



NYU | TANDON

LeetCode Bootcamp

Presented By: Spriha Jha

10.10.2022

Session Outline

- 01.** Introduction to Strings ([CheatSheet](#))
- 02.** Problem Sets
- 03.** Debrief & Q/A

PART 02

Problem Sets

Steps to approach the question:

Understand the problem

Take time to carefully read through the problem from start to finish is critical in finding the correct and complete solution to the problem in hand.

Code your solution

Map out your solution before you write any code. Avoid too much time trying to find the perfect solution. Validate your solution early and often.

Manage your time

Don't forget, you have multiple questions to complete within a said time. Make sure you allocate enough time to carefully consider all problems.

Problem 1: Valid Palindrome

The screenshot shows the LeetCode interface for problem 125, 'Valid Palindrome'. The problem is categorized as 'Easy' with 5024 likes and 6081 dislikes. The description states that 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. The task is to return true if it is a palindrome, or false otherwise. Three examples are provided: Example 1 with input 'A man, a plan, a canal: Panama' and output true; Example 2 with input 'race a car' and output false; and Example 3 with an empty string and output true. The code editor on the right shows a Python3 solution with a class Solution and a method isPalindrome.

LeetCode Explore Problems Interview Contest Discuss Store

Description Solution Discuss (999+) Submissions

125. Valid Palindrome

Easy 5024 6081 Add to List Share

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.

```
1 class Solution:
2     def isPalindrome(self, s: str) -> bool:
3
```

Problems Pick One < Prev 125/2435 Next > Console Contribute i Run Code Submit

PROBLEM 1

Approach: Two Pointers

```
def isPalindrome(self, s: str) -> bool:
```

```
    # Define the two pointers on each end
```

```
    i, j = 0, len(s) - 1
```

```
    while i < j:
```

```
        # Increase the left-end pointer if current character is non-alphanumeric
```

```
        while i < j and not s[i].isalnum():
```

```
            i += 1
```

```
        # Decrease the right-end pointer if current character is non-alphanumeric
```

```
        while i < j and not s[j].isalnum():
```

```
            j -= 1
```

```
        # Compare both pointers should point to equivalent character or else break early
```

```
        if s[i].lower() != s[j].lower():
```

```
            return False
```

```
    #Increase the pointers
```

```
    i += 1
```

```
    j -= 1
```

```
    return True
```

Time complexity: $O(n)$, in length n of the string. We traverse over each character at-most once, until the two pointers meet in the middle, or when we break and return early.

Space complexity: $O(1)$, no extra space required, at all.



Problem 2: Unique email addresses

LeetCode

Explore

Problems

Interview

Contest

Discuss

Store

Description

Solution

Discuss (999+)

Submissions

Python3

Autocomplete

929. Unique Email Addresses

Easy

1992

259

Add to List

Share

Every **valid email** consists of a **local name** and a **domain name**, separated by the '@' sign. Besides lowercase letters, the email may contain one or more '.' or '+'.

- For example, in "alice@leetcode.com", "alice" is the **local name**, and "leetcode.com" is the **domain name**.

If you add periods '.' between some characters in the **local name** part of an email address, mail sent there will be forwarded to the same address without dots in the local name. Note that this rule **does not apply** to **domain names**.

- For example, "alice.z@leetcode.com" and "alicez@leetcode.com" forward to the same email address.

If you add a plus '+' in the **local name**, everything after the first plus sign **will be ignored**. This allows certain emails to be filtered. Note that this rule **does not apply** to **domain names**.

- For example, "m.y+name@email.com" will be forwarded to "my@email.com".

It is possible to use both of these rules at the same time.

Given an array of strings `emails` where we send one email to each `emails[i]`, return the *number of different addresses that actually receive mails*.

Example 1:

Input: emails = ["test.email+alex@leetcode.com","test.e.mail+bob.cathy@leetcode.com","testemail@leetcode.com"]

Output: 2

```
1 class Solution:
2     def numUniqueEmails(self, emails: List[str]) -> int:
3
```

Problems

Pick One

< Prev

929/2435

Next >

Console

Contribute

Run Code

Submit

PROBLEM 2

Approach: String Split method

```
def numUniqueEmails(self, emails: List[str]) -> int:
```

```
    # Hashset to store all the unique emails.
```

```
    uniqueEmails = set()
```

```
    for email in emails:
```

```
        # Split into two parts: local and domain.
```

```
        name, domain = email.split('@')
```

```
        # Split local by '+' and replace all '!' with ''.
```

```
        local = name.split('+')[0].replace('!', '')
```

```
        # Concatenate local, '@', and domain.
```

```
        uniqueEmails.add(local + '@' + domain)
```

```
    return len(uniqueEmails)
```

Let N be the number of the emails and M be the average length of an email.

