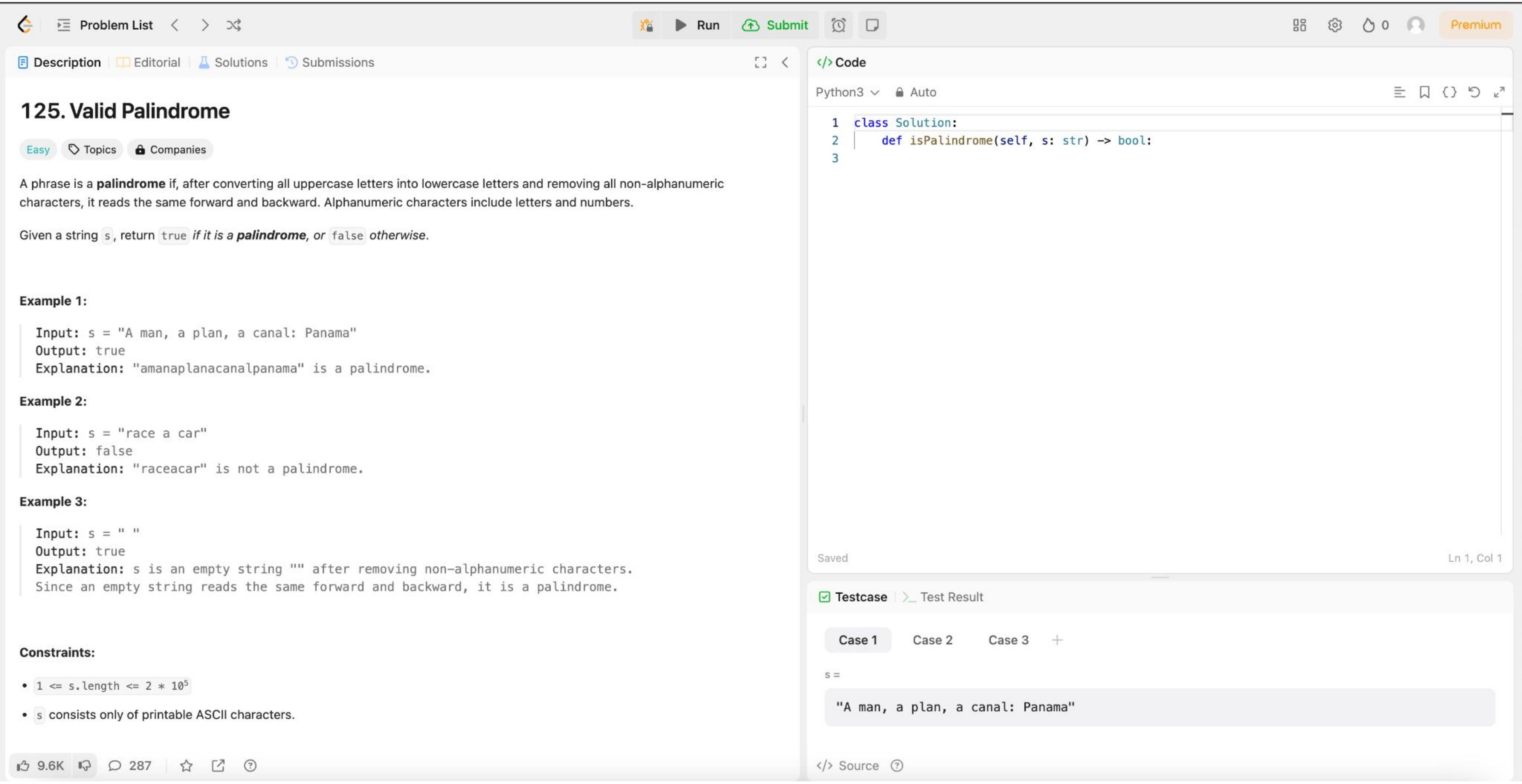


125. Valid Palindrome

Easy · Topics · Companies

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.
```

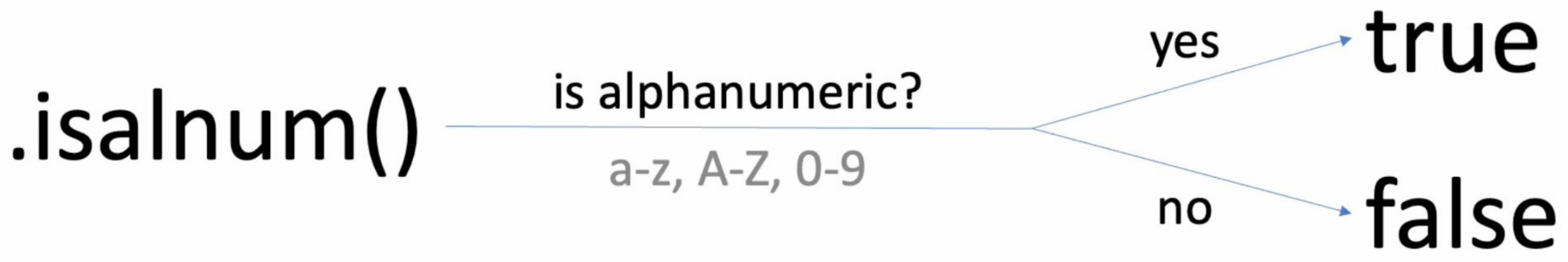
Constraints:

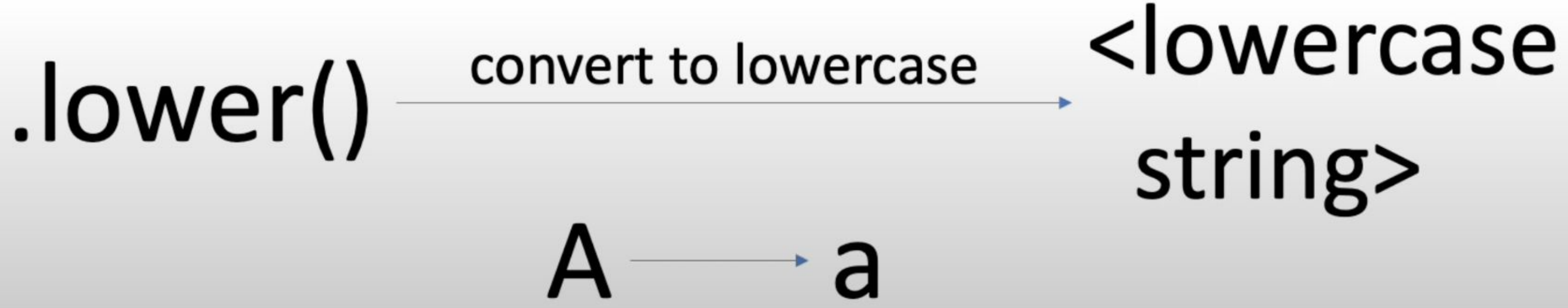
- `1 <= s.length <= 2 * 10`^{5}
