**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**

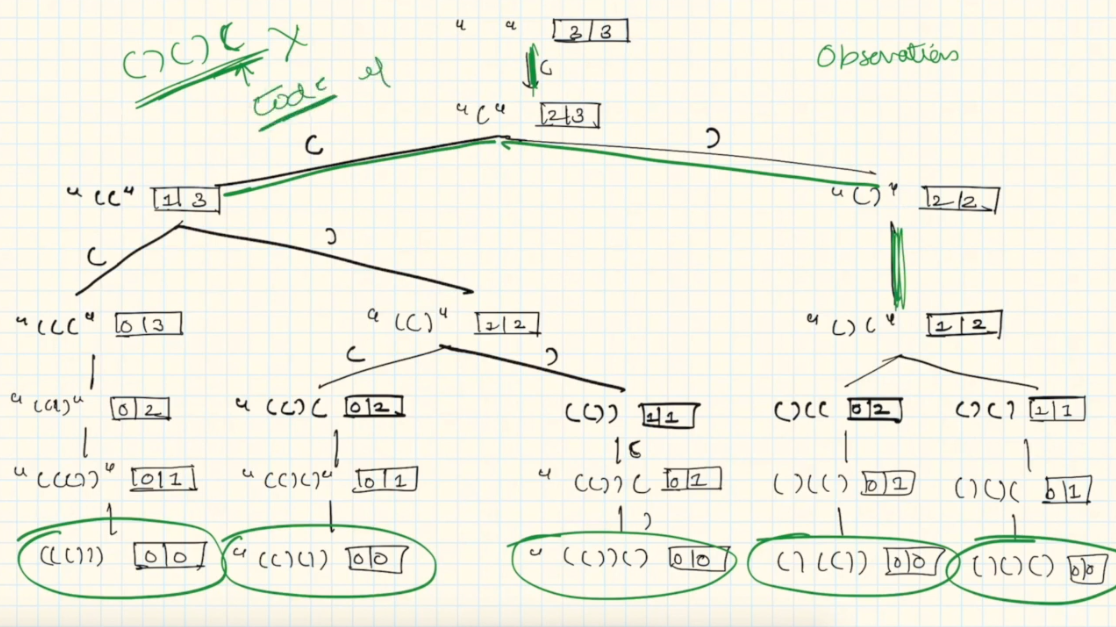
Given n pairs of parentheses, write a function to *generate all combinations of well-formed parentheses*.

**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.

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

**Print all sub sequence of a string**

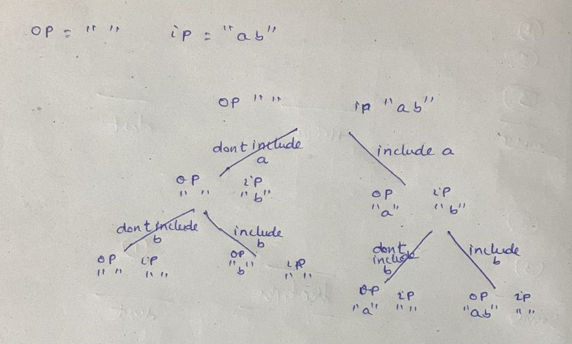
Input: “ab”

Op: “”, “b”, “a”, “ab”

**Sol:**

Take op = “” and ip=”ab” and draw the recursive tree diagram

In the diagram we can see that at leaf node when ip length == 0 we are getting result.



Take String op1 = op

String op2 = op

Op1 we are taking when we are making decision to not include the element into output and op2 we are taking for making decision to include the element.

When we decide to no include the element then op1 will be remain op1 only. but once we take the decision to include ‘a’ then op2 = op2 + ip.charAt(0); and ip = ip.subString(1)

Once we make both decisions then do the recursive call for each decision.

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

**Permutation with spaces**

**Given ip = “abc”**

Op: print all combination of abc with spaces in between letters.

