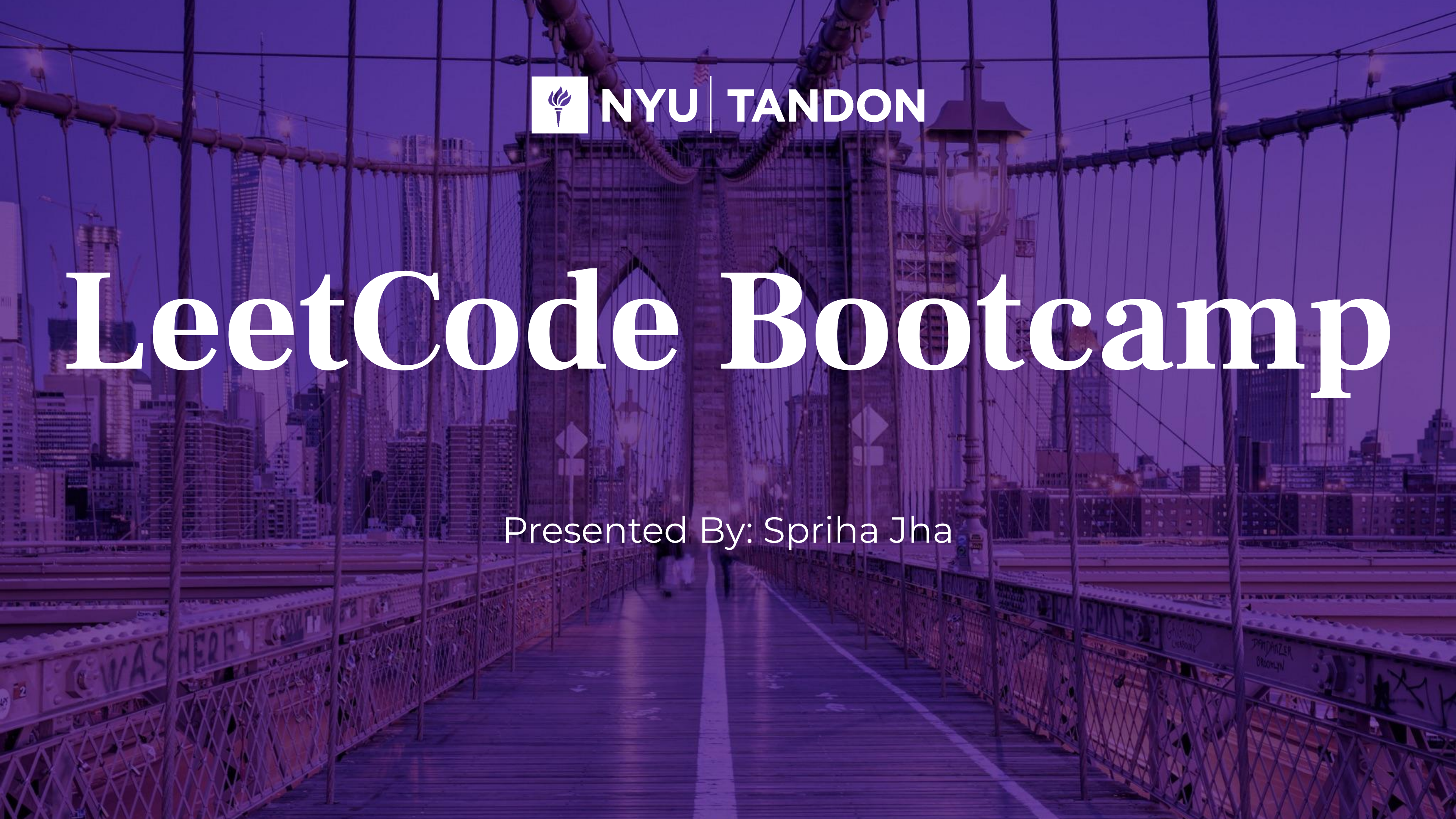


NYU | TANDON

LeetCode Bootcamp

Presented By: Spriha Jha

Session Outline

Strings

Corner cases:

- Empty string
- String with 1 or 2 characters
- String with repeated characters
- Strings with only distinct characters

Techniques:

