Leet Code

May Coding Challenge

E-Mail	sunil016@yahoo.com
HackerRank	https://www.hackerrank.com/atworksunil
GitHub	https://github.com/Ysunil016
Linkedin	https://www.linkedin.com/in/sunil016/

May - Week 1

First Bad Version

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have `n` versions `[1, 2, ..., n]` and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API `bool isBadVersion(version)` which will return whether `version` is bad. Implement a function to find the first bad version. You should minimise the number of calls to the API.

Given n = 5, and version = 4 is the first bad version.

call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true

Then 4 is the first bad version.

Coding

```
public static int recursiveA(int left, int right) {
        if (left == right)
            return left;

        int mid = left + (right - left) / 2;
        if (getV(mid)) {
            return recursiveA(left, mid);
        } else {
            return recursiveA(mid + 1, right);
        }

    }
```

Jewels and Stones

You're given strings `J` representing the types of stones that are jewels, and `S` representing the stones you have. Each character in `S` is a type of stone you have. You want to know how many of the stones you have are also jewels.

The letters in `J` are guaranteed distinct, and all characters in `J` and `S` are letters. Letters are case sensitive, so `"a"` is considered a different type of stone from `"A"`.

Input: J = "aA", S = "aAAbbbb"

Output: 3

Coding

```
public static int numJewelsInStones(String J, String S) {
    if (J == null || S == null || J.length() == 0 || S.length() == 0)
        return 0;

    int jCount = 0;

    for (int i=0;i<S.length();i++) {
        for (int j=0;j<J.length();j++) {
            if (S.charAt(i) == J.charAt(j)) {
                jCount++;
                break;
            }
        }
    }
    return jCount;
}
```

Ransom Note

Given an arbitrary ransom note string and another string containing letters from all the magazines, write a function that will return true if the ransom note can be constructed from the magazines ; otherwise, it will return false.

Each letter in the magazine string can only be used once in your ransom note.

Note:
You may assume that both strings contain only lowercase letters.

canConstruct("a", "b") -> false
canConstruct("aa", "ab") -> false
canConstruct("aa", "aab") -> true

Coding

```
 public static boolean canConstructO(String ransom, String
magazine) {
        int[] pool = new int[26];
        for (char c : ransom.toCharArray())
            pool[c - 'a'] -= 1;
        for (char c : magazine.toCharArray())
            pool[c - 'a'] += 1;

        for (int i = 0; i < 26; i++)
            if (pool[i] < 0)
                return false;
        return true;
    }
```

Number Complement

Given a positive integer, output its complement number. The complement strategy is to flip the bits of its binary representation.

Example 1:

```
Input: 5
Output: 2
Explanation: The binary representation of 5 is 101 (no leading zero bits), and its complement is 010. So you need to output 2.
```

Coding

```
static int getComplement(int number) {
    // Finding all the Bits with which Operation is Needed
    int bitsNeeded = (int) (Math.log(number) / Math.log(2)) + 1;
    int bit = 1;
    for (int i = 0; i < bitsNeeded; i++) {
        number = number ^ (bit);
        bit = bit << 1;
    }
    return number;
}
```

First Unique Character in a String

Given a string, find the first non-repeating character in it and return it's index. If it doesn't exist, return -1.

Examples:

```
s = "leetcode"
return 0.

s = "loveleetcode",
return 2.
```

Coding

```java
public static int firstUniqChar(String s) {
    if(s==null || s.length()==0)
        return -1;

    int[] arr = new int[26];
    for(char x:s.toCharArray()){
        arr[x-'a'] += 1;
    }
    for(int i=0;i<s.length();i++){
        if(arr[s.charAt(i)-'a']==1)
            return i;
    }
    return -1;
}
```

Majority Element

Given an array of size *n*, find the majority element. The majority element is the element that appears **more than** `⌊ n/2 ⌋` times.

You may assume that the array is non-empty and the majority element always exist in the array.

Example 1:

```
Input: [3,2,3]
Output: 3
```

Example 2:

```
Input: [2,2,1,1,1,2,2]
Output: 2
```

Coding

```java
// Java

// O(nlogn)
    public static int majorityElement_(int[] nums) {
        Arrays.sort(nums);
        return nums[nums.length/2];
    }
```

```golang
// Golang

func main() {
    X := []int{2, 2, 1, 1, 1, 2, 2}
    sort.Ints(X)
    fmt.Println(X[len(X)/2])
}
```

Cousins in Binary Tree

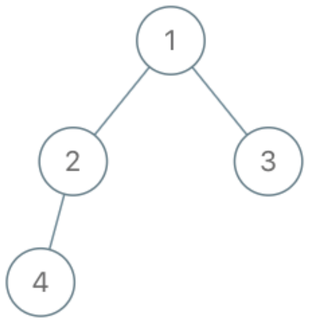
In a binary tree, the root node is at depth `0`, and children of each depth `k` node are at depth `k+1`.

Two nodes of a binary tree are *cousins* if they have the same depth, but have **different parents**.

We are given the `root` of a binary tree with unique values, and the values `x` and `y` of two different nodes in the tree.