**Sol:**

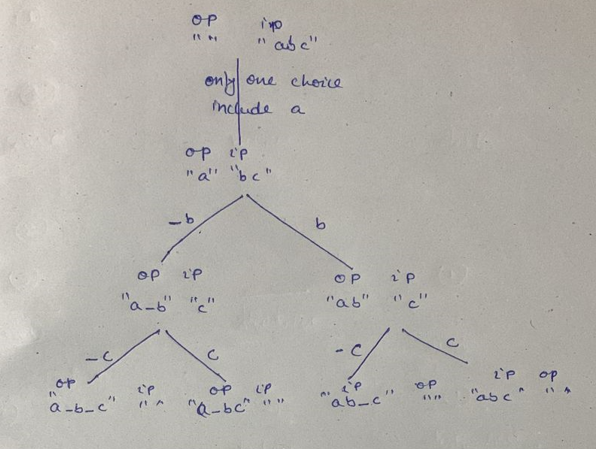
Here we are given that we cant add spaced before first char and in the last of the string. We can add the spaces in between letters.

So in the recursive choice diagram at first we have only one choice to include a in op string

Once we include ‘a’ in op string ip will be ‘bc’

Now we have 2 choices, whether we want to include the spaces or not.

From here it is like previous question.s

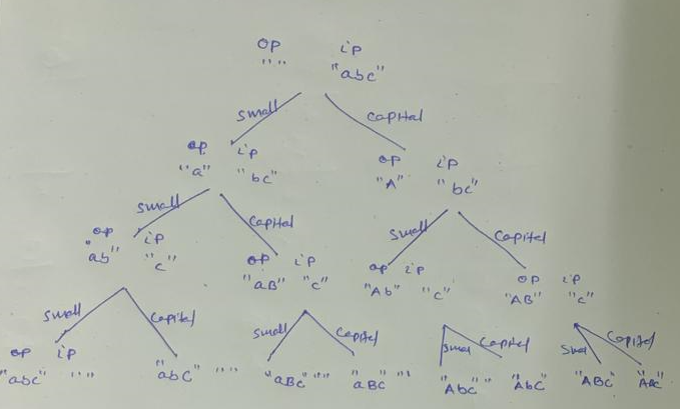


<https://github.com/hareramcse/Datastructure/blob/master/String/src/com/hs/leetcode/PermutationWithSpaces.java>

**Permutation with case change**

Given ip = “abc” we need to find out all the combination if we change the case of the letter

**Sol:**



Similiarly previous problem, here also we have 2 choices.

1st include with small case and another include with upper case

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

**784. Letter Case Permutation**

Given a string s, you can transform every letter individually to be lowercase or uppercase to create another string.

Return *a list of all possible strings we could create*. Return the output in **any order**.

**Input:** s = "a1b2"

**Output:** ["a1b2","a1B2","A1b2","A1B2"]

**Sol:**

Same technique like previous question.

We will check whether char is digit or letter. If it is letter then we will make choice for small case and capital case.

If it is digit then we will have only one choice to add the digit. Once we add the digit then that digit is processed so remove the processed char from the input.

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

**14. Longest Common Prefix**

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

**Input:** strs = ["flower","flow","flight"]

**Output:** "fl"

**Sol:**

Here we will take 1st word as starting point and compare it with next words. We will compare first char of 1st word to first char of all other words. If it does not match then we will return the str[j].subString(0, i); once both loop is over it means whole 1st word is common prefix so we will return it.

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

**409. Longest Palindrome**

Given a string s which consists of lowercase or uppercase letters, return *the length of the****longest palindrome*** that can be built with those letters.

Letters are **case sensitive**, for example, "Aa" is not considered a palindrome here.