- Counting characters
- Anagram
- Palindrome

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

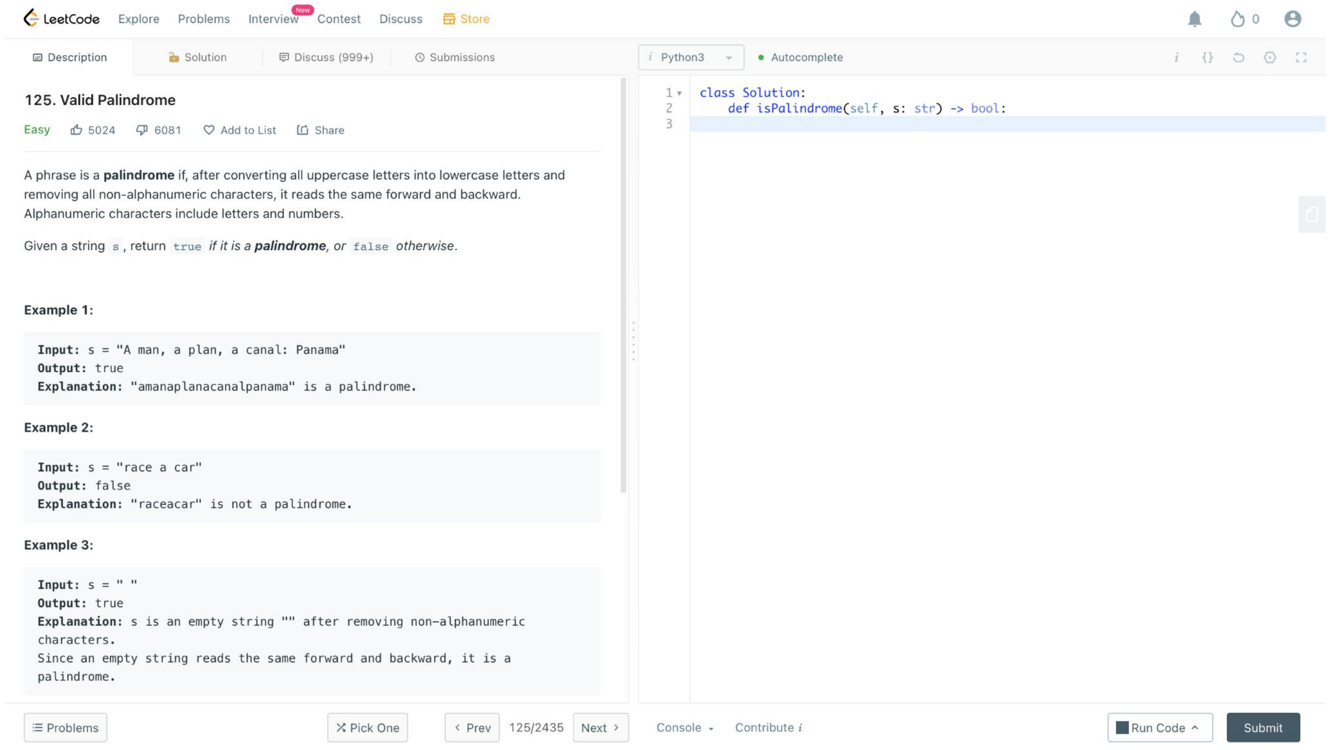
Map out your solution before you write any code. Avoid too much time trying to find the perfect solution. Validate your solution early and often.

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.

