Unit 3: Collections, Strings, and Regular Expressions

1. **Longest Substring Without Repeating Characters (#3)**

Given a string s, find the length of the longest substring without duplicate characters.
Example 1:
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
Example 2:
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
Example 3:
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

2. **Longest Palindromic Substring (#5)**

Given a string s, return *the longest palindromic substring* in s.
Example 1:
Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.
Example 2:
Input: s = "cbbd"
Output: "bb"

3. **Valid Parentheses (#20)**

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.
An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:
Input: s = "()"
Output: true
Example 2:
Input: s = "()[]{}"
Output: true
Example 3:
Input: s = "(]"
Output: false

4. **Valid Palindrome (#125)**

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers. Given a string s, return true *if it is a* ***palindrome****, or* false *otherwise*.

Example 1:
Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.
Example 2:
Input: s = "race a car"
Output: false
Explanation: "raceacar" is not a palindrome.
Example 3:
Input: s = " "
Output: true
Explanation: s is an empty string "" after removing non-alphanumeric characters.
Since an empty string reads the same forward and backward, it is a palindrome.

5. **Find All Anagrams in a String (#438)**

Given two strings s and p, return an array of all the start indices of p's anagrams in s. You may return the answer in any order.

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".
Example 2:
Input: s = "abab", p = "ab"
Output: [0,1,2]
Explanation:
The substring with start index = 0 is "ab", which is an anagram of "ab".
The substring with start index = 1 is "ba", which is an anagram of "ab".
The substring with start index = 2 is "ab", which is an anagram of "ab".

6. **Reverse Words in a String (#151)**

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.

Example 1:
Input: s = "the sky is blue"
Output: "blue is sky the"
Example 2:
Input: s = " hello world "
Output: "world hello"

Explanation: Your reversed string should not contain leading or trailing spaces.
Example 3:
Input: s = "a good   example"
Output: "example good a"
Explanation: You need to reduce multiple spaces between two words to a single space in the reversed string.

7. **Find the Index of the First Occurrence in a String (#28)**

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.
Example 1:
Input: haystack = "sadbutsad", needle = "sad"
Output: 0
Explanation: "sad" occurs at index 0 and 6.
The first occurrence is at index 0, so we return 0.
Example 2:
Input: haystack = "leetcode", needle = "leeto"
Output: -1
Explanation: "leeto" did not occur in "leetcode", so we return -1.

8. **Reverse Prefix of Word (#2000)**

Given a 0-indexed string word and a character ch, reverse the segment of word that starts at index 0 and ends at the index of the first occurrence of ch (inclusive). If the character ch does not exist in word, do nothing.

- For example, if word = "abcdefd" and ch = "d", then you should reverse the segment that starts at 0 and ends at 3 (inclusive). The resulting string will be "dcbaefd".

Return *the resulting string*.
Example 1:
Input: word = "abcdefd", ch = "d"
Output: "dcbaefd"
Explanation: The first occurrence of "d" is at index 3.
Reverse the part of word from 0 to 3 (inclusive), the resulting string is "dcbaefd".
Example 2:
Input: word = "xyxzxe", ch = "z"
Output: "zxyxxe"
Explanation: The first and only occurrence of "z" is at index 3.
Reverse the part of word from 0 to 3 (inclusive), the resulting string is "zxyxxe".
Example 3:
Input: word = "abcd", ch = "z"
Output: "abcd"
Explanation: "z" does not exist in word.
You should not do any reverse operation, the resulting string is "abcd".

9. **Regular Expression Matching (#10)**

Given an input string s and a pattern p, implement regular expression matching with support for '.' and '*' where:

- '.' Matches any single character.
- '*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).
Example 1:
Input: s = "aa", p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".
Example 2:
Input: s = "aa", p = "a*"
Output: true
Explanation: '*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it becomes "aa".
Example 3:
Input: s = "ab", p = ".*"
Output: true
Explanation: ".*" means "zero or more (*) of any character (.)".