- `s` consists only of printable ASCII characters.

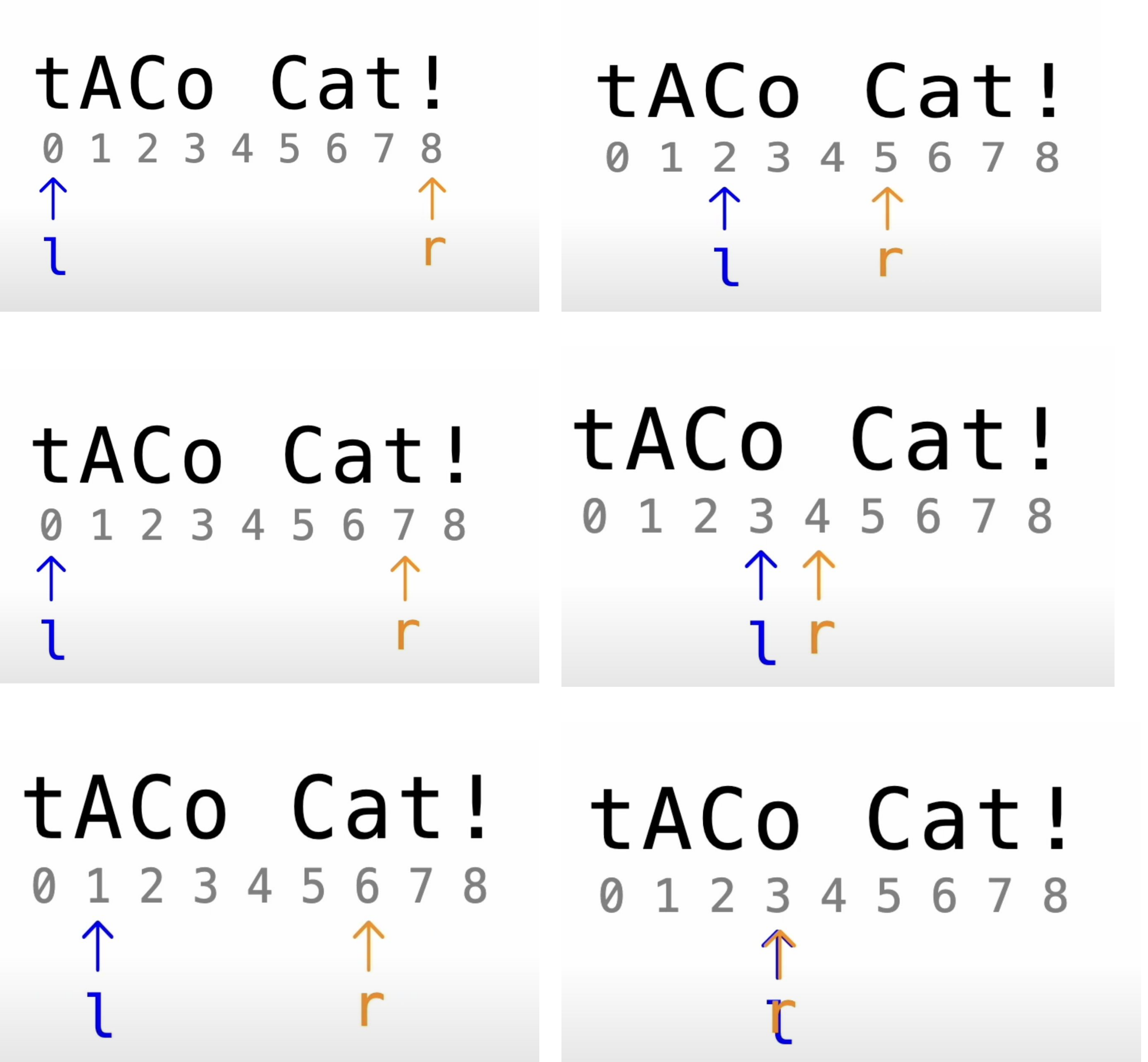
```python
class Solution:
    def isPalindrome(self, s: str) -> bool:

```

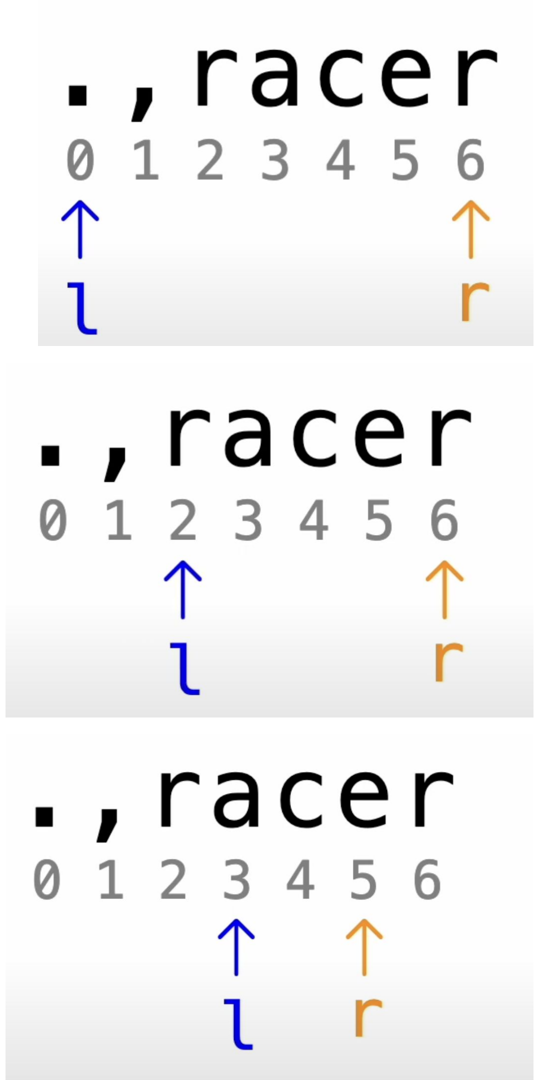
s =

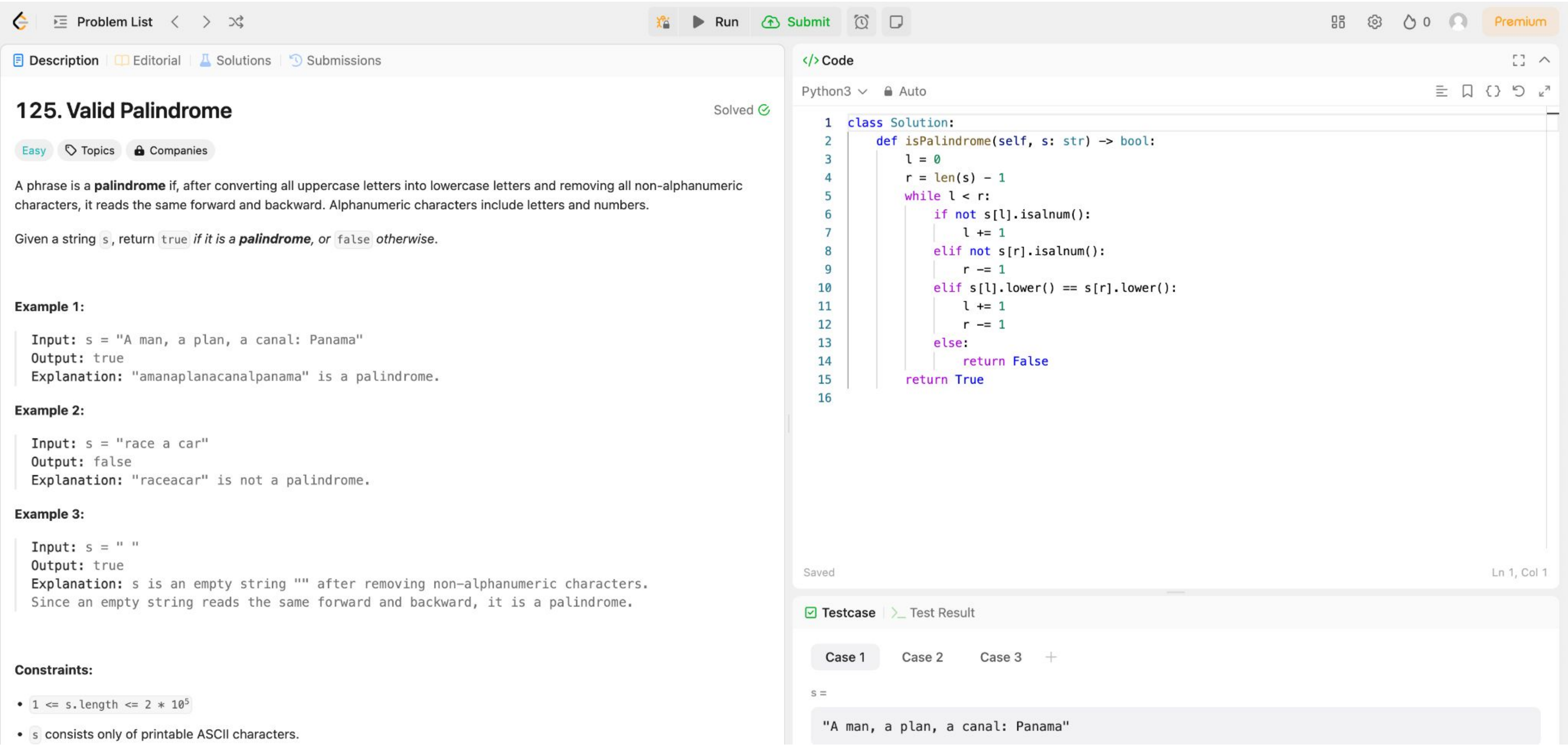
```
"A man, a plan, a canal: Panama"
```


.isalnum()
is alphanumeric?
a-z, A-Z, 0-9
yes
true
no
false


.lower()
convert to lowercase
<lowercase string>
A
a


tACo Cat!
0 1 2 3 4 5 6 7 8
l
r
tACo Cat!
0 1 2 3 4 5 6 7 8
l
r
tACo Cat!
0 1 2 3 4 5 6 7 8
l
r
tACo Cat!
0 1 2 3 4 5 6 7 8
l
r
tACo Cat!
0 1 2 3 4 5 6 7 8
l
r
tACo Cat!
0 1 2 3 4 5 6 7 8
l
r


.,racer
0 1 2 3 4 5 6
l
r
.,racer
0 1 2 3 4 5 6
l
r
.,racer
0 1 2 3 4 5 6
l
r



125. Valid Palindrome

Solved

Easy Topics Companies

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.
```