PART 02

Problem 1: Valid Palindrome



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.
```

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

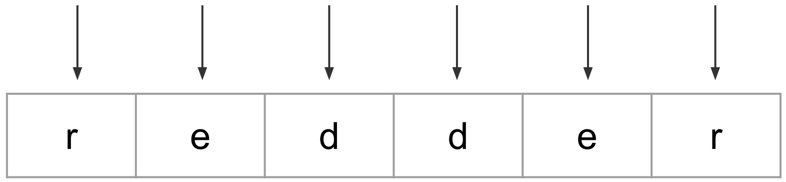
Palindrome

A palindrome is a word, phrase, or sequence that reads the same backwards as forwards.

# Compare with reverse
Filter out non-alphanumeric characters.
Convert the remaining characters to lower-case.
Compare the original and the reversed strings.

# Two pointers
Since the input string contains characters that we need to ignore in our palindromic check, it becomes tedious to figure out the real middle point of our palindromic input.

Instead of going outwards from the middle,
we could just go inwards towards the middle!



PROBLEM 1

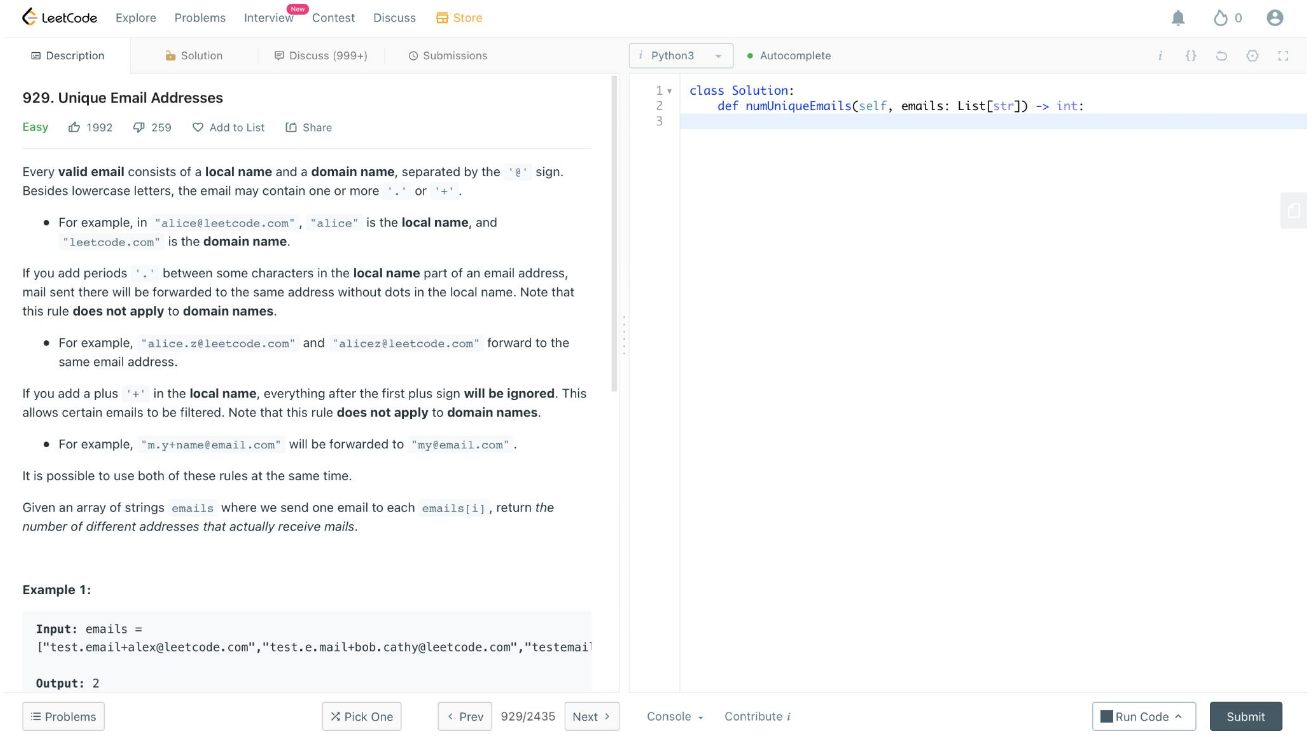
Approach: Two Pointers

```
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.