**Input:** s = "abccccdd"

**Output:** 7

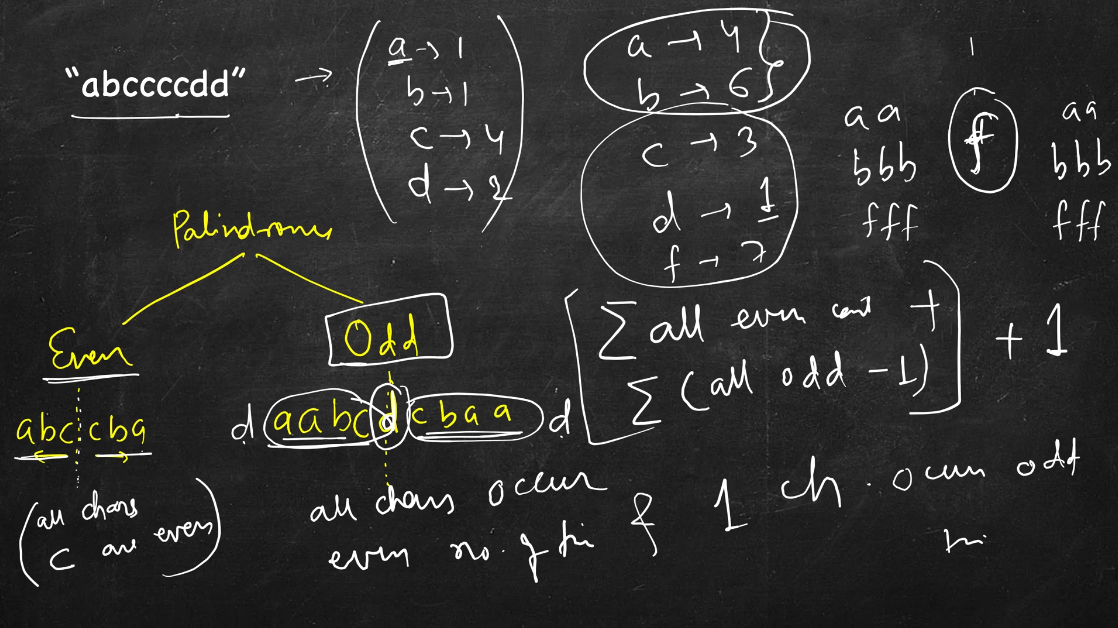
**Explanation:** One longest palindrome that can be built is "dccaccd", whose length is 7.

**Sol:**

Take the map and store the count of each char in of the input.

Palindrome can be two type: even length palindrome and odd length palindrome.

If it is even length palindrome then we can simply keep same number of char to both side. If it is odd length of char then we will make this char as middle element of the palindrome and reduce its size by 1 so that it will be even length and then we will equally put in both half.



**5. Longest Palindromic Substring**

Given a string s, return *the longest palindromic substring* in s.

**Input:** s = "babad"

**Output:** "bab"

**Explanation:** "aba" is also a valid answer.

**Sol:**

Consider each char as a middle element and expand it on both sides to find the maximum length of palindrome.

For odd length palindrome center will be ‘i’ and for even length palindrome center will be ‘i’ to ‘i+1’.

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

**3. Longest Substring Without Repeating Characters**

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

**Input:** s = "abcabcbb"

**Output:** 3

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

**Sol:**

Use sliding window technique.

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

**32. Longest Valid Parentheses**

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

**Input:** s = ")()())"

**Output:** 4

**Explanation:** The longest valid parentheses substring is "()()".

**Sol:**

We will have 2 pointers open and close, open will store the no of opening brackets and close will store the no of closing brackets.

We will iterate each character and if it is opening bracket, we will increment the count of open pointer. If it is closing bracket then we will increment the close pointer

If any point open == close then it means till there parenthesis is valid and length of the parenthesis will be open + close pointers.

But if close > open then it means closing brackets is more than opening brackets so it will be invalid parenthesis so we will put open = close = 0;

We will do this operation from start to end and end to start index.

While traversing from end to start we will check if open > close the we will make open = close = 0

in above example lets check the steps **Input:** s = ")()())"

start from start to end index

open : 0

close : 1

here at first index we got close > open so we will make open = close = 0

on 2nd iteration open = 1, close = 0

on 3rd iteration open = 1, close = 1…once we get open == close then we will find out max(ans, open + close) we got 2

on 4th iteration open = 2, close = 1

on 5th iteration open = 2, close = 2… open == close then we will find out max(ans, open + close) we got 4

on 6th iteration open = 2 and close = 3 so we will make open = close = 0

till now max = 4

do the reverse order traversing and find the max.