Return `true` if and only if the nodes corresponding to the values `x` and `y` are cousins.

Example 1:




```
Input: root = [1,2,3,4], x = 4, y = 3
Output: false
```

Coding

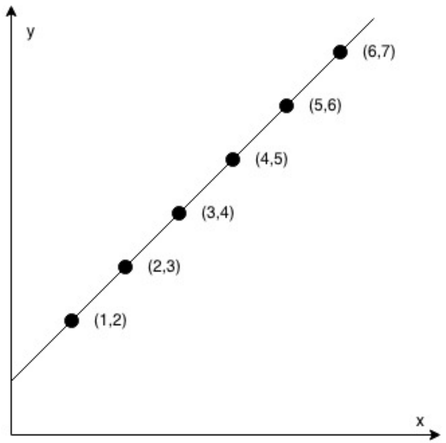
```
static boolean isCousins(TreeNode root, int x, int y) {
        if (root == null)
            return false;
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        class Info {
            TreeNode node = null;
            TreeNode parent = null;
            Info() {}
            void setInfo(TreeNode node, TreeNode parent) {
                this.node = node;
                this.parent = parent;
            }
        }
        queue.add(root);
        while (true) {
            int size = queue.size();
            if (size == 0)
                break;
            Info XXX = new Info();
            Info YYY = new Info();
            while (size > 0) {
                TreeNode n = queue.poll();
                if (n.left != null) {
                    queue.add(n.left);
                    if (n.left.val == x) {
                        XXX.setInfo(n.left, n);
                    }
                    if (n.left.val == y) {
                        YYY.setInfo(n.left, n);
                    }
                }
                if (n.right != null) {
                    queue.add(n.right);
                    if (n.right.val == x) {
                        XXX.setInfo(n.right, n);
                    }
                    if (n.right.val == y) {
                        YYY.setInfo(n.right, n);
                    }
                }
                size--;
            }
            if (XXX.node != null && YYY.node != null) {
                if (XXX.parent != YYY.parent)
                    return true;
                else
                    return false;
            }
        }
        return false;
    }
```

May - Week 2

Check If It Is a Straight Line

You are given an array `coordinates`, `coordinates[i] = [x, y]`, where `[x, y]` represents the coordinate of a point. Check if these points make a straight line in the XY plane.

Example 1:




```
Input: coordinates = [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]
Output: true
```

Coding

```
public static boolean checkStraightLine(int[][] coordinates) {
        if(coordinates==null || coordinates.length==0 || coordinates.length ==1)
            return false;

            int X1 = coordinates[0][0];
            int Y1 = coordinates[0][1];
            int X2 = coordinates[1][0];
            int Y2 = coordinates[1][1];

            float primarySlope = Float.MAX_VALUE;
            if(X2-X1!=0)
                primarySlope = (float)(Y2-Y1)/(float)(X2-X1);

            for(int i=0;i<coordinates.length-1;i++){

            X1 = coordinates[i][0];
            Y1 = coordinates[i][1];
            X2 = coordinates[i+1][0];
            Y2 = coordinates[i+1][1];

            int X_D = X2-X1;
            int Y_D = Y2-Y1;

            if(X_D==0){
                if(primarySlope!=Float.MAX_VALUE)
                    return false;
            }else{
                float slope = (float)Y_D/(float)X_D;
                if(slope!=primarySlope)
                    return false;
            }

            }

        return true;
    }
```

Perfect Square

Given a positive integer *num*, write a function which returns True if *num* is a perfect square else False.

Note: Do not use any built-in library function such as `sqrt`.

Example 1:

```
Input: 16
Output: true
```

Coding

```java
public static boolean isPerfectSqr(int num) {
    if (num == 1)
        return true;

    int low = 1;
    int high = num / 2;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (mid == num / mid && num % mid == 0) {
            return true;
        } else if (mid < num / mid) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return false;

}
```

Find the Town Judge

In a town, there are `N` people labelled from `1` to `N`. There is a rumor that one of these people is secretly the town judge.

If the town judge exists, then:

1. The town judge trusts nobody.
2. Everybody (except for the town judge) trusts the town judge.
3. There is exactly one person that satisfies properties 1 and 2.

You are given `trust`, an array of pairs `trust[i] = [a, b]` representing that the person labelled `a` trusts the person labelled `b`.

If the town judge exists and can be identified, return the label of the town judge. Otherwise, return `-1`.

Example 1:

```
Input: N = 2, trust = [[1,2]]
Output: 2
```

Coding

```
public static int findJudge(int N, int[][] trust) {
        int[] store = new int[N];
        if(trust.length==0)
            return 1;
        for(int i=0;i<trust.length;i++){
            int A = trust[i][0];
            int B = trust[i][1];
            // A Cannot be Judge, Checking If He is Nominated for
Judge Ambigously
                store[A-1] -= 1;
            // B Can Be, if Not Already Eliminated
                store[B-1] += 1;
        }
       int judgeLen = 0;
        int judgeIndex = -1;
        for(int i=0;i<N;i++){
            if(store[i] == N-1){
                judgeLen++;
                judgeIndex = i+1;
            }
            if(judgeLen>1)
                return -1;
        }
        return judgeIndex;
    }
```

Flood Fill

An `image` is represented by a 2-D array of integers, each integer representing the pixel value of the image (from 0 to 65535).