NYU | TANDON

Problem 2: Unique email addresses



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","testemai

Output: 2
```

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

Unique email check:

#Rules to clean email
- If there are periods '.' in local name ignore them.

- If there is a plus '+' in local name skip all local name characters till '@'.

- There is only one '@' symbol and the substring after it is our domain name; we will keep the domain name as it is.

# Use built-in functions
- Split the string into two parts separated by'@', local name, and domain name.

- Split the local name into parts separated by '+'.

- Since we do not need the part after '+', let the first part be the local name.

- Remove all '.' from the local name and append the domain name to it.

- After cleaning the email, insert it into the hash set.

- Return the size of the hash set.

NYU | TANDON

PROBLEM 2

Approach: String Split method

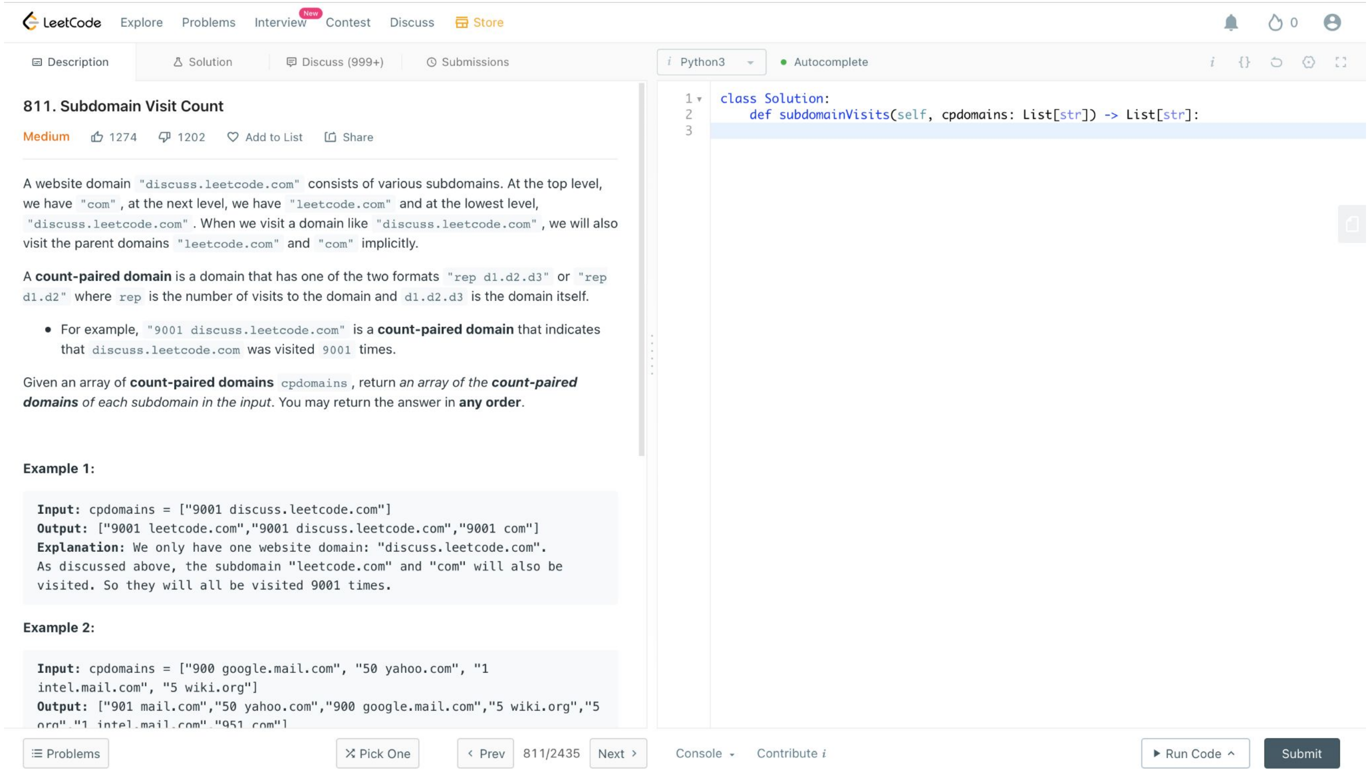
```
def numUniqueEmails(self, emails: List[str]) -> int:
    # Hashset to store all the unique emails.
    uniqueEmails = set()

    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.

