**438. Find All Anagrams in a String**

Given two strings s and p, return *an array of all the start indices of*p*'s anagrams in*s

**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".

**Solution:**

Apply the sliding window technique and when size of the map == 0 then add I to list and return it.

<https://github.com/hareramcse/Datastructure/blob/master/String/src/com/hs/leetcode/AnagramSubStringSearch.java>

**179. Largest Number**

Given a list of non-negative integers nums, arrange them such that they form the largest number and return it.

Since the result may be very large, so you need to return a string instead of an integer.

**Input:** nums = [10,2]

**Output:** "210"

Solution:

Arrays.*sort*(arr, **new** Comparator<String>() {

**public** **int** compare(String a, String b) {

**return** (b + a).compareTo(a + b);

}

});

Sort the array like above and append the sorted array data into string.

<https://github.com/hareramcse/Datastructure/blob/master/String/src/com/hs/leetcode/ArrangeGivenNoForBiggestNumber.java>

**1328. Break a Palindrome**

Given a palindromic string of lowercase English letters palindrome, replace **exactly one** character with any lowercase English letter so that the resulting string is **not** a palindrome and that it is the **lexicographically smallest** one possible.

**Input:** palindrome = "abccba"

**Output:** "aaccba"

**Explanation:** There are many ways to make "abccba" not a palindrome, such as "zbccba", "aaccba", and "abacba".

Of all the ways, "aaccba" is the lexicographically smallest.

**Sol:**

We will iterate the string from left to right till half of length and if we find other than ‘a’ then we will replace it with ‘a’ and return the string. If we replace middle element with any char still it will be a palindrome. So in this case we need to replace the last char of the string with ‘b’.

<https://github.com/hareramcse/Datastructure/blob/master/String/src/com/hs/leetcode/BreakPalindrome.java>

**1002. Find Common Characters**

Given a string array words, return *an array of all characters that show up in all strings within the*words*(including duplicates)*. You may return the answer in **any order**.

**Input:** words = ["bella","label","roller"]

**Output:** ["e","l","l"]

**Sol:**

Initialize the array with max value

int[] minFrequencies = new int[26];

Frequency for bella : [1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Once we get the char count we find out

minFrequencies[i] = Math.*min*(minFrequencies[i], charFrequencies[i]);

Frequency for label: [1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

minFrequencies[i] = Math.*min*(minFrequencies[i], charFrequencies[i]);

Frequency for roller: [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0]

minFrequencies[i] = Math.*min*(minFrequencies[i], charFrequencies[i]);

in last we will get

common char count = [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

from this count we will find out the chars.

<https://github.com/hareramcse/Datastructure/blob/master/String/src/com/hs/leetcode/CommonCharFromNString.java>

**387. First Unique Character in a String**

Given a string s, *find the first non-repeating character in it and return its index*. If it does not exist, return -1.

**Input:** s = "loveleetcode"

**Output:** 2

**Sol:**

In map put count of each char. Once we get the whole map return the index when map.get(ch) == 1.

<https://github.com/hareramcse/Datastructure/blob/master/String/src/com/hs/leetcode/FirstUniqueChar.java>

