Weekly Contest 3

392. Is Subsequence

Solved

Easy | Topics | Companies

Given two strings `s` and `t`, return `true` *if* `s` *is a* ***subsequence*** *of* `t`, *or* `false` *otherwise.*

A **subsequence** of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., `"ace"` is a subsequence of `"abcde"` while `"aec"` is not).

S → empty → True
T → empty → False

Example 1:

```
Input: s = "abc", t = "ahbgdc"
Output: true
```

Example 2:

```
Input: s = "axc", t = "ahbgdc"
Output: false
```

Constraints:

- `0 <= s.length <= 100`
- `0 <= t.length <= 10^4`
- `s` and `t` consist only of lowercase English letters.

```java
public static boolean isSubsequence(String s, String t) {
    char[] sChars = s.toCharArray();
    char[] tChars = t.toCharArray();
    int sLength = sChars.length;
    int tLength = tChars.length;

    if (sLength == 0) return true;
    if (tLength == 0) return false;
    int sIdx = 0;
    int tIdx = 0;

    while (sIdx < sLength && tIdx < tLength) {
        if (sChars[sIdx] == tChars[tIdx]) sIdx++;
        tIdx++;
    } return sIdx == sLength;
}
```

394. Decode String

Solved

Medium | Topics | Companies

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"
```

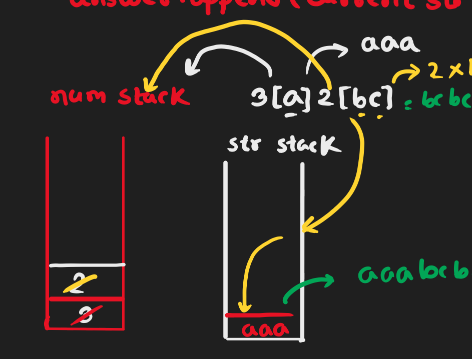
Example 3:

```
Input: s = "2[abc]3[cd]ef"
Output: "abcabccdcdcdef"
```

```java
class Solution {
    public String decodeString(String s) {
        Stack<Integer> numStack = new Stack<>();
        Stack<StringBuilder> strStack = new Stack<>();

        int num = 0;
        StringBuilder sb = new StringBuilder();
        for (char c : s.toCharArray()) {
            if (Character.isDigit(c)) {
                num = num * 10 + c - '0';
            } else if (c == '[') {
                numStack.push(num);
                strStack.push(sb);
                num = 0;
                sb = new StringBuilder();
            } else if (c == ']') {
                int top = numStack.pop();
                StringBuilder topStr = strStack.pop();
                while (top-- > 0)
                    topStr.append(sb);
                if (!topStr.isEmpty()) sb = topStr;
            } else {
                sb.append(c);
            }
        } return sb.toString();
    }
}
```

1) Keep a num stack
2) Keep a str stack
3) [→ push current str to the stack, → push into num stack
4) char → append to current str
5)] → pop from num stack
answer.append(current str × num.pop())



395. Longest Substring with At Least K Repeating Characters

Description | Editorial | Solutions | Submissions

Medium | Topics | Companies

Given a string `s` and an integer `k`, return *the length of the longest substring of* `s` *such that the frequency of each character in this substring is greater than or equal to* `k`.

if no such substring exists, return 0.

Example 1:

```
Input: s = "aaabb", k = 3
Output: 3
Explanation: The longest substring is "aaa", as 'a' is repeated 3 times.
```

Example 2:

```
Input: s = "ababbc", k = 2
Output: 5
Explanation: The longest substring is "ababb", as 'a' is repeated 2 times and 'b' is repeated 3 times.
```

Constraints:

- `1 <= s.length <= 10^4`
- `s` consists of only lowercase English letters.
- `1 <= k <= 10^5`

abacbcdbaabccad

Lets say, we have a frequency map.

a → 5 K = 3
b → 3 So, we have to look for the substrings that do not have "d"
c → 3 x x x x x d x x x x x d x x x
d → 2 < K (left) (right)
e → 4 ← left — — right →

Longestsubstring (str, K) → Base Case
  if (K ≤ 1) → Return 1
  if (str is empty or n < K) Return 0
  freq[] ← store the frequency of each char (of len 26)
  for(c:str) freq[c]++
  # find the char which has < K freq
  left ← 0
  while(left < n and freq[str[left]] ≥ K)
    left ++
  if(left ≥ n-1) return left Reached the end
  leftmax = Longestsubstring(str[0...left], K)
  # Skip the char which has freq < K
  while(left < n and freq[str[left]] < K)
    left ++
  Rightmax = Longestsubstring(str[left...n], K)
Answer ← max = Max(leftmax, Rightmax)

```java
public static int longestSubstring(String s, int k) {  4 usages
    // base cases
    int n = s.length();
    if (n == 0 || n < k) return 0;
    if (k <= 1) return n;

    // store the frequency of each char
    int[] freq = new int[26];
    for (char c : s.toCharArray())
        freq[c-'a']++;

    int left = 0;
    // look for the chars which has the frequency < k
    // because the resultant substring should not contain this char
    while (left < n && freq[s.charAt(left)-'a'] >= k)
        left++; // when the char is found, split the string into left and
    // check we have checked all the characters
    if (left >= n-1) return left; // reached the end

    int leftMax = longestSubstring(s.substring(0, left), k);

    // ignore the chars which has frequency < k
    while (left < n && freq[s.charAt(left)-'a'] < k)
        left++;

    int rightMax = longestSubstring(s.substring(left), k);
    return Math.max(leftMax, rightMax);
}
```

393. UTF-8 Validation

Medium | Topics | Companies | Hint

Given an integer array `data` representing the data, return whether it is a valid **UTF-8** encoding (i.e. it translates to a sequence of valid UTF-8 encoded characters).

A character in **UTF8** can be from **1 to 4 bytes** long, subjected to the following rules:

1. For a **1-byte** character, the first bit is a `0`, followed by its Unicode code.
2. For an **n-bytes** character, the first `n` bits are all one's, the `n + 1` bit is `0`, followed by `n - 1` bytes with the most significant `2` bits being `10`.

This is how the UTF-8 encoding would work:

```
   Number of Bytes   |        UTF-8 Octet Sequence
                     |              (binary)
   --------------------+-----------------------------------------
            1          |   0xxxxxxx
            2          |   110xxxxx 10xxxxxx
            3          |   1110xxxx 10xxxxxx 10xxxxxx
            4          |   11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

`x` denotes a bit in the binary form of a byte that may be either `0` or `1`.

Note: The input is an array of integers. Only the **least significant 8 bits** of each integer is used to store the data. This means each integer represents only 1 byte of data.

Example 1:

```
Input: data = [197,130,1]
Output: true
Explanation: data represents the octet sequence: 11000101 10000010 00000001.
It is a valid utf-8 encoding for a 2-bytes character followed by a 1-byte character.
```