Given a coordinate `(sr, sc)` representing the starting pixel (row and column) of the flood fill, and a pixel value `newColor`, "flood fill" the image.

To perform a "flood fill", consider the starting pixel, plus any pixels connected 4-directionally to the starting pixel of the same color as the starting pixel, plus any pixels connected 4-directionally to those pixels (also with the same color as the starting pixel), and so on. Replace the color of all of the aforementioned pixels with the newColor.

At the end, return the modified image.

Example 1:

```
Input:
image = [[1,1,1],[1,1,0],[1,0,1]]
sr = 1, sc = 1, newColor = 2
Output: [[2,2,2],[2,2,0],[2,0,1]]
Explanation:
From the center of the image (with position (sr, sc)
= (1, 1)), all pixels connected
by a path of the same color as the starting pixel
are colored with the new color.
Note the bottom corner is not colored 2, because it
is not 4-directionally connected
to the starting pixel.
```

Coding

```
public static int[][] floodFill(int[][] image, int sr, int sc, int newColor) {
    track(image, sr, sc, newColor, image[sr][sc]);
    return image;
}


public static void track(int[][] image, int sr, int sc, int newColor,int
oldColor) {
    if(sr<0 || sc<0 || sr>=image.length || sc >= image[0].length)
        return;

    if(image[sr][sc]==newColor)
        return;


    if(image[sr][sc]!=oldColor)
        return;

    image[sr][sc] = newColor;

    track(image,sr,sc+1,newColor,oldColor);
    track(image,sr+1,sc,newColor,oldColor);
    track(image,sr,sc-1,newColor,oldColor);
    track(image,sr-1,sc,newColor,oldColor);
}
```

Single Element in a Sorted Array

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once. Find this single element that appears only once.

Example 1:

```
Input: [1,1,2,3,3,4,4,8,8]
Output: 2
```

Coding

```java
public static int singleNonDuplicate(int[] nums) {
        return keepTrack(nums, 0, nums.length - 1);
    }

    static int keepTrack(int[] nums, int start, int end) {
        if (start > end)
            return 0;

        int mid = start + (end - start) / 2;

        if (mid == 0) {
            return nums[mid];
        }
        if (mid == nums.length - 1) {
            return nums[mid];
        }

        if (nums[mid] != nums[mid - 1] && nums[mid] != nums[mid + 1])
            return nums[mid];

        if (nums[mid] != nums[mid - 1]) {
            if (mid % 2 == 0) {
                return keepTrack(nums, mid + 1, end);
            } else {
                return keepTrack(nums, start, mid);
            }

        } else {
            if (mid % 2 == 0) {
                return keepTrack(nums, start, mid);
            } else {
                return keepTrack(nums, mid + 1, end);

            }
        }

    }
```

Remove K Digit

Given a non-negative integer *num* represented as a string, remove *k* digits from the number so that the new number is the smallest possible.

Note:

- The length of *num* is less than 10002 and will be ≥ *k*.
- The given *num* does not contain any leading zero.

Example 1:

```
Input: num = "1432219", k = 3
Output: "1219"
Explanation: Remove the three digits 4, 3,
and 2 to form the new number 1219 which is
the smallest.
```

Coding

```
public static String removeKdigits(String num, int k) {
    char[] charArray = num.toCharArray();
    Stack<Character> stack = new Stack<Character>();

    for (int i = 0; i < charArray.length; i++) {
        while (k > 0 && !stack.isEmpty() && stack.peek() >
charArray[i]) {
            stack.pop();
            k--;
        }
        stack.push(charArray[i]);
        i++;
    }
    while (k > 0) {
        stack.pop();
        k--;
    }

    StringBuilder sb = new StringBuilder();
    while (!stack.isEmpty())
        sb.append(stack.pop());
    sb.reverse();
    while (sb.length() > 1 && sb.charAt(0) == '0')
        sb.deleteCharAt(0);
    return sb.toString();
}
```

Implement Trie (Prefix Tree)

Implement a trie with `insert`, `search`, and `startsWith` methods.

Example:

```
Trie trie = new Trie();

trie.insert("apple");
trie.search("apple");   // returns true
trie.search("app");     // returns false
trie.startsWith("app"); // returns true
trie.insert("app");
trie.search("app");     // returns true
```

Note:

- You may assume that all inputs are consist of lowercase letters `a-z`.
- All inputs are guaranteed to be non-empty strings.
-

Coding

```
static class Trie {
    private Trie children[];
    boolean isEndOfWord;

    public Trie() {
        children = new Trie[26];
        isEndOfWord = false;
    }

    public void insert(String word) {
        Trie current = this;
        for (char x : word.toCharArray()) {
            if (current.children[x - 'a'] == null) {
                current.children[x - 'a'] = new Trie();
            }
            current = current.children[x - 'a'];
        }
        current.isEndOfWord = true;
    }

    public boolean search(String word) {
        Trie current = this;
        for (char x : word.toCharArray()) {
            current = current.children[x - 'a'];
            if (current == null) {
                return false;
            }
        }
        if (current.isEndOfWord == true)
            return true;
        return false;
    }

    public boolean startsWith(String prefix) {
        Trie current = this;
        for (char x : prefix.toCharArray()) {
            current = current.children[x - 'a'];
            if (current == null) {
                return false;
            }
        }
        return true;
    }
}
```

May - Week 3

Maximum Sum Circular Subarray

Given a **circular array C** of integers represented by `A`, find the maximum possible sum of a non-empty subarray of **C**.

