|  |
| --- |
| Write a function which compress string AAACCCBBD to A3C3B2D  and other function to generate from the compressed. |
| Given the root of a binary tree. Check whether it is a BST or not. **Note:**We are considering that BSTs cannot contain duplicate Nodes. A **BST** is defined as follows:   * The left subtree of a node contains only nodes with keys **less than** the node's key. * The right subtree of a node contains only nodes with keys **greater than** the node's key. * Both the left and right subtrees must also be binary search trees.     **Example 1:**  **Input:**     2  /    \  1      3  **Output:** 1  **Explanation:**  The left subtree of root node contains node  with key lesser than the root nodes key and  the right subtree of root node contains node  with key greater than the root nodes key.  Hence, the tree is a BST.  **Example 2:**  **Input:**  2    \    7    \    6    \    5    \    9    \    2    \    6  **Output:** 0  **Explanation:**  Since the node with value 7 has right subtree  nodes with keys less than 7, this is not a BST.  **Expected Time Complexity:** O(N). **Expected Auxiliary Space:** O(Height of the BST).  **Constraints:** 0 <= Number of edges <= 100000 |
| Given a matrix **mat**[][] of size **N** x **M**, where every row and column is sorted in increasing order, and a number **X** is given. The task is to find whether element **X** is present in the matrix or not.  **Example 1:**  **Input**:  N = 3, M = 3  mat[][] = 3 30 38  44 52 54  57 60 69  X = 62  **Output**  0  **Explanation**:  62 is not present in the  matrix, so output is 0  **Example 2:**  **Input**:  N = 1, M = 6  mat[][]= 18 21 27 38 55 67  X = 55  **Output**:  1  **Explanation**:  55 is present in the  matrix at 5th cell.  **Expected Time Complexity**: O(N+M). **Expected Auxiliary Space**: O(1).  **Constraints**: 1 <= N, M <= 1005 1 <= mat[][] <= 10000000 1<= X <= 10000000 |
| Given a string **S**, find the length of the longest substring without repeating characters.  **Example 1:**  **Input:**  S = "geeksforgeeks"  **Output:**  7  **Explanation:**  Longest substring is  "eksforg".  **Example 2:**  **Input:**  S = "abdefgabef"  **Output:**  6  **Explanation:**  Longest substring are  "abdefg" , "bdefga" and "defgab".  **Expected Time Complexity:**O(|S|). **Expected Auxiliary Space:**O(K) where K is constant  **Constraints:** 1 ≤ |S| ≤ 105  It is guaranteed that all characters of the String S will be lowercase letters from **'a'** to **'z'** |
| Given a sorted and rotated array A of N distinct elements which is rotated at some point, and given an element key. The task is to find the index of the given element key in the array A.  **Example 1:**  **Input:**  N = 9  A[] = {5, 6, 7, 8, 9, 10, 1, 2, 3}  key = 10  **Output**:  5  **Explanation**: 10 is found at index 5.  **Example 2**:  **Input**:  N = 4  A[] = {3, 5, 1, 2}  key = 6  **Output**:  -1  **Explanation**: There is no element that has value 6.  Can you solve it in expected time complexity?  **Expected Time Complexity**: O(log N). **Expected Auxiliary Space**: O(1).  **Constraints**: 1 ≤ N ≤ 107 0 ≤ A[i] ≤ 108 1 ≤ key ≤ 108 |
| Suppose there is a circle. There are **N** petrol pumps on that circle. You will be given two sets of data. **1.** The amount of petrol that every petrol pump has. **2.** Distance from that petrol pump to the next petrol pump. Find a starting point where the truck can start to get through the complete circle without exhausting its petrol in between. **Note :**  Assume for 1 litre petrol, the truck can go 1 unit of distance.  **Example 1:**  **Input:**  N = 4  Petrol = 4 6 7 4  Distance = 6 5 3 5  **Output:** 1  **Explanation: T**here are 4 petrol pumps with  amount of petrol and distance to next  petrol pump value pairs as {4, 6}, {6, 5},  {7, 3} and {4, 5}. The first point from  where truck can make a circular tour is  2nd petrol pump. Output in this case is 1  (index of 2nd petrol pump).  **Expected Time Complexity:**O(N) **Expected Auxiliary Space**: O(1)  **Constraints:** 2 ≤ N ≤ 10000 1 ≤ petrol, distance ≤ 1000 |
| Given a 2D matrix M of dimensions RxC. Find the maximum sum submatrix in it.  **Example 1:**  **Input:**  R=4  C=5  M=[[1,2,-1,-4,-20],  [-8,-3,4,2,1],  [3,8,10,1,3],  [-4,-1,1,7,-6]]  **Output:**  29  **Explanation:**  The matrix is as follows and the  blue rectangle denotes the maximum sum  rectangle.  Thumbnail  **Example 2:**  **Input:**  R=2  C=2  M=[[-1,-2],[-3,-4]]  **Output:**  -1  **Explanation:**  Taking only the first cell is the  optimal choice.  **Expected Time Complexity:**O(R\*R\*C) **Expected Auxillary Space:**O(R\*C)  **Constraints:** 1<=R,C<=500 -1000<=M[i][j]<=1000 |
| Given a Binary Tree (BT), convert it to a Doubly Linked List(DLL) In-Place. The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be same as Inorder of the given Binary Tree. The first node of Inorder traversal (leftmost node in BT) must be the head node of the DLL.  TreeToList  **Example 1:**  **Input:**        1     /  \    3    2  **Output:**  3 1 2  2 1 3  **Explanation:** DLL would be 3<=>1<=>2  **Example 2:**  **Input:**         10       /   \  20   30    /   \   40   60  **Output:**  40 20 60 10 30  30 10 60 20 40  **Explanation:**  DLL would be  40<=>20<=>60<=>10<=>30.  **Expected Time Complexity:**O(N). **Expected Auxiliary Space:**O(H). **Note:**H is the height of the tree and this space is used implicitly for the recursion stack.  **Constraints:** 1 ≤ Number of nodes ≤ 105 0 ≤ Data of a node ≤ 105 |
| 1. Given a Binary Tree, check if all leaves are at same level or not.   **Example 1:**  **Input:**  1  / \  2 3  **Output:** 1  **Explanation:**  Leaves 2 and 3 are at same level.  **Example 2:**  **Input:**  10  / \  20 30  / \  10 15  **Output:** 0  **Explanation:**  Leaves 10, 15 and 30 are not at same level.  **Expected Time Complexity:**O(N) **Expected Auxiliary Space:** O(height of tree)  **Constraints:** 1 ≤ N ≤ 10^3 |
| Given two strings **s** and **t.**Return the minimum number of operations required to convert **s**to **t**. The possible operations are permitted:   1. Insert a character at any position of the string. 2. Remove any character from the string. 3. Replace any character from the string with any other character.     **Example 1:**  **Input:**  s = "geek", t = "gesek"  **Output:** 1  **Explanation:** One operation is required  inserting 's' between two 'e's of str1.  **Example 2:**  **Input :**  s = "gfg", t = "gfg"  **Output:**  0  **Explanation:** Both strings are same.  **Expected Time Complexity:**O(|s|\*|t|) **Expected Space Complexity:**O(|s|\*|t|)  **Constraints:** 1 ≤ Length of both strings ≤ 100 Both the strings are in lowercase. |

.