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

**720. Longest Word in Dictionary**

Given an array of strings words representing an English Dictionary, return *the longest word in* words *that can be built one character at a time by other words in* words.

If there is more than one possible answer, return the longest word with the smallest lexicographical order. If there is no answer, return the empty string.

**Input:** words = ["w","wo","wor","worl","world"]

**Output:** "world"

**Explanation:** The word "world" can be built one character at a time by "w", "wo", "wor", and "worl".

**Sol:**

Sort the words…by using Arrays.*sort*(words);

Maintain a set to store the word.

Now iterate the words array

**for** (String word : words) {

**if** (word.length() == 1 || set.contains(word.substring(0, word.length() - 1))) {

set.add(word);

**if** (word.length() > result.length())

result = word;

}

}

Once we wort the array it will be in lexicographically sorted.

It will iterate words one by one

In 1st iteration word = “w” …..its length = 1 so it will be added into set

In 2nd iteration word = wo….its length is 2 so it will go in 2nd if condition and word.substring(0, word.length() - 1) will return w

So it will check in set if w is present or not….as in 1st iteration w is already present it will add this word “wo” in the set.. so by now set will contain “w” and “wo”

In 3rd iteration word = wor and word.substring(0, word.length() - 1) it will return wo which is already present in set so this word also gets added into set

and similarly it will check all the words one by one

and in each step find the max length word and return in the last.

Solution 2: create trie data structure and add all the char to it and then do the dfs to find out the max length word.

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

**1239. Maximum Length of a Concatenated String with Unique Characters**

You are given an array of strings arr. A string s is formed by the **concatenation** of a **subsequence** of arr that has **unique characters**.

Return *the****maximum****possible length* of s.

**Input:** arr = ["cha","r","act","ers"]

**Output:** 6

**Explanation:** Possible longest valid concatenations are "chaers" ("cha" + "ers") and "acters" ("act" + "ers").

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

**1247. Minimum Swaps to Make Strings Equal**

You are given two strings s1 and s2 of equal length consisting of letters "x" and "y" **only**. Your task is to make these two strings equal to each other. You can swap any two characters that belong to **different** strings, which means: swap s1[i] and s2[j].

Return the minimum number of swaps required to make s1 and s2 equal, or return -1 if it is impossible to do so.

**Input:** s1 = "xy", s2 = "yx"

**Output:** 2

**Explanation:** Swap s1[0] and s2[0], s1 = "yy", s2 = "xx".

Swap s1[0] and s2[1], s1 = "xy", s2 = "xy".

Note that you cannot swap s1[0] and s1[1] to make s1 equal to "yx", cause we can only swap chars in different strings.

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

**819. Most Common Word**

Given a string paragraph and a string array of the banned words banned, return *the most frequent word that is not banned*. It is **guaranteed** there is **at least one word** that is not banned, and that the answer is **unique**.

The words in paragraph are **case-insensitive** and the answer should be returned in **lowercase**.

**Input:** paragraph = "Bob hit a ball, the hit BALL flew far after it was hit.", banned = ["hit"]

**Output:** "ball"

**Explanation:**

"hit" occurs 3 times, but it is a banned word.

"ball" occurs twice (and no other word does), so it is the most frequent non-banned word in the paragraph.

Note that words in the paragraph are not case sensitive,

that punctuation is ignored (even if adjacent to words, such as "ball,"),

and that "hit" isn't the answer even though it occurs more because it is banned.

**Sol:**

Here we can take a Set of String and add all the banned word to this set. And one map to store the count of the word. split the paragraph we will get the array of words. We will iterate this array and check if this word is not in the set we will add to the map and increment the count for this word into map.

In the map we will have non banned words, so return the word with max count

**647. Palindromic Substrings**

Given a string s, return *the number of****palindromic substrings****in it*.

A string is a **palindrome** when it reads the same backward as forward.