Here, a *circular array* means the end of the array connects to the beginning of the array. (Formally, `C[i] = A[i]` when `0 <= i < A.length`, and `C[i+A.length] = C[i]` when `i >= 0`.)

Also, a subarray may only include each element of the fixed buffer `A` at most once. (Formally, for a subarray `C[i], C[i+1], ..., C[j]`, there does not exist `i <= k1, k2 <= j` with `k1 % A.length = k2 % A.length`.)

Example 1:

```
Input: [1,-2,3,-2]
Output: 3
Explanation: Subarray [3] has maximum sum 3
```

Example 2:

```
Input: [5,-3,5]
Output: 10
Explanation: Subarray [5,5] has maximum sum 5 + 5 = 10
```

Example 3:

```
Input: [3,-1,2,-1]
Output: 4
Explanation: Subarray [2,-1,3] has maximum sum 2 + (-1) + 3 = 4
```

Coding

https://github.com/Ysunil016/Coding_Challenges/blob/master/LeetCode/src/May/WeekThree/OddEvenLinkedList.java

```java
static int getCircularMaxVal(int[] nums) {
        int kadane_one = kadane(nums);
        int sum = 0;
        for (int i = 0; i < nums.length; i++) {
            sum += nums[i];
            nums[i] *= -1;
        }

        System.out.println(kadane_one);
        System.out.println(sum);

        System.out.println(kadane_one);
        int kadane_two = kadane(nums) + sum;
        if (kadane_two > kadane_one && kadane_two != 0) {
            return kadane_two;
        } else {
            return kadane_one;
        }
    }

    static int kadane(int[] arr) {
        int len = arr.length;
        int max_so_far = Integer.MIN_VALUE;
        int max_ending_here = 0;

        for (int i = 0; i < len; i++) {
            max_ending_here = max_ending_here + arr[i];
            if (max_so_far < max_ending_here) {
                max_so_far = max_ending_here;
            }
            if (max_ending_here < 0)
                max_ending_here = 0;
        }
        return max_so_far;
    }
```

Odd Even Linked List

Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.

You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.

Example 1:

```
Input: 1->2->3->4->5->NULL
Output: 1->3->5->2->4->NULL
```

Example 2:

```
Input: 2->1->3->5->6->4->7->NULL
Output: 2->3->6->7->1->5->4->NULL
```

Note:

- The relative order inside both the even and odd groups should remain as it was in the input.
- The first node is considered odd, the second node even and so on ...

Coding

```
static Node jumbleList(Node head) {
    if (head == null)
        return head;
    if (head.next == null)
        return head;
    Node currentNode = head;
    Node odd = new Node(-1);
    Node even = new Node(-1);
    Node oddC = odd;
    Node evenC = even;

    while (currentNode != null) {
        oddC.next = currentNode;
        oddC = oddC.next;
        if (currentNode.next != null) {
            evenC.next = currentNode.next;
            evenC = evenC.next;
        } else {
            evenC.next = null;
            break;
        }
        currentNode = currentNode.next.next;
    }

    oddC.next = even.next;
    return (odd.next);
}
```

Find All Anagrams in a String

Given a string **s** and a **non-empty** string **p**, find all the start indices of **p**'s anagrams in **s**.

Strings consists of lowercase English letters only and the length of both strings **s** and **p** will not be larger than 20,100.

The order of output does not matter.

Example 1:

Input:
`s: "cbaebabacd" p: "abc"`

Output:
`[0, 6]`

Explanation:
The substring with start index = 0 is "cba", which is an anagram of "abc".
The substring with start index = 6 is "bac", which is an anagram of "abc".

Coding

```
static List<Integer> findAnagrams(String s, String p) {
        int[] Primary = new int[26];
        for (int i = 0; i < p.length(); i++) {
            Primary[p.charAt(i) - 'a']++;
        }
        List<Integer> result = new ArrayList<Integer>();
        int[] secondary = new int[26];
        for (int i = 0; i < s.length(); i++) {
            secondary[s.charAt(i) - 'a']++;
            if (i >= p.length()) {
                secondary[s.charAt(i - p.length()) - 'a']--;
            }
            if (compare(Primary, secondary)) {
                result.add(i - p.length() + 1);
            }
        }
        return result;
    }

    static boolean compare(int[] x, int[] y) {
        for (int i = 0; i < 26; i++) {
            if (x[i] != y[i])
                return false;
        }
        return true;
    }
```

Permutation in String

Given two strings **s1** and **s2**, write a function to return true if **s2** contains the permutation of **s1**. In other words, one of the first string's permutations is the **substring** of the second string.