Constraints:

- `1 <= s.length <= 2 * 10`^{5}
- `s` consists only of printable ASCII characters.

```python
class Solution:
    def isPalindrome(self, s: str) -> bool:
        l = 0
        r = len(s) - 1
        while l < r:
            if not s[l].isalnum():
                l += 1
            elif not s[r].isalnum():
                r -= 1
            elif s[l].lower() == s[r].lower():
                l += 1
                r -= 1
            else:
                return False
        return True
```

Saved

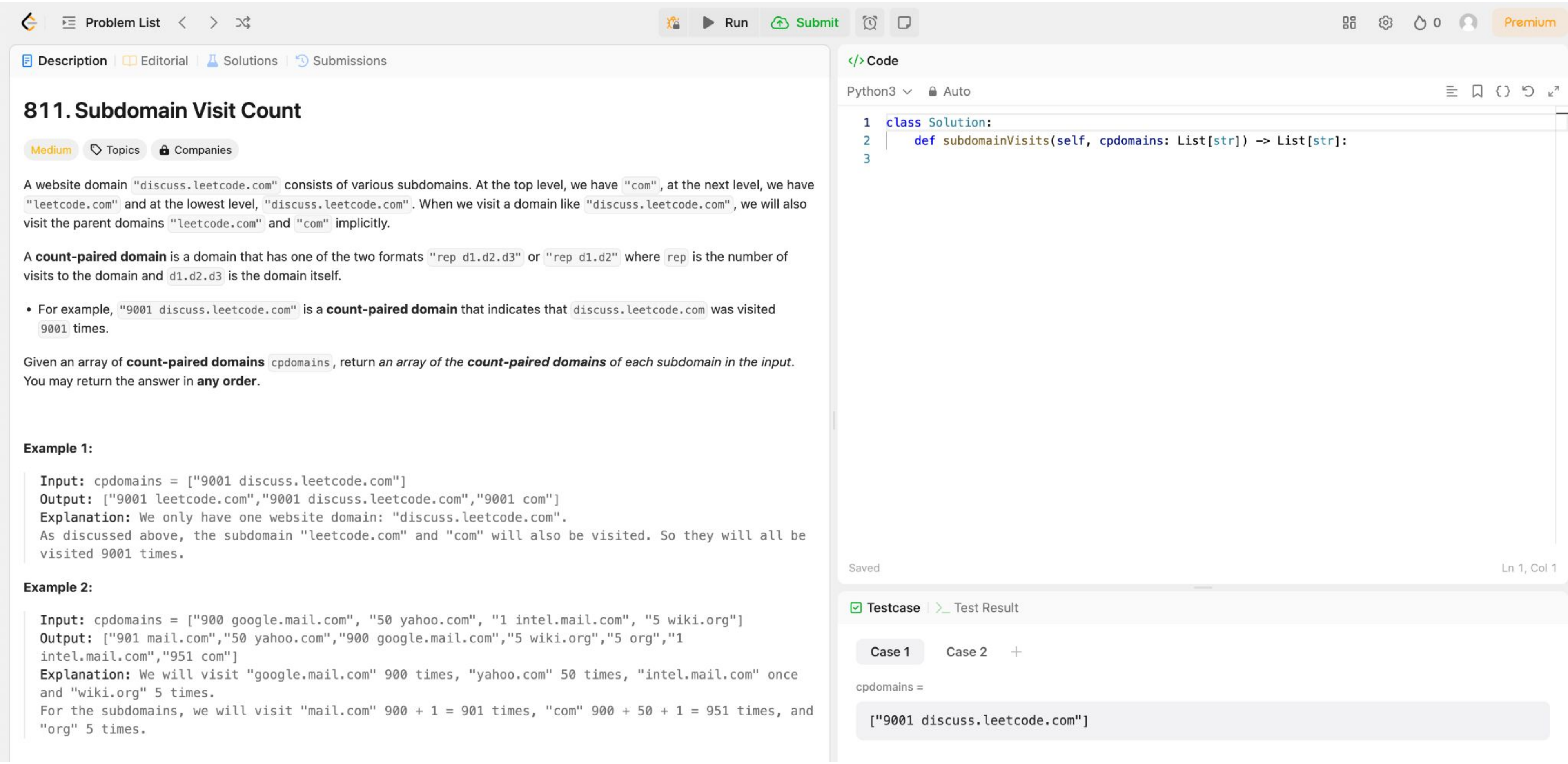
Ln 1, Col 1

Testcase Test Result

Case 1 Case 2 Case 3 +

s =

```
"A man, a plan, a canal: Panama"
```



811. Subdomain Visit Count