**Input:** s = "abc"

**Output:** 3

**Explanation:** Three palindromic strings: "a", "b", "c".

**Sol:**

Consider every char as middle element and expand it left and right till its not palindrome. For even palindrome middle element will be ‘i’ and for the odd palindrome middle element will be ‘i+1’

Add the count for the even and odd palindromic result.

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

**1324. Print Words Vertically**

Given a string s. Return all the words vertically in the same order in which they appear in s.  
Words are returned as a list of strings, complete with spaces when is necessary. (Trailing spaces are not allowed).  
Each word would be put on only one column and that in one column there will be only one word.

**Input:** s = "TO BE OR NOT TO BE"

**Output:** ["TBONTB","OEROOE"," T"]

**Explanation:** Trailing spaces is not allowed.

"TBONTB"

"OEROOE"

" T"

**Sol:**

Split the String with “ ” and return the array of strins.

TO

OR

TO

NOT

BE

If we arrange the word after split it looks like above. So if we print 1st column it will be “TOTNB” and 2nd column will be “OROOE” and 3rd column will be “ T” only 3 spaces are there as in question it is mentioned that trailing spaces are not allowed.

So here total no of columns will be 3 which is max length of words. Once we get the length of longest word we iterate and add the 1st char of each word into string builder if i < word.length. if I >= word.length then just add the “ ”. and in last remove the trailing spaces from right.

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

**316. Remove Duplicate Letters**

Given a string s, remove duplicate letters so that every letter appears once and only once. You must make sure your result is **the smallest in lexicographical order** among all possible results.

**Input:** s = "cbacdcbc"

**Output:** "acdb"

**Sol:**

Unique char from the input s = “cbacdcbc” are

Cbad

Bacd

Acdb

Adcb

Adbc

So from out of these option lexicographicalysmaller is acdb

We will iterate it from left to right. Suppose we want to skip ‘c’ then it can be skipped only if it is present later in the string. So for this we should keep track of presence of each letter. So we will take a map and store its count.

Count of each char in “cbacdcbc” are bellows

c – 4, b – 2, a – 1, d – 1

in every iteration count of each char will decrement by 1

**In first iteration ‘c’ will come.**

Decrement its count by 1..so its count will remain **c - 3**

**In the 2nd iteration ‘b’ will come**.

Decrement its count by 1. So its count will remain **b - 1**

Last char of result that is ‘c’ > current char ‘b’ so in this case we will remove the last char from the result. Will do so if its count greater than 0. Here count of c is 3. So we will remove it from the result. Once we delete the char from result we mark that presence to false.

And now add the current char into the result. **So result will contain “b”**

Once we will add the char into result, will mark its presence to true.

**In 3rd iteration ‘a’ will come.**

Decrement its count by 1. So its count will remain **a - 0**

In the result last char is ‘b’ which is greater than current char ‘a’. will check count of last char in result that is 1. so we will remove last char from the result. Once we delete the char from result we mark that presence to false.

And then add current char to result and mark its presence to true. **Now result will contain “a”**

**In the next iteration ‘c’ will come**.

Decrement its count by 1. So its count will remain **c - 2**

Last char of the result which is ‘a’ is less current char ‘c’. So we will directly add this into result and mark its presence to true. **Now result will contain “ac”**

**In the next iteration d will come**.

Decrement its count by 1. So its count will remain **d - 0**

Last char of the result which is ‘c’ less than current char ‘d’ so will add directly into result and mark its presence to true. **Now result will contain “acd”**

**In the next iteration c will come.**

Decrement is count by 1. So its count will remain **c - 1**

‘c’ is already present in the result so will skip it

**In the next iteration b will come**.

Decrement its count by 1. So its count will remain **b - 0**

Last char of the result ‘d’ which is greater than ‘b’ but its count is not greater than 0. So we will add ‘b’ in result.

Now result will contain **“acdb”**

**In the next iteration c will come.**

C is already present so we will skip it.

So our result stack will be “acdb”. Return this result.

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

**557. Reverse Words in a String III**

Given a string s, reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

**Input:** s = "Let's take LeetCode contest"

**Output:** "s'teL ekat edoCteeL tsetnoc"

**Sol:**

Int i = 0;

We will iterate the String. Once we get the white space we stop there. Suppose we got the white space at index j. it means we have some char from i to j – 1 index. we will reverse the string from i to j - 1.