Example 1:

```
Input: s1 = "ab" s2 = "eidbaooo"
Output: True
Explanation: s2 contains one permutation of s1 ("ba").
```

Example 2:

```
Input:s1= "ab" s2 = "eidboaoo"
Output: False
```

Coding

```
public static boolean checkInclusion(String s1, String s2) {
    int[] prime = new int[26];
    for (int i = 0; i < s1.length(); i++) {
        prime[s1.charAt(i) - 'a']++;
    }
    int[] second = new int[26];
    for (int i = 0; i < s2.length(); i++) {
        second[s2.charAt(i) - 'a']++;
        if (i >= s1.length()) {
            second[s2.charAt(i - s1.length()) - 'a']--;
        }
        if (compare(prime, second)) {
            return true;
        }
    }
    return false;
}
 static boolean compare(int[] x, int[] y) {
    for (int i = 0; i < 26; i++) {
        if (x[i] != y[i])
            return false;
    }
    return true;
}
```

Online Stock Span

Write a class `StockSpanner` which collects daily price quotes for some stock, and returns the *span* of that stock's price for the current day.

The span of the stock's price today is defined as the maximum number of consecutive days (starting from today and going backwards) for which the price of the stock was less than or equal to today's price.

For example, if the price of a stock over the next 7 days were `[100, 80, 60, 70, 60, 75, 85]`, then the stock spans would be `[1, 1, 1, 2, 1, 4, 6]`.

Example 1:

Input: `["StockSpanner","next","next","next","next","next","next","next"], [[],[100],[80],[60],[70],[60],[75],[85]]`
Output: `[null,1,1,1,2,1,4,6]`
Explanation:
First, S = StockSpanner() is initialized. Then:
S.next(100) is called and returns 1,
S.next(80) is called and returns 1,
S.next(60) is called and returns 1,
S.next(70) is called and returns 2,
S.next(60) is called and returns 1,
S.next(75) is called and returns 4,
S.next(85) is called and returns 6.

Note that (for example) S.next(75) returned 4, because the last 4 prices
(including today's price of 75) were less than or equal to today's price.

Coding

```
class StockSpanner {

    private Stack<DailyQ> st;

    public StockSpanner() {
        st = new Stack<>();
    }

    public int next(int price) {
        int res = 1;
        while (!st.isEmpty() && st.peek().price <= price) {
            res += st.pop().spanNum;
        }
        st.push(new DailyQ(price, res));
        return res;
    }

    public class DailyQ {
        public final int price, spanNum;

        public DailyQ(int price, int spanNum) {
            this.price = price;
            this.spanNum = spanNum;
        }
    }
}
```

Kth Smallest Element in a BST

Given a binary search tree, write a function `kthSmallest` to find the **k**th smallest element in it.

Note:
You may assume k is always valid, 1 ≤ k ≤ BST's total elements.

Example 1:

```
Input: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
Output: 1
```

Example 2:

```
Input: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
Output: 3
```

Coding

```
static class TreeNode {
        int val;
        TreeNode left;
        TreeNode right;

        TreeNode(int val) {
            this.val = val;
        }
        TreeNode(int val, TreeNode left, TreeNode right) {
            this.val = val;
            this.left = left;
            this.right = right;
        }
    }
    static int Counter = 0;
    static int KthSmallest = Integer.MIN_VALUE;

    public static int kthSmallest(TreeNode root, int k) {
        traverse(root, k);
        return KthSmallest;
    }
    static void traverse(TreeNode root, int k) {
        if (root == null)
            return;
        traverse(root.left, k);
        Counter++;
        if (Counter == k) {
            KthSmallest = root.val;
        }
        traverse(root.right, k);
    }
```

Count Square Submatrices with All Ones

Given a `m * n` matrix of ones and zeros, return how many **square** submatrices have all ones.

Example 1:

```
Input: matrix =
[
  [0,1,1,1],
  [1,1,1,1],
  [0,1,1,1]
]
Output: 15
Explanation:
There are 10 squares of side 1.
There are 4 squares of side 2.
There is  1 square of side 3.
Total number of squares = 10 + 4 + 1 = 15.
```

Example 2:

```
Input: matrix =
[
  [1,0,1],
  [1,1,0],
  [1,1,0]
]
Output: 7
Explanation:
There are 6 squares of side 1.
There is 1 square of side 2.
Total number of squares = 6 + 1 = 7.
```

Coding

```java
public static int countSquares(int[][] arr) {
    int count = 0;
    for (int i = 0; i < arr[0].length; i++) {
        count += arr[0][i];
    }

    for (int i = 1; i < arr.length; i++) {
        count += arr[i][0];
    }

    for (int i = 1; i < arr.length; i++) {
        for (int j = 1; j < arr[0].length; j++) {
            if (arr[i][j] != 0) {
                arr[i][j] = Math.min(arr[i - 1][j - 1], Math.min(arr[i][j - 1], arr[i - 1][j])) + 1;
            }
            count += arr[i][j];
        }
    }
    return count;
}
```

May - Week 4

Sort Characters By Frequency

Given a string, sort it in decreasing order based on the frequency of characters.