**22. Generate Parentheses**

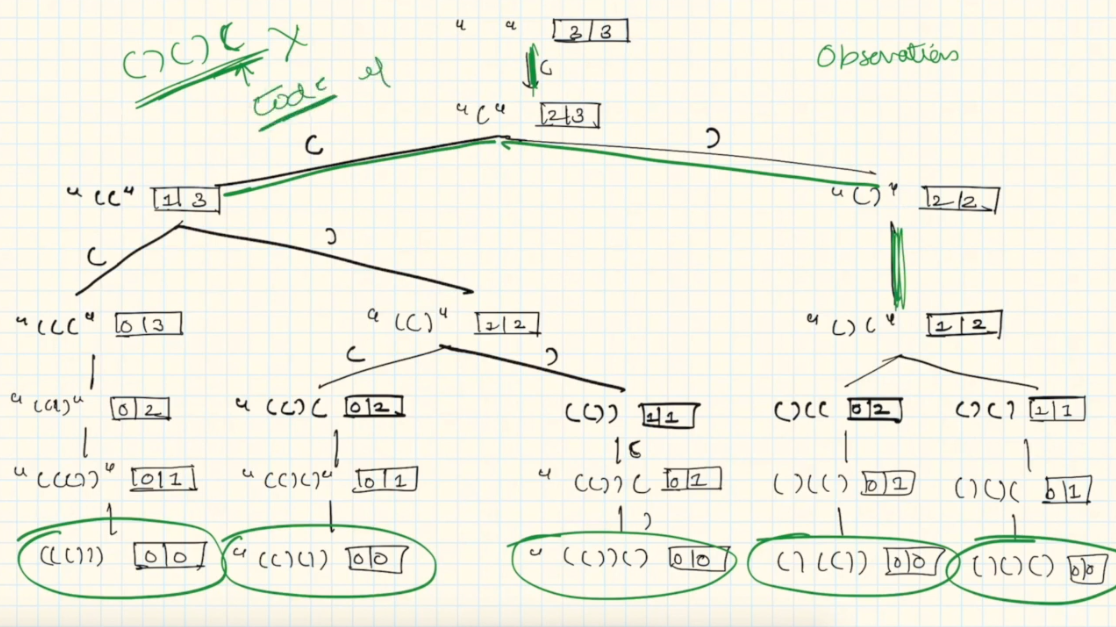
Given n pairs of parentheses, write a function to *generate all combinations of well-formed parentheses*.

**Sol:**

Here we are given n = 3;

It means we have to generate all the combination with 3 open brackets and 3 closed brackets.

Draw the recursive tree diagram with output = “”, open = 3, and closed = 3



In the diagram we can see that opening bracket is always coming. Closing bracket is coming only when closing bracket count is greater than opening bracket count. And when open == closed == 0 then we are getting the result.

<https://github.com/hareramcse/Datastructure/blob/master/String/src/com/hs/leetcode/GenerateAllBalancedParanthesis.java>

**28. Implement strStr()**

Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

**Input:** haystack = "hello", needle = "ll"

**Output:** 2

<https://github.com/hareramcse/Datastructure/blob/master/String/src/com/hs/leetcode/ImplementStrstr.java>

**12. Integer to Roman**

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

**Symbol** **Value**

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

For example, 2 is written as II in Roman numeral, just two one's added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

* I can be placed before V (5) and X (10) to make 4 and 9.
* X can be placed before L (50) and C (100) to make 40 and 90.
* C can be placed before D (500) and M (1000) to make 400 and 900.

Given an integer, convert it to a roman numeral.

<https://github.com/hareramcse/Datastructure/blob/master/String/src/com/hs/leetcode/IntegerToRoman.java>

**779. K-th Symbol in Grammar**

We build a table of n rows (**1-indexed**). We start by writing 0 in the 1st row. Now in every subsequent row, we look at the previous row and replace each occurrence of 0 with 01, and each occurrence of 1 with 10.

* For example, for n = 3, the 1st row is 0, the 2nd row is 01, and the 3rd row is 0110.

Given two integer n and k, return the kth (**1-indexed**) symbol in the nth row of a table of n rows.

**Input:** n = 2, k = 1

**Output:** 0

**Sol:**

Draw grammar with n = 4 and k = 6

**0**

1. **1**

**0 1 1 0**

**0 1 1 0 1 0 0 1**

This is the generated grammar with n = 4. Now we need to find out value at index (4, 6) that is 0

If we see closely the grammar length is increasing 2 times if we change n to n+1. At n = 1 we have 1 element in grammar, at n = 2 we have 2, at n = 3 we have 4 and at n = 4 we have 8 elements.

If we observe closely n = 3 and n = 4 we see that

Half of the 4th row is equal to 3rd row and another half of 4th row is just ^(XOR) of 1st half.

W divide the row in 2 half. If k <= mid then we can do the recursive call with n-1, k parameter. If k > mid then we just need to add the 1 ^ operator to the 1st half method calls.