Once it is reversed. We will make i = j + 1 because jth char is the white space so no need to reverse it. So next char onwards which we need to reverse is, will start from j + 1.

Will do this step in loop. So it will reverse the string till 2nd last char. It will not reverse the last char as we wont find the white space after last word.

So to reverse the last word we need to reverse the String i to str.length – 1.

After doing this step. Return the chars.

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

**151. Reverse Words in a String**

Given an input string s, reverse the order of the **words**.

A **word** is defined as a sequence of non-space characters. The **words** in s will be separated by at least one space.

Return *a string of the words in reverse order concatenated by a single space.*

**Note** that s may contain leading or trailing spaces or multiple spaces between two words. The returned string should only have a single space separating the words. Do not include any extra spaces.

**Input:** s = "the sky is blue"

**Output:** "blue is sky the"

**Sol:**

Split the string by “ ”

Then reverse the array from starting from end. If arr[i] != “” then add arr[i] and “ ” to string builder. And in the last remove the last white space from the string.

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

**856. Score of Parentheses**

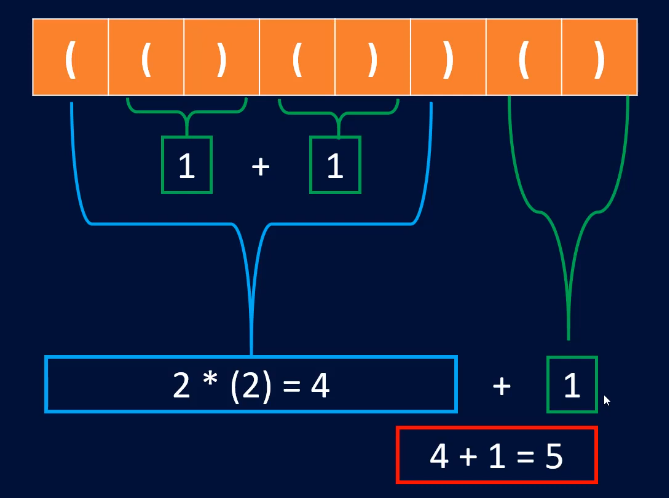
Given a balanced parentheses string s, return *the****score****of the string*.

The **score** of a balanced parentheses string is based on the following rule:

* "()" has score 1.
* AB has score A + B, where A and B are balanced parentheses strings.
* (A) has score 2 \* A, where A is a balanced parentheses string.
* **Input:** s = "(()())()"
* **Output:** 5

Sol:

We will use stack.



We will iterate the string one by one.

If char is ‘(‘ then we will push 0 to the stack. If char is ‘)’ then we will do the calculation as follows.

For 1st char ‘(‘ we will push 0 to the stack…..**so in stack we have 0 as of now.**

For 2nd char ‘(‘ we will push 0 to the stack… **so in the stack we have 0, 0 as of now.**

For 3rd char ‘)’ we will pop the stack till stack.peek() != 0

Here stack.peek() = 0 so it will not go in this while loop.

Will check Math.max(2\*val, 1) it will return 1 so we will pop the stack and push val.

**So now stack will contain 0, 1**

For 4th char ‘(‘ we will push 0 to stack. **So in stack we have 0, 1, 0 as of now**

For 5th char ‘)’ we will pop the stack till stack.peek() != 0

Here stack.peek == 0 so it wont go in while loop

Will check Math.max(2\*val, 1) it will return 1 so we will pop the stack and push val.

**So now stack will contain 0, 1, 1**

For 6th char ‘)’ we will pop the stack till stack.peek() != 0 and add the val values. So val will be 1 + 1 = 2

**So stack will now contain 0**

Once we come out of while loop we check Math.max(2\*val, 1) which will return 4

We will pop the stack and push this val into stack

**So stack will contain now 4**

For 7th char ‘(‘ we will push 0 to stack. **So stack will contain now = 4, 0**

For 8th char ‘)’ we will pop the stak till stack.peek() != 0

It will not go in loop and outside loop we will check Math.max(2\*val, 1) which will return 1

Will pop the stack and push val into it.

