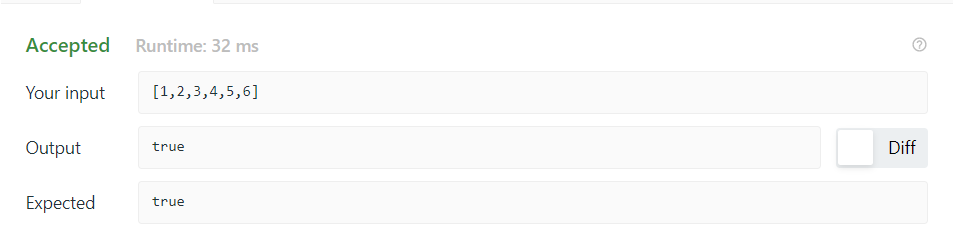
                    return False

                queue.append(node.left)

                queue.append(node.right)

        return True

**Output:**

****

**Problem Number 13 =>**Merge Intervals

class Solution:

    def merge(self, intervals: List[List[int]]) -> List[List[int]]:

        answer = []

        intervals = sorted(intervals)

        start, end = None, None

        for interval in intervals:

            if start is None:

                start, end = interval

            elif interval[0] <= end <= interval[1]:

                end = interval[1]

            elif end < interval[0]:

                answer.append((start, end))

                start, end = interval

        answer.append((start, end))

        return answer

**Output:**

****

**Problem Number 14 =>**K-diff Pairs in an Array

class Solution:

    def findPairs(self, nums: List[int], k: int) -> int:

        nums = sorted(nums)

        numdict = dict()

        res = set()

        for n in nums :

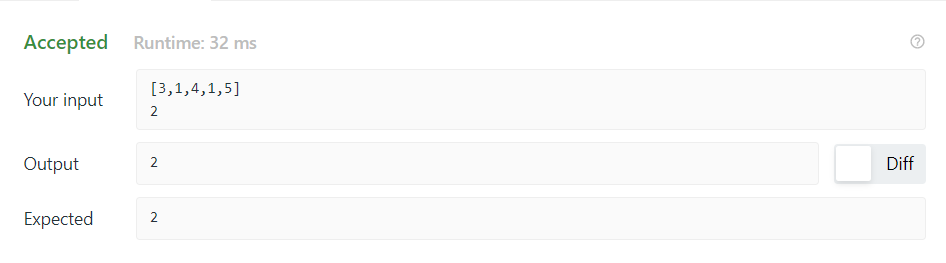
            if n in numdict :

                res.add((n,numdict[n]))

            numdict[k+n]=n

        return len(res)

**Output:**

****