NYU TANDON

Problem 3: Subdomain visit count



811. Subdomain Visit Count

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"]
```

```python
class Solution:
    def subdomainVisits(self, cpdomains: List[str]) -> List[str]:
```

Subdomain counter:

#Rules to count subdomain
- For an address like a.b.c, we will count a.b.c, b.c, and c.

- For an address like x.y, we will count x.y and y.

- To count these strings, we will use a hash map.

- To split the strings into the required pieces, we will use library split functions.

# Use collections.counter
- Counter is a dict subclass for counting hashable objects. (supports multiset, unordered, faster access)

- Elements are stored as dictionary keys and their counts are stored as dictionary values.

- Counts are allowed to be any integer value including zero or negative counts.

PROBLEM 3

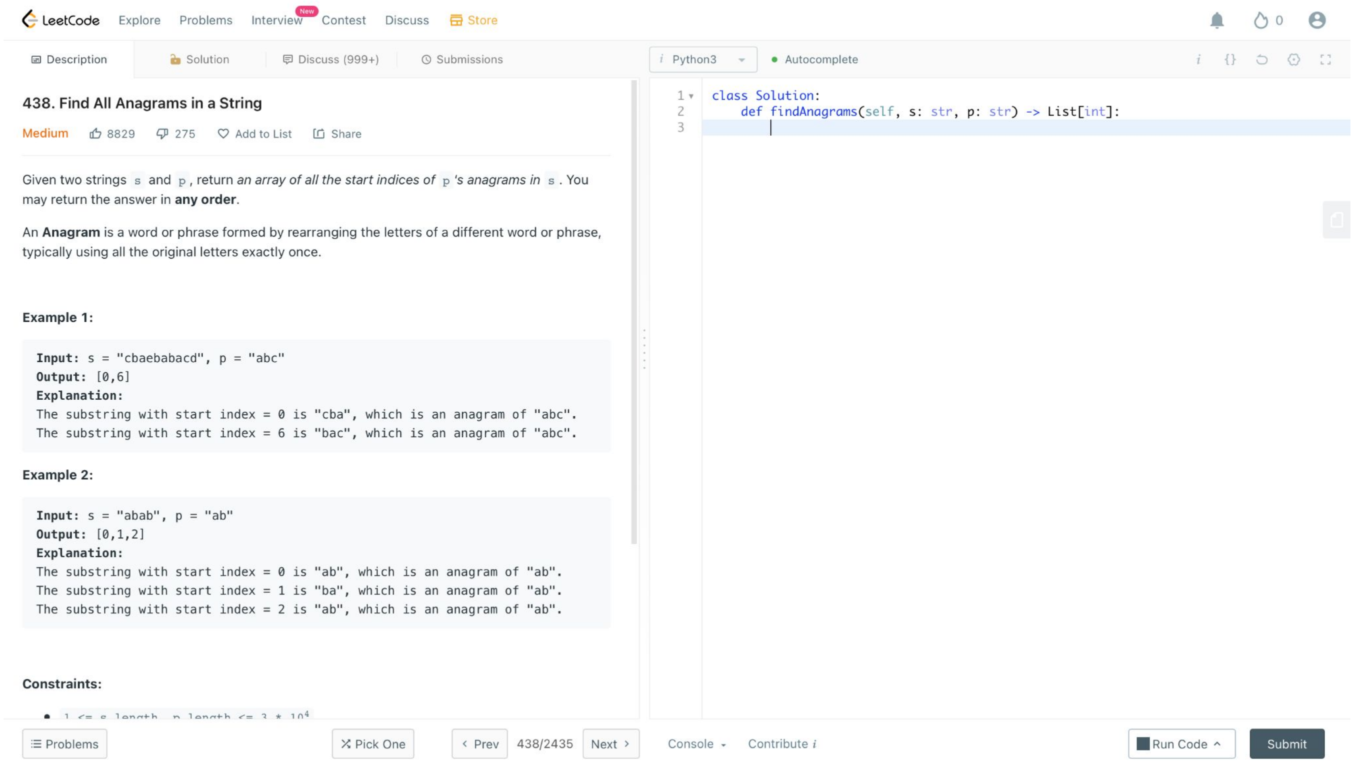
Approach: Hash Map

```
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)):
            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.

NYU | TANDON

Problem 4: Find all Anagrams in a String



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 * 10^4`

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

Sliding Window

- Reduces the use of nested loops.
- Slide over the data in chunks.
- Window moves across the input array from one end to the other.
- The input data structure does support random access.
- Data is processed via segmentation.

Anagrams

# Using 26-elements array instead of hashmap:
- Element number 0 contains count of letter a.
- Element number 1 contains count of letter b.

...

- Element number 25 contains count of letter z.

Algorithm

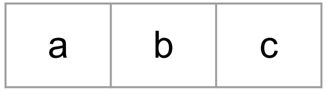
# Steps
- Build reference array pCount for string p.

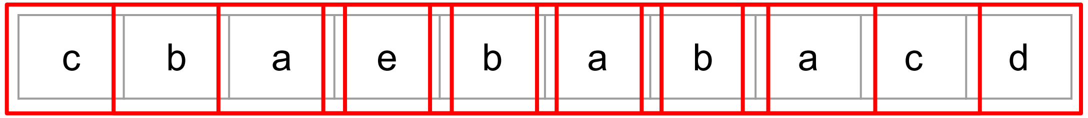
- Move sliding window along the string s.

- Recompute sliding window array sCount at each step by adding one letter on the right and removing one letter on the left.

- If sCount == pCount, update the output list.

- Return output list.





NYU | TANDON

Approach: Sliding window with Hashmap

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

    p_count = Counter(p)
    s_count = Counter()

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

    return output
```

PART 06

Q/A

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)

NYU | TANDON

Thank you!

Upcoming: Matrix & Linked Lists