Example 1:

```
Input:
"tree"

Output:
"eert"

Explanation:
'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. Therefore
"eetr" is also a valid answer.
```

Example 2:

```
Input:
"cccaaa"

Output:
"cccaaa"

Explanation:
Both 'c' and 'a' appear three times, so "aaaccc" is
also a valid answer.
Note that "cacaca" is incorrect, as the same
characters must be together.
```

Coding

```
public static String frequencySort(String s) {
        char[] arr = s.toCharArray();
        HashMap<Character, Integer> store = new
LinkedHashMap<>();
        for (char x : arr) {
            if (store.containsKey(x)) {
                store.put(x, store.get(x) + 1);
            } else {
                store.put(x, 1);
            }
        }
        // Sorting w.r.t Values Stored in HashMap
        List<Map.Entry<Character, Integer>> list = new
LinkedList<Map.Entry<Character, Integer>>(store.entrySet());
        Collections.sort(list, (Map.Entry<Character, Integer> arg0,
Map.Entry<Character, Integer> arg1)->
arg1.getValue().compareTo(arg0.getValue()));

        StringBuilder str = new StringBuilder();
        for (Map.Entry<Character, Integer> x : list) {
            for (int y=0;y<x.getValue();y++) {
                str.append(x.getKey());
            }
        }
        return new String(str);
    }
```

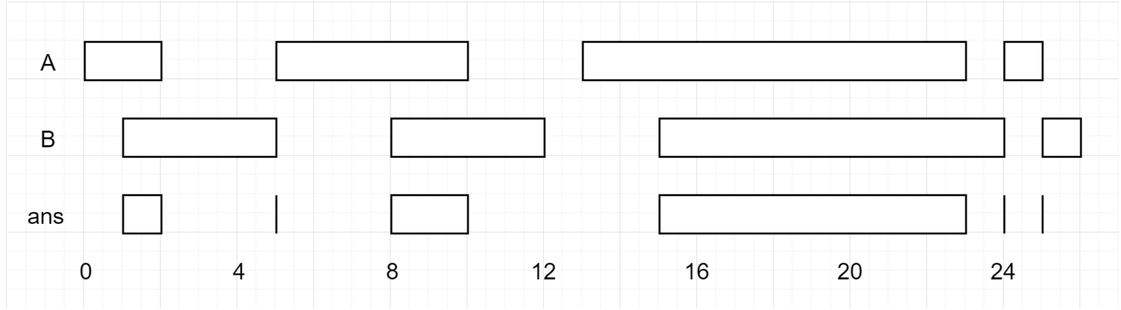
Interval List Intersections

Given two lists of **closed** intervals, each list of intervals is pairwise disjoint and in sorted order.

Return the intersection of these two interval lists.

(Formally, a closed interval `[a, b]` (with `a <= b`) denotes the set of real numbers `x` with `a <= x <= b`. The intersection of two closed intervals is a set of real numbers that is either empty, or can be represented as a closed interval. For example, the intersection of [1, 3] and [2, 4] is [2, 3].)

Example 1:




```
Input: A = [[0,2],[5,10],[13,23],[24,25]], B = [[1,5],[8,12],
[15,24],[25,26]]
Output: [[1,2],[5,5],[8,10],[15,23],[24,24],[25,25]]
Reminder: The inputs and the desired output are lists of
Interval objects, and not arrays or lists.
```

Note:

- `1.0 <= A.length < 1000`
- `2.0 <= B.length < 1000`
- `3.0 <= A[i].start, A[i].end, B[i].start, B[i].end < 10^9`

NOTE: input types have been changed on April 15, 2019. Please reset to default code definition to get new method signature.

Coding

```
static int[][] findInterPoint(int[][] x, int[][] y) {
    List<int[]> list = new ArrayList<int[]>();
    int CounterA = 0;
    int CounterB = 0;
    while (CounterA < x.length && CounterB < y.length) {
        int[] tR = { Math.max(x[CounterA][0], y[CounterB][0]),
Math.min(x[CounterA][1], y[CounterB][1]) };
        if (tR[0] <= tR[1]) {
            list.add(tR);
        }
        if (x[CounterA][1] < y[CounterB][1]) {
            CounterA++;
        } else {
            CounterB++;
        }
    }
    int[][] result = new int[list.size()][0];
    int i = 0;
    for (int[] val : list) {
        result[i] = val;
        i++;
    }
    return result;
}
```

Construct Binary Search Tree from Preorder Traversal

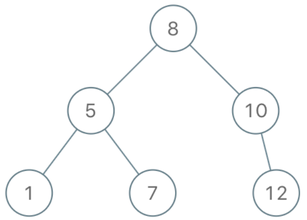
Return the root node of a binary **search** tree that matches the given `preorder` traversal.

(Recall that a binary search tree is a binary tree where for every `node`, any descendant of `node.left` has a value `< node.val`, and any descendant of `node.right` has a value `> node.val`. Also recall that a preorder traversal displays the value of the `node` first, then traverses `node.left`, then traverses `node.right`.)

It's guaranteed that for the given test cases there is always possible to find a binary search tree with the given requirements.