Medium · Topics · Companies

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.
```

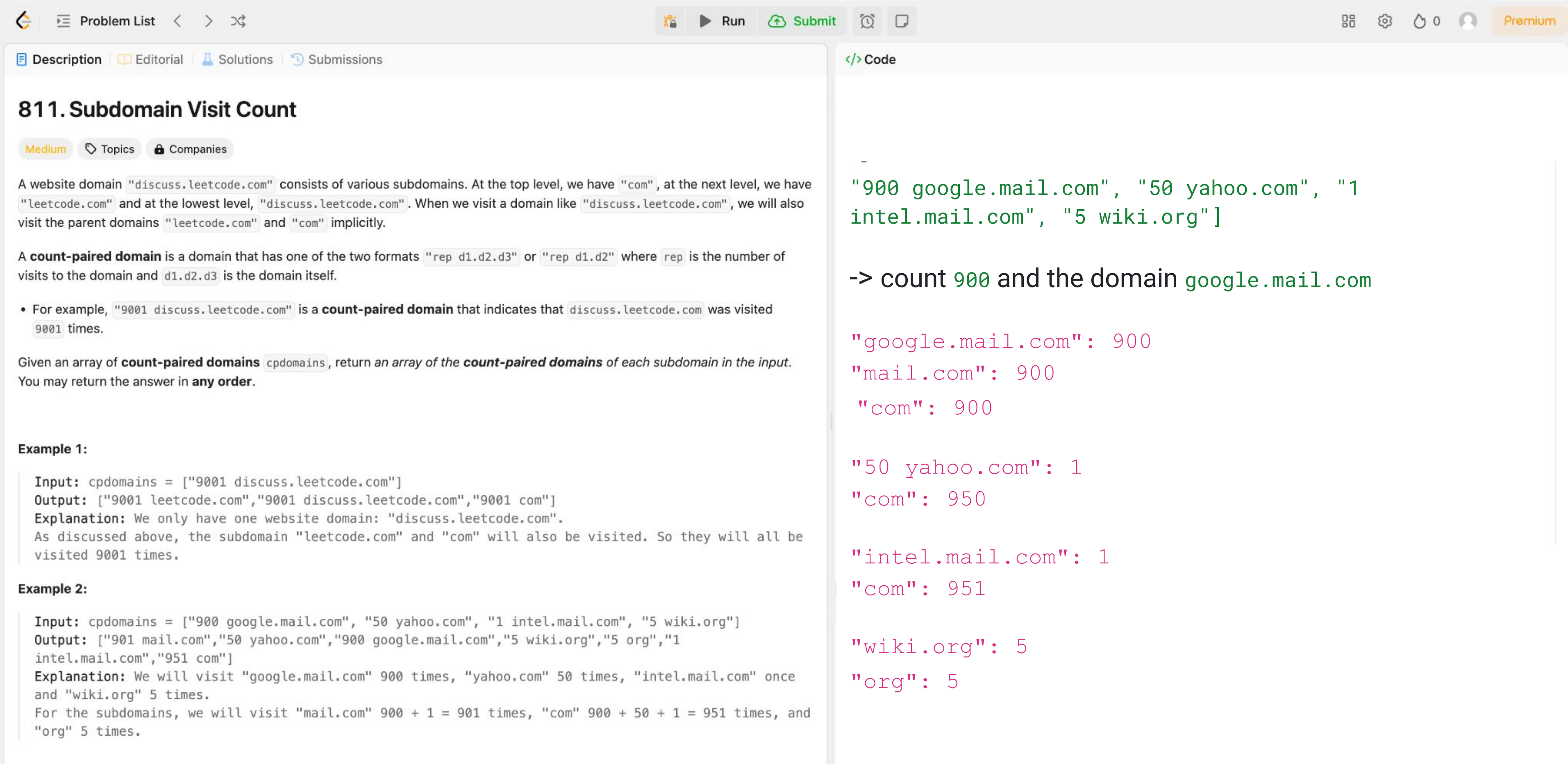
Example 2:

```
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"]
Explanation: We will visit "google.mail.com" 900 times, "yahoo.com" 50 times, "intel.mail.com" once
and "wiki.org" 5 times.
For the subdomains, we will visit "mail.com" 900 + 1 = 901 times, "com" 900 + 50 + 1 = 951 times, and
"org" 5 times.
```

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

cpdomains =

```
["9001 discuss.leetcode.com"]
```



811. Subdomain Visit Count

Medium Topics Companies

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.
```

