Algorithms & Data Structures Additional Sample Exam Questions

Integer to Roman

Given a decimal integer number, convert it to roman number.
Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, 3 is written as III in Roman numeral, just three one's added together. 13 is written as XIII, which is simply X + III. The number 28 is written as XXVIII, which is XX + V + III.

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.

Roman to Integer

Given a roman number convert it to a decimal integer

Maximum Profit from Trading Stocks

You are given an integer array prices where prices[i] is the price of a given stock on the ith day.

Design an algorithm to find the maximum profit. You may complete at most k transactions.

Notice that you may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

Example 1:
Input: k = 2, prices = [2,4,1]
Return 2
Explanation: Buy on day 1 (price = 2) and sell on day 2 (price = 4), profit = 4-2 = 2.

Example 2:
Input: k = 2, prices = [3,2,6,5,0,3]
Return 7
Explanation: Buy on day 2 (price = 2) and sell on day 3 (price = 6), profit = 6-2 = 4. Then buy on day 5 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3.

Check Word Abbreviation

Given a non-empty string s and an abbreviation abbr, return whether the string matches with the given abbreviation.

A string such as "word" contains only the following valid abbreviations:
["word", "1ord", "w1rd", "wo1d", "wor1", "2rd", "w2d", "wo2", "1o1d", "1or1", "w1r1", "1o2", "2r1", "3d", "w3", "4"]

Notice that only the above abbreviations are valid abbreviations of the string "word". Any other string is not a valid abbreviation of "word".

Note:
Assume s contains only lowercase letters and abbr contains only lowercase letters and digits.

Example 1:
Given s = "internationalization", abbr = "i12iz4n":
Return True.

Example 2:
Given s = "apple", abbr = "a2e":
Return False.

Password Checker

A password is considered strong if below conditions are all met:

- It has at least 6 characters and at most 20 characters.
- It must contain at least one lowercase letter, at least one uppercase letter, and at least one digit.

- It must NOT contain three repeating characters in a row ("...aaa..." is weak, but "...aa...a..." is strong, assuming other conditions are met).

Write a function, that takes a string s as input, and return the MINIMUM change required to make s a strong password. If s is already strong, return 0.

Insertion, deletion or replace of any one character are all considered as one change.

Sliding Window Median

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle value.

Examples:
[2,3,4] , the median is 3

[2,3], the median is (2 + 3) / 2 = 2.5

Given an array nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Your job is to output the median array for each window in the original array.

For example,
Given nums = [1,3,-1,-3,5,3,6,7], and k = 3.

```
Window position                Median
---------------               -----
[1  3  -1] -3  5  3  6  7       1
 1 [3  -1  -3] 5  3  6  7       -1
 1  3 [-1  -3  5] 3  6  7       -1
 1  3  -1 [-3  5  3] 6  7       3
 1  3  -1  -3 [5  3  6] 7       5
 1  3  -1  -3  5 [3  6  7]      6
```

Therefore, return the median sliding window as [1,-1,-1,3,5,6].

Note:
You may assume k is always valid, ie: k is always smaller than input array's size for non-empty array.

Design Search Autocomplete System

Design a search autocomplete system for a search engine. Users may input a sentence (at least one word and end with a special character '#'). For each character they type except '#', you need to return the top 3 historical hot sentences that have prefix the same as the part of sentence already typed. Here are the specific rules:

1. The hot degree for a sentence is defined as the number of times a user typed the exactly same sentence before.
2. The returned top 3 hot sentences should be sorted by hot degree (The first is the hottest one). If several sentences have the same degree of hot, you need to use ASCII-code order (smaller one appears first).
3. If less than 3 hot sentences exist, then just return as many as you can.
4. When the input is a special character, it means the sentence ends, and in this case, you need to return an empty list.

Your job is to implement the following functions:

The constructor function:
AutocompleteSystem(sentences, times): This is the constructor. The input is historical data. Sentences is a string array consists of previously typed sentences. Times is the corresponding times a sentence has been typed. Your system should record these historical data.

Now, the user wants to input a new sentence. The following function will provide the next character the user types:

input(char c): The input c is the next character typed by the user. The character will only be lower-case letters ('a' to 'z'), blank space (' ') or a special character ('#'). Also, the previously typed sentence should be recorded in your system. The output will be the top 3 historical hot sentences that have prefix the same as the part of sentence already typed.

Example:
Operation: AutocompleteSystem(["i love you", "island","ironman", "i love leetcode"], [5,3,2,2])
The system have already tracked down the following sentences and their corresponding times:
"i love you" : 5 times
"island" : 3 times
"ironman" : 2 times
"i love leetcode" : 2 times
Now, the user begins another search:

Operation: input('i')
Output: ["i love you", "island","i love leetcode"]
Explanation:

There are four sentences that have prefix "i". Among them, "ironman" and "i love leetcode" have same hot degree. Since ' ' has ASCII code 32 and 'r' has ASCII code 114, "i love leetcode" should be in front of "ironman". Also we only need to output top 3 hot sentences, so "ironman" will be ignored.

Operation: input(' ')
Output: ["i love you","i love leetcode"]
Explanation:
There are only two sentences that have prefix "i ".

Operation: input('a')
Output: []
Explanation:
There are no sentences that have prefix "i a".

Operation: input('#')
Output: []
Explanation:
The user finished the input, the sentence "i a" should be saved as a historical sentence in the system. And the following input will be counted as a new search.