Example 1:

```
Input: [8,5,1,7,10,12]
Output: [8,5,10,1,7,null,12]
```



Constraints:

- `1 <= preorder.length <= 100`
- `1 <= preorder[i] <= 10^8`
- The values of `preorder` are distinct.

Coding

```
class Solution {
    TreeNode root;

    public TreeNode bstFromPreorder(int[] preorder) {
        root = new TreeNode(preorder[0]);
        for (int i = 1; i < preorder.length; i++) {
            InsertDataInBST(root, preorder[i]);
        }
        return root;
    }

    public void InsertDataInBST(TreeNode root, int data) {
        if (root == null)
            return;
        if (data < root.val) {
            if (root.left == null)
                root.left = new TreeNode(data);
            else
                InsertDataInBST(root.left, data);
        }
        if (data > root.val) {
            if (root.right == null)
                root.right = new TreeNode(data);
            else
                InsertDataInBST(root.right, data);
        }
    }
}
```

Uncrossed Lines

We write the integers of `A` and `B` (in the order they are given) on two separate horizontal lines.

Now, we may draw *connecting lines*: a straight line connecting two numbers `A[i]` and `B[j]` such that:

- `A[i] == B[j]`;
- The line we draw does not intersect any other connecting (non-horizontal) line.

Note that a connecting lines cannot intersect even at the endpoints: each number can only belong to one connecting line.

Return the maximum number of connecting lines we can draw in this way.

Example 1:

```
Input: A = [1,4,2], B = [1,2,4]
Output: 2
Explanation: We can draw 2 uncrossed lines as in the diagram.
We cannot draw 3 uncrossed lines, because the line from A[1]=4 to B[2]=4 will intersect the line from A[2]=2 to B[1]=2.
```

Example 2:

```
Input: A = [2,5,1,2,5], B = [10,5,2,1,5,2]
Output: 3
```

Coding

```
public static int maxUncrossedLines(int[] A, int[] B) {
        int[][] ARRAY_DY = new int[A.length][B.length];
        for (int i = 0; i < A.length; i++) {
            for (int j = 0; j < B.length; j++) {
                if (A[i] == B[j]) {
                    if (i > 0 && j > 0) {
                        ARRAY_DY[i][j] = ARRAY_DY[i - 1][j - 1] + 1;
                    } else {
                        ARRAY_DY[i][j] = 1;
                    }
                } else {
                    if (i > 0 && j > 0) {
                        ARRAY_DY[i][j] = Math.max(ARRAY_DY[i][j - 1],
ARRAY_DY[i - 1][j]);
                    } else {
                        if (i == 0 && j != 0)
                            ARRAY_DY[i][j] = ARRAY_DY[i][j - 1];
                        else if (i != 0 && j == 0)
                            ARRAY_DY[i][j] = ARRAY_DY[i - 1][j];
                    }
                }

            }
        }
        return ARRAY_DY[A.length - 1][B.length - 1];
    }
```

Contiguous Array

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

Example 1:

```
Input: [0,1]
Output: 2
Explanation: [0, 1] is the longest contiguous subarray with equal number of 0 and 1.
```

Example 2:

```
Input: [0,1,0]
Output: 2
Explanation: [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0 and 1.
```

Note: The length of the given binary array will not exceed 50,000.

Coding

```
public static int findMaxLength(int[] nums) {
    int fResult = 0;
    int counter = 0;
    HashMap<Integer, Integer> store = new HashMap<Integer, Integer>();
    for (int i = 0; i < nums.length; i++) {
        if (nums[i] == 0) {
            counter -= 1;
        } else {
            counter += 1;
        }
        if (counter == 0) {
            fResult = i + 1;
        }
        if (store.containsKey(counter)) {
            fResult = Math.max(fResult, i - store.get(counter));
        } else {
            store.put(counter, i);
        }
    }
    return fResult;
}
```

Possible Bipartition

Given a set of `N` people (numbered `1, 2, ..., N`), we would like to split everyone into two groups of **any** size.

Each person may dislike some other people, and they should not go into the same group.

Formally, if `dislikes[i] = [a, b]`, it means it is not allowed to put the people numbered `a` and `b` into the same group.

Return `true` if and only if it is possible to split everyone into two groups in this way.

Example 1:

```
Input: N = 4, dislikes = [[1,2],[1,3],[2,4]]
Output: true
Explanation: group1 [1,4], group2 [2,3]
```

Example 2:

```
Input: N = 3, dislikes = [[1,2],[1,3],[2,3]]
Output: false
```

Coding

```java
public static boolean possibleBipartition(int N, int[][] dislikes) {
    int[] visited = new int[N];
    Arrays.fill(visited, -1);
    ArrayList<Integer>[] adjArray = new ArrayList[N];
    for (int i = 0; i < N; i++) {
        adjArray[i] = new ArrayList<Integer>();
    }
    // Populating ADG List
    for (int i = 0; i < dislikes.length; i++) {
        adjArray[dislikes[i][0] - 1].add(dislikes[i][1] - 1);
        adjArray[dislikes[i][1] - 1].add(dislikes[i][0] - 1);
    }
    for (int i = 0; i < N; i++) {
        if (visited[i] == -1 && !dfs(adjArray, visited, i, 0))
            return false;
    }

    return true;
}
static boolean dfs(ArrayList<Integer>[] adjArray, int[] visited, int index, int grp) {
    if (visited[index] == -1) {
        visited[index] = grp;
    } else
        return visited[index] == grp;
    for (int X : adjArray[index]) {
        if (!dfs(adjArray, visited, X, 1 - grp))
            return false;
    }
    return true;
}
```

Counting Bits

Given a non negative integer number **num**. For every numbers **i** in the range **0 ≤ i ≤ num** calculate the number of 1's in their binary representation and return them as an array.