10. **Decode String (#394)**

Given an encoded string, return its decoded string. The encoding rule is: k[encoded_string], where the encoded_string inside the square brackets is being repeated exactly k times. Note that k is guaranteed to be a positive integer. You may assume that the input string is always valid; there are no extra white spaces, square brackets are well-formed, etc. Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, k. For example, there will not be input like 3a or 2[4]. The test cases are generated so that the length of the output will never exceed 10^5.
Example 1:
Input: s = "3[a]2[bc]"
Output: "aaabcbc"
Example 2:
Input: s = "3[a2[c]]"
Output: "accaccacc"
Example 3:
Input: s = "2[abc]3[cd]ef"
Output: "abcabccdcdcdef"

11. **Generate Parentheses (#22)**

Given n pairs of parentheses, write a function to *generate all combinations of well-formed parentheses*.
Example 1:
Input: n = 3
Output: ["((()))","(()())","(())()","()(())","()()()"]
Example 2:
Input: n = 1
Output: ["()"]

12. Palindrome Number (#9)

Given an integer x, return true *if* x *is a* ***palindrome****, and* false *otherwise*.
Example 1:
Input: x = 121
Output: true
Explanation: 121 reads as 121 from left to right and from right to left.
Example 2:
Input: x = -121
Output: false
Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.
Example 3:
Input: x = 10
Output: false
Explanation: Reads 01 from right to left. Therefore it is not a palindrome.

13. Jump Game II (#45)

You are given a **0-indexed** array of integers nums of length n. You are initially positioned at index 0.
Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at index i, you can jump to any index (i + j) where:

- 0 <= j <= nums[i] and
- i + j < n

Return *the minimum number of jumps to reach index* n - 1. The test cases are generated such that you can reach index n - 1.
Example 1:
Input: nums = [2,3,1,1,4]
Output: 2
Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.
Example 2:
Input: nums = [2,3,0,1,4]
Output: 2

14. Repeated Substring Pattern (#459)

Given a string s, check if it can be constructed by taking a substring of it and appending multiple copies of the substring together.
Example 1:
Input: s = "abab"
Output: true
Explanation: It is the substring "ab" twice.
Example 2:
Input: s = "aba"
Output: false

15. Roman to Integer (#13)

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 ones 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 a roman numeral, convert it to an integer.
Example 1:
Input: s = "III"
Output: 3
Explanation: III = 3.
Example 2:
Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.
Example 3:
Input: s = "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

16. Add Binary (#67)

Given two binary strings a and b, return *their sum as a binary string*.
Example 1:
Input: a = "11", b = "1"
Output: "100"
Example 2:
Input: a = "1010", b = "1011"
Output: "10101"

17. Number of Matching Subsequences (#792)

Given a string s and an array of strings words, return *the number of* words[i] *that is a subsequence of* s.
A subsequence of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, "ace" is a subsequence of "abcde".

Example 1:
Input: s = "abcde", words = ["a","bb","acd","ace"]
Output: 3
Explanation: There are three strings in words that are a subsequence of s: "a", "acd", "ace".
Example 2:
Input: s = "dsahjpjauf", words = ["ahjpjau","ja","ahbwzgqnuk","tnmlanowax"]
Output: 2

18. Reverse Vowels of a String (#345)

Given a string s, reverse only all the vowels in the string and return it. The vowels are 'a', 'e', 'i', 'o', and 'u', and they can appear in both lower and upper cases, more than once.
Example 1:
Input: s = "IceCreAm"
Output: "AceCreIm"
Explanation:
The vowels in s are ['I', 'e', 'e', 'A']. On reversing the vowels, s becomes "AceCreIm".
Example 2:
Input: s = "leetcode"
Output: "leotcede"

19. Count Binary Substrings (#696)

Given a binary string s, return the number of non-empty substrings that have the same number of 0's and 1's, and all the 0's and all the 1's in these substrings are grouped consecutively.
Substrings that occur multiple times are counted the number of times they occur.
Example 1:
Input: s = "00110011"
Output: 6
Explanation: There are 6 substrings that have equal number of consecutive 1's and 0's: "0011", "01", "1100", "10", "0011", and "01".
Notice that some of these substrings repeat and are counted the number of times they occur.
Also, "00110011" is not a valid substring because all the 0's (and 1's) are not grouped together.
Example 2:
Input: s = "10101"
Output: 4
Explanation: There are 4 substrings: "10", "01", "10", "01" that have equal number of consecutive 1's and 0's.

20. Find All Anagrams in a String (#438)

Given two strings s and p, return an array of all the start indices of p's anagrams in s. You may return the answer in any order.

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".

Example 2:
Input: s = "abab", p = "ab"
Output: [0,1,2]
Explanation:
The substring with start index = 0 is "ab", which is an anagram of "ab".
The substring with start index = 1 is "ba", which is an anagram of "ab".
The substring with start index = 2 is "ab", which is an anagram of "ab".