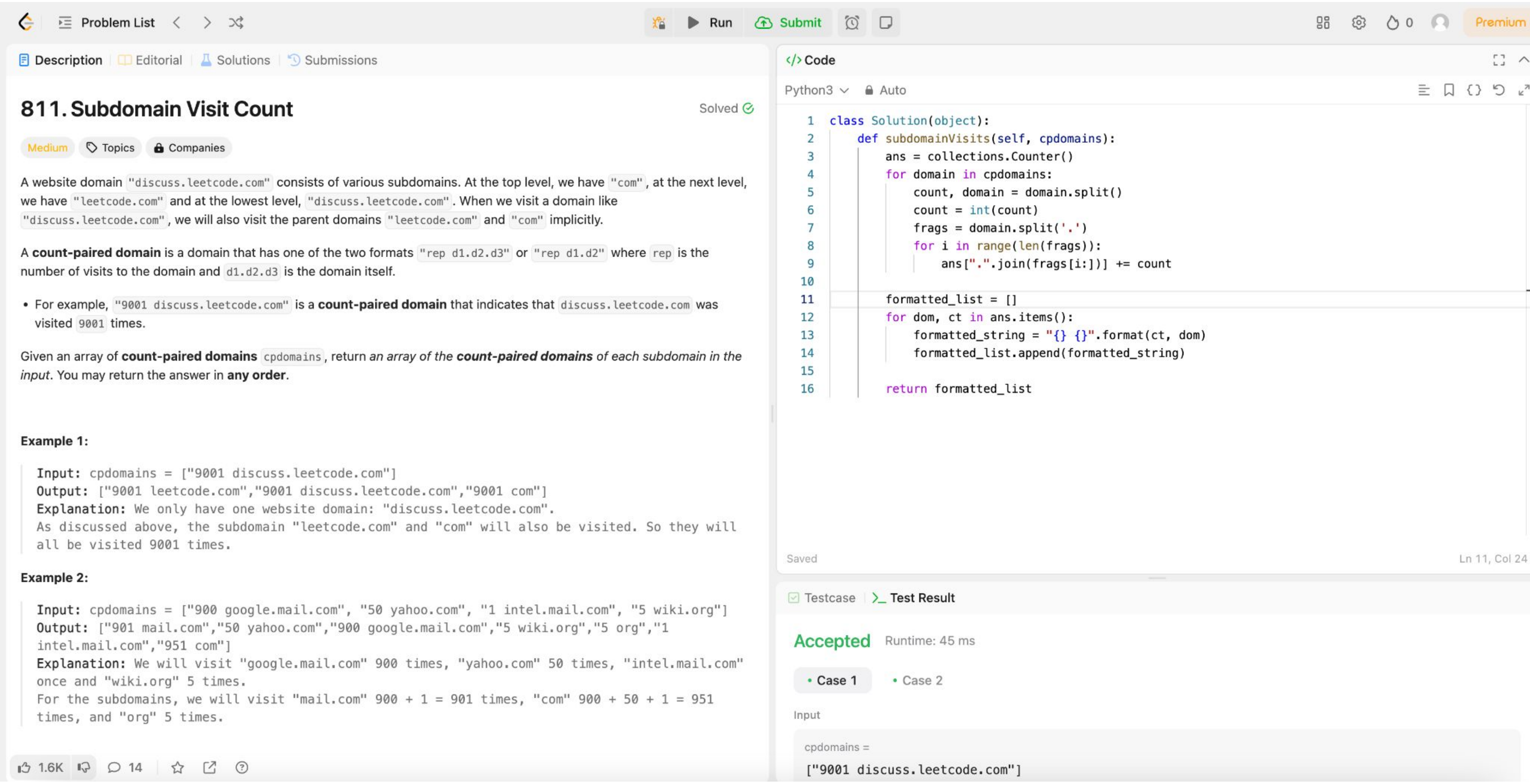
Example 2:

```
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"]
Explanation: We will visit "google.mail.com" 900 times, "yahoo.com" 50 times, "intel.mail.com" once
and "wiki.org" 5 times.
For the subdomains, we will visit "mail.com" 900 + 1 = 901 times, "com" 900 + 50 + 1 = 951 times, and
"org" 5 times.
```

`"900 google.mail.com", "50 yahoo.com", "1 intel.mail.com", "5 wiki.org"]`

-> count `900` and the domain `google.mail.com`

```
"google.mail.com": 900
"mail.com": 900
 "com": 900

"50 yahoo.com": 1
"com": 950

"intel.mail.com": 1
"com": 951

"wiki.org": 5
"org": 5
```



811. Subdomain Visit Count

Solved

Medium Topics Companies

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"]
Explanation: We will visit "google.mail.com" 900 times, "yahoo.com" 50 times, "intel.mail.com"
once and "wiki.org" 5 times.
For the subdomains, we will visit "mail.com" 900 + 1 = 901 times, "com" 900 + 50 + 1 = 951
times, and "org" 5 times.
```

1.6K 14