Example 1:

```
Input: 2
Output: [0,1,1]
```

Example 2:

```
Input: 5
Output: [0,1,1,2,1,2]
```

Follow up:

- It is very easy to come up with a solution with run time **O(n*sizeof(integer))**. But can you do it in linear time **O(n)** /possibly in a single pass?
- Space complexity should be **O(n)**.
-

Coding

```
public static int[] countBits(int nums) {
        int[] arr = new int[nums + 1];
        if (nums == 0) {
            return new int[] { 0 };
        }
        if (nums == 1) {
            return new int[] { 0, 1 };
        }
        arr[0] = 0;
        arr[1] = 1;
        // Pattern Recognized
        for (int i = 2; i <= nums; i++) {
            if (i % 2 == 0) {
                arr[i] = arr[i / 2];
            } else {
                arr[i] = arr[i / 2] + 1;
            }
        }
        return arr;
    }
```

Course Schedule

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses-1`.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: `[0,1]`

Given the total number of courses and a list of prerequisite **pairs**, is it possible for you to finish all courses?

Example 1:

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
             To take course 1 you should have
finished course 0. So it is possible.
```

Example 2:

```
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
             To take course 1 you should have
finished course 0, and to take course 0 you should
             also have finished course 1. So it is
impossible.
```

Coding

```java
public static boolean canFinish(int numCourses, int[][] prerequisites) {
        class Node {
                int inBound = 0;
                ArrayList<Integer> neighbour = new ArrayList<Integer>();

                public Node decrementInBound() {
                        this.inBound--;
                        return this;
                }

                public Node incrementInBound() {
                        this.inBound++;
                        return this;
                }
                public void addNeighbour(int val) {
                        ArrayList<Integer> neighbour = this.neighbour;
                        neighbour.add(val);
                        this.neighbour = neighbour;
                }
        }
        Node[] nodeStore = new Node[numCourses];
        for (int i = 0; i < numCourses; i++) {
                nodeStore[i] = new Node();
        }
        for (int i = 0; i < prerequisites.length; i++) {
                nodeStore[prerequisites[i][0]].addNeighbour(prerequisites[i][1]);
                nodeStore[prerequisites[i][1]].incrementInBound();
        }
        Queue<Node> queue = new LinkedList<Node>();
        int crossCount = 0;

        for (int i = 0; i < numCourses; i++) {
                if (nodeStore[i].inBound == 0) {
                        queue.add(nodeStore[i]);
                        crossCount++;
                }
        }
        while (!queue.isEmpty()) {
                Node current = queue.poll();
                for (int x : current.neighbour) {
                        nodeStore[x].decrementInBound();
                        if (nodeStore[x].inBound == 0) {
                                queue.add(nodeStore[x]);
                                crossCount++;
                        }
                }
        }
        return (crossCount == numCourses) ? true : false;
}
```

K Closest Points to Origin

We have a list of `points` on the plane. Find the `K` closest points to the origin `(0, 0)`.

(Here, the distance between two points on a plane is the Euclidean distance.)

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in.)

Example 1:

```
Input: points = [[1,3],[-2,2]], K = 1
Output: [[-2,2]]
Explanation:
The distance between (1, 3) and the origin is sqrt(10).
The distance between (-2, 2) and the origin is sqrt(8).
Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.
We only want the closest K = 1 points from the origin, so the answer is just [[-2,2]].
```

Example 2:

```
Input: points = [[3,3],[5,-1],[-2,4]], K = 2
Output: [[3,3],[-2,4]]
(The answer [[-2,4],[3,3]] would also be accepted.)
```

Coding

```java
public static int[][] kClosest(int[][] points, int K) {
        class Node implements Comparable<Node> {
            int distance;
            int[] coord = new int[2];
            Node(int[] coord) {
                this.distance = coord[0] * coord[0] + coord[1] * coord[1];
                this.coord = coord;
            }
            @Override
            public int compareTo(Node t) {
                return (this.distance > t.distance) ? 1 : (this.distance < t.distance) ? -1 : 0;
            }
        }
        ArrayList<Node> store = new ArrayList<Node>();
        for (int i = 0; i < points.length; i++) {
            store.add(new Node(points[i]));
        }
        // Sorting Object
        Collections.sort(store);

        int[][] result = new int[K][2];
        int counter = 0;
        for (Node x : store) {
            result[counter] = (x.coord);
            counter++;
            if (counter >= K) {
                break;
            }
        }

        return result;
    }
```

C

Coding