Time complexity: $O(N \cdot M)$, The split method must iterate over all of the characters in each email and the replace method must iterate over all of the characters in each local name.

Space complexity: $O(N \cdot M)$ In the worst case, when all emails are unique, we will store every email address given to us in the hash set.

Problem 3: Subdomain visit count

LeetCode

Explore

Problems

Interview

Contest

Discuss

Store

Description

△ Solution

Discuss (999+)

Submissions

Python3

Autocomplete

1

2

3

class Solution:

def subdomainVisits(self, cpdomains: List[str]) -> List[str]:

811. Subdomain Visit Count

Medium 1274 1202 Add to List Share

A website domain "discuss.leetcode.com" consists of various subdomains. At the top level, we have "com", at the next level, we have "leetcode.com" and at the lowest level, "discuss.leetcode.com". When we visit a domain like "discuss.leetcode.com", we will also visit the parent domains "leetcode.com" and "com" implicitly.

A **count-paired domain** is a domain that has one of the two formats "rep d1.d2.d3" or "rep d1.d2" where rep is the number of visits to the domain and d1.d2.d3 is the domain itself.

- For example, "9001 discuss.leetcode.com" is a **count-paired domain** that indicates that discuss.leetcode.com was visited 9001 times.

Given an array of **count-paired domains** cpdomains, return an array of the **count-paired domains** of each subdomain in the input. You may return the answer in **any order**.

Example 1:

Input: cpdomains = ["9001 discuss.leetcode.com"]
Output: ["9001 leetcode.com","9001 discuss.leetcode.com","9001 com"]
Explanation: We only have one website domain: "discuss.leetcode.com". As discussed above, the subdomain "leetcode.com" and "com" will also be visited. So they will all be visited 9001 times.

Example 2:

Input: cpdomains = ["900 google.mail.com", "50 yahoo.com", "1 intel.mail.com", "5 wiki.org"]
Output: ["901 mail.com","50 yahoo.com","900 google.mail.com","5 wiki.org","5 org","1 intel.mail.com","951 com"]

⌵ Problems

✕ Pick One

< Prev

811/2435

Next >

Console

Contribute

► Run Code

Submit

PROBLEM 3

Approach: Hash Map

```
def subdomainVisits(self, cpdomains):  
    ans = collections.Counter()  
  
    for domain in cpdomains:  
        count, domain = domain.split() # Separate the domain and its count  
        count = int(count) # Converts count from String to Int  
        fragments = domain.split('.') # Separates the subdomain fragments  
  
        # Combines the count of subdomains  
        for i in range(len(fragments)-1):  
            ans[".".join(fragments[i:])]. += count  
  
    return ["{} {}".format(ct, dom) for dom, ct in ans.items()]
```

Time complexity: $O(N)$, where N is the length of `cpdomains`, and assuming the length of `cpdomains[i]` is fixed.

Space complexity: $O(N)$, the space used in our count.

Problem 4: Find all Anagrams in a String

LeetCode

Explore

Problems

Interview

Contest

Discuss

Store

Description

Solution

Discuss (999+)

Submissions

Python3

Autocomplete

0

438. Find All Anagrams in a String

Medium 8829 275 Add to List Share

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**.

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.

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".

Constraints:

- `1 <= s.length, p.length <= 3 * 104`

```
1 class Solution:
2     def findAnagrams(self, s: str, p: str) -> List[int]:
3
```

Problems

Pick One

Prev 438/2435 Next

Console

Contribute

Run Code

Submit

PROBLEM 4

Approach: Sliding window with Array

```
def findAnagrams(self, s: str, p: str) -> List[int]:
    s_length, p_length = len(s), len(p)
    if s_length < p_length:
        return []

    p_count, s_count = [0] * 26, [0] * 26
    # build reference array using string p
    for char in p:
        p_count[ord(char) - ord('a')] += 1

    output = []
    # sliding window on the string s
    for i in range(s_length):
        # add one more letter on the right side of the window
        s_count[ord(s[i]) - ord('a')] += 1
        # remove one letter from the left side of the window
        if i >= p_length:
            s_count[ord(s[i - p_length]) - ord('a')] -= 1
        # compare array in the sliding window with the reference array
        if p_count == s_count:
            output.append(i - p_length + 1)

    return output
```

PART 06

Q/A

Slack Hours

[Join Slack Workspace!](#)

Office Hours: Tuesday (10AM - 1PM)

Problem Assignments

- 01.** Longest Palindrome (Easy)
- 02.** Valid Parentheses (Easy)
- 03.** Isomorphic Strings (Easy)
- 04.** Zigzag Conversion (Medium)
- 05.** Longest Palindromic Substring (Medium)
- 06.** Longest Substring without repeating characters (Medium)
- 07.** Minimum window substring (Hard)



Thank you!

Upcoming: Linked Lists (17/10)