**So stack will contain now 4, 1**

We have processed all the char of the string. In last we will sum up the stack values that will come 5 and will return it as result.

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

**214. Shortest Palindrome**

You are given a string s. You can convert s to a palindrome by adding characters in front of it.

Return *the shortest palindrome you can find by performing this transformation*.

**Input:** s = "abcd"

**Output:** "dcbabcd"

Sol:

// TODO

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

**451. Sort Characters By Frequency**

Given a string s, sort it in decreasing order based on the frequency of the characters. The frequency of a character is the number of times it appears in the string.

Return *the sorted string*. If there are multiple answers, return *any of them*.

**Input:** s = "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.

**Sol:**

Take a map and store the count of each char.

Then we will take

Queue<Character> maxHeap = **new** PriorityQueue<>((a, b) -> map.get(b) - map.get(a));

In this maxheap add all the key of the map.

Then iterate the maxheap till it is not empty.

Remove char from maxheap one by one and takes its count from the map and then print the char as many times as it is in map.

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

**443. String Compression**

Given an array of characters chars, compress it using the following algorithm:

Begin with an empty string s. For each group of consecutive repeating characters in chars:

* If the group's length is 1, append the character to s.
* Otherwise, append the character followed by the group's length.

The compressed string s should not be returned separately, but instead, be stored in the input character array chars. Note that group lengths that are 10 or longer will be split into multiple characters in chars.

After you are done modifying the input array, return *the new length of the array*.

You must write an algorithm that uses only constant extra space.

**Input:** chars = ["a","a","b","b","c","c","c"]

**Output:** Return 6, and the first 6 characters of the input array should be: ["a","2","b","2","c","3"]

**Explanation:** The groups are "aa", "bb", and "ccc". This compresses to "a2b2c3".

**Sol:**

Take two pointer I and j and index

Iterate the char array. Initialy I and j will point the same char array.

We will forward j to right till char[j] != char[i]

At index 2 a != b so loop will break.

Now we will assign this a to char[index++] = char[i]

Now we will check if j – I > 1 it means we have count more than 1 for a char. In this case add the count t “” so that count will also be string.

Now iterate the char count and append in the char like char[index++] = ch;

Once it is done assign I = j so that I will point to the next char starting point.

And in the last return the index of the char.

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

**796. Rotate String**

Given two strings s and goal, return true *if and only if* s *can become* goal *after some number of****shifts****on* s.

A **shift** on s consists of moving the leftmost character of s to the rightmost position.

* For example, if s = "abcde", then it will be "bcdea" after one shift.

**Sol:**

Add input string 2 times. Here input string is “abcde” so after adding the same string again input string will be

“abcdeabcde”. If goal is the roatated version of input string then it must be in the newly formed input string.

I mean “abcdeabcde”.contains(“bcdea”) returns true then it is rotated string of input string.

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

**8. String to Integer (atoi)**

Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function).

The algorithm for myAtoi(string s) is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range [-231, 231 - 1], then clamp the integer so that it remains in the range. Specifically, integers less than -231 should be clamped to -231, and integers greater than 231 - 1 should be clamped to 231 - 1.
6. Return the integer as the final result.

**Input:** s = " -42"

**Output:** -42

**Explanation:**

Step 1: " -42" (leading whitespace is read and ignored)

^

Step 2: " -42" ('-' is read, so the result should be negative)

^

Step 3: " -42" ("42" is read in)

^

The parsed integer is -42.

Since -42 is in the range [-231, 231 - 1], the final result is -42.

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

**242. Valid Anagram**

Given two strings s and t, return true *if* t *is an anagram of* s*, and* false *otherwise*.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

**Input:** s = "anagram", t = "nagaram"

**Output:** true

Sol:

Find the frequency of the char from the String s and then while traversing the String t check if same char is coming then decrement the count for that char.

And at last check all the char have count 0 or not. If count == 0 then anagram else not

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