Weekly Contest - Warmup

387. First Unique Character in a String

Solved

Easy | Topics | Companies

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

Example 1:

Input: `s = "leetcode"`

Output: `0`

Explanation:

The character `'l'` at index 0 is the first character that does not occur at any other index.

Example 2:

Input: `s = "loveleetcode"`

Output: `2`

Example 3:

Input: `s = "aabb"`

Output: `-1`

387

counts frequency of each character

```java
class Solution {
    public int firstUniqChar(String s) {
        int[] freq = new int[26];
        for (char c : s.toCharArray())
            freq[c-'a']++;

        for (int i=0; i<s.length(); i++)
            if (freq[s.charAt(i)-'a'] == 1)
                return i;
        return -1;
    }
}
```

check which character has frequency of 1

Description | Editorial | Solutions | Submissions

386. Lexicographical Numbers

Solved

Medium | Topics | Companies

Given an integer `n`, return all the numbers in the range `[1, n]` sorted in lexicographical order.

You must write an algorithm that runs in `O(n)` time and uses `O(1)` extra space.

Example 1:

Input: `n = 13`
Output: `[1,10,11,12,13,2,3,4,5,6,7,8,9]`

Example 2:

Input: `n = 2`
Output: `[1,2]`

Constraints:

- `1 <= n <= 5 * 10^4`

386

func lexicalOrder(n)
ans[] ← will be answer
for(num = 1; num <= 10; num++)
solve(ans, num, n)
return ans;

func solve(ans[], num, n)
if(num > n) return
ans.add(num)
num *= 10
for(i=1, i<10, i++)
if(num + i > n) return
solve(ans, num+i, n)

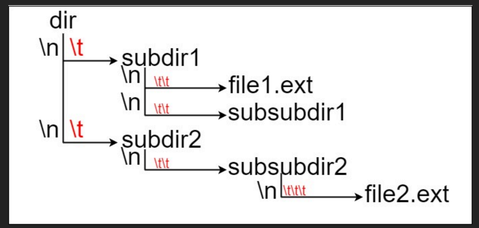
```java
class Solution {
    public List<Integer> lexicalOrder(int n) {
        List<Integer> answer = new ArrayList<>();
        for (int number = 1; number <= 9; number++)
            solve(answer, number, n);
        return answer;
    }
    private void solve(List<Integer> answer, int number, int n) {
        if (number > n) return;
        answer.add(number);
        number *= 10;

        for (int i=0; i<=9; i++) {
            if (number+i > n) return;
            solve(answer, number+i, n);
        }
    }
}
```

388. Longest Absolute File Path

Medium | Topics | Companies

Suppose we have a file system that stores both files and directories. An example of one system is represented in the following picture:



Here, we have `dir` as the only directory in the root. `dir` contains two subdirectories, `subdir1` and `subdir2`. `subdir1` contains a file `file1.ext` and subdirectory `subsubdir1`. `subdir2` contains a subdirectory `subsubdir2`, which contains a file `file2.ext`.

In text form, it looks like this (with ⟶ representing the tab character):

```
dir
⟶ subdir1
⟶ ⟶ file1.ext
⟶ ⟶ subsubdir1
⟶ subdir2
⟶ ⟶ subsubdir2
⟶ ⟶ ⟶ file2.ext
```

If we were to write this representation in code, it will look like this: `"dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext"`. Note that the `'\n'` and `'\t'` are the new-line and tab characters.

Every file and directory has a unique **absolute path** in the file system, which is the order of directories that must be opened to reach the file/directory itself, all concatenated by `'/'s`. Using the above example, the **absolute path** to `file2.ext` is `"dir/subdir2/subsubdir2/file2.ext"`. Each directory name consists of letters, digits, and/or spaces. Each file name is of the form `name.extension`, where `name` and `extension` consist of letters, digits, and/or spaces.

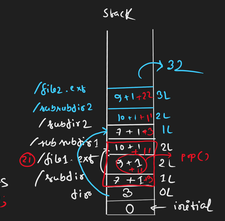
Given a string `input` representing the file system in the explained format, return *the length of the* ***longest absolute path*** *to a* ***file*** *in the abstracted file system.* If there is no file in the system, return `0`.

Note that the testcases are generated such that the file system is valid and no file or directory name has length 0.

388

func LongestPath(input)
lines[] ← input(\n)
stack<int> ← to store the length of
stack.push(0)
for(line : lines)
level ← lastIndexOf(\t) + 1
→ Extra 0
while(stack.size - 1 > level)
stack.pop()
len ← stack.top() + 1 + line.length - level;
# stack.push(len) ✗ ignore
if (line.contains(.))
maxLen ← max(maxLen, len) // no/at the end of file
else stack.push(len);
return maxLen

dir/file remove extra spaces




```java
public static int lengthLongestPath(String input) { 2 usages
    String[] lines = input.split( regex: "\n");
    int maxLen = 0;
    Stack<Integer> stack = new Stack<>();
    stack.push( item: 0);

    for (String line : lines) {
        System.out.println(line);
        int level = line.lastIndexOf( str: "\t") + 1;
        while (stack.size()-1 > level)
            stack.pop();

        int length = stack.peek() + 1 + line.length() - level;
        // +1 for / and - level to ignore extra spaces
        if (line.contains("."))
            maxLen = Math.max(maxLen, length-1); // no / at the end of file
        else stack.push(length);
    }
    return maxLen;
}
```