```python
class Solution(object):
    def subdomainVisits(self, cpdomains):
        ans = collections.Counter()
        for domain in cpdomains:
            count, domain = domain.split()
            count = int(count)
            frags = domain.split('.')
            for i in range(len(frags)):
                ans[".".join(frags[i:])] += count

        formatted_list = []
        for dom, ct in ans.items():
            formatted_string = "{} {}".format(ct, dom)
            formatted_list.append(formatted_string)

        return formatted_list
```

Saved Ln 11, Col 24

Testcase Test Result

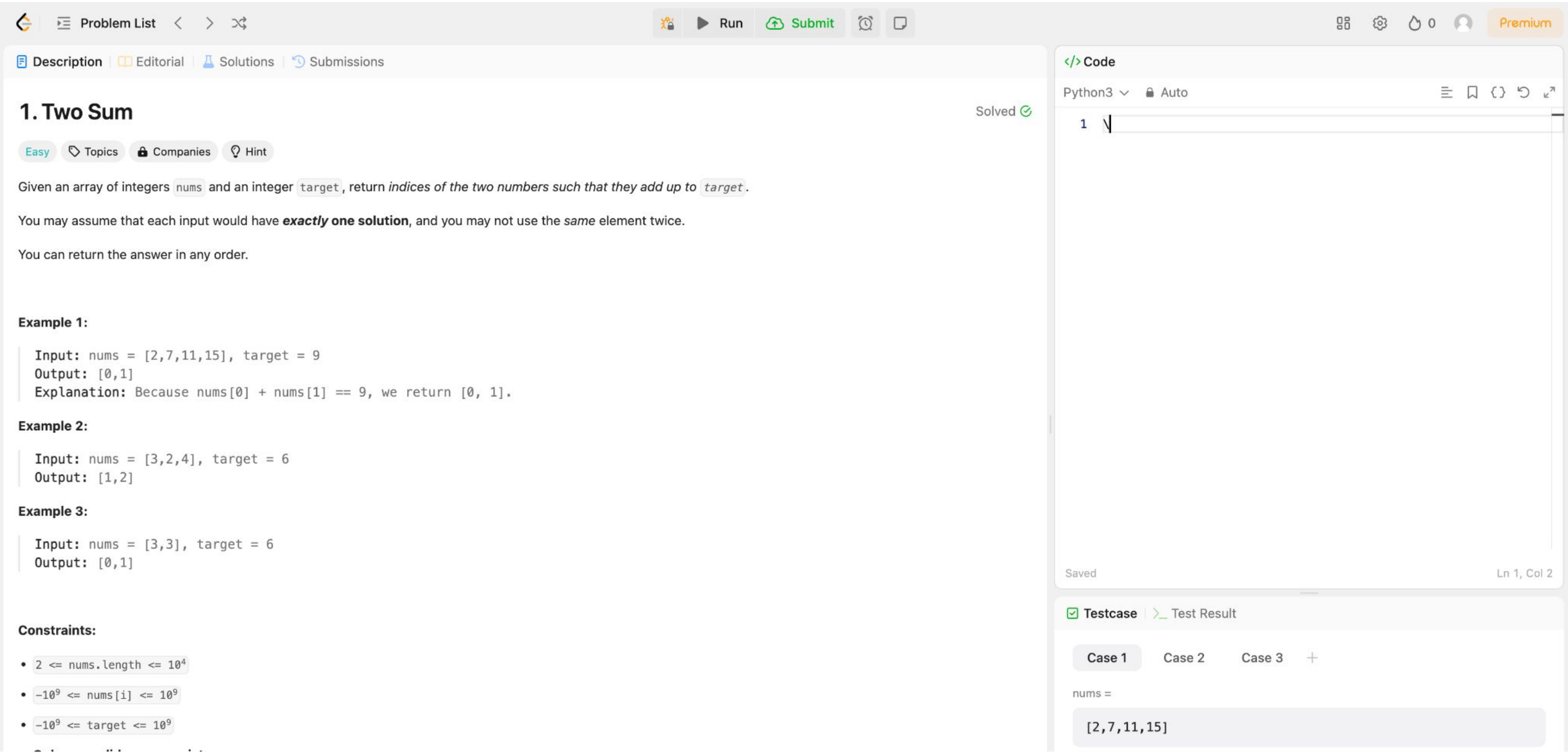
Accepted Runtime: 45 ms

Case 1 Case 2

Input

cpdomains =

["9001 discuss.leetcode.com"]



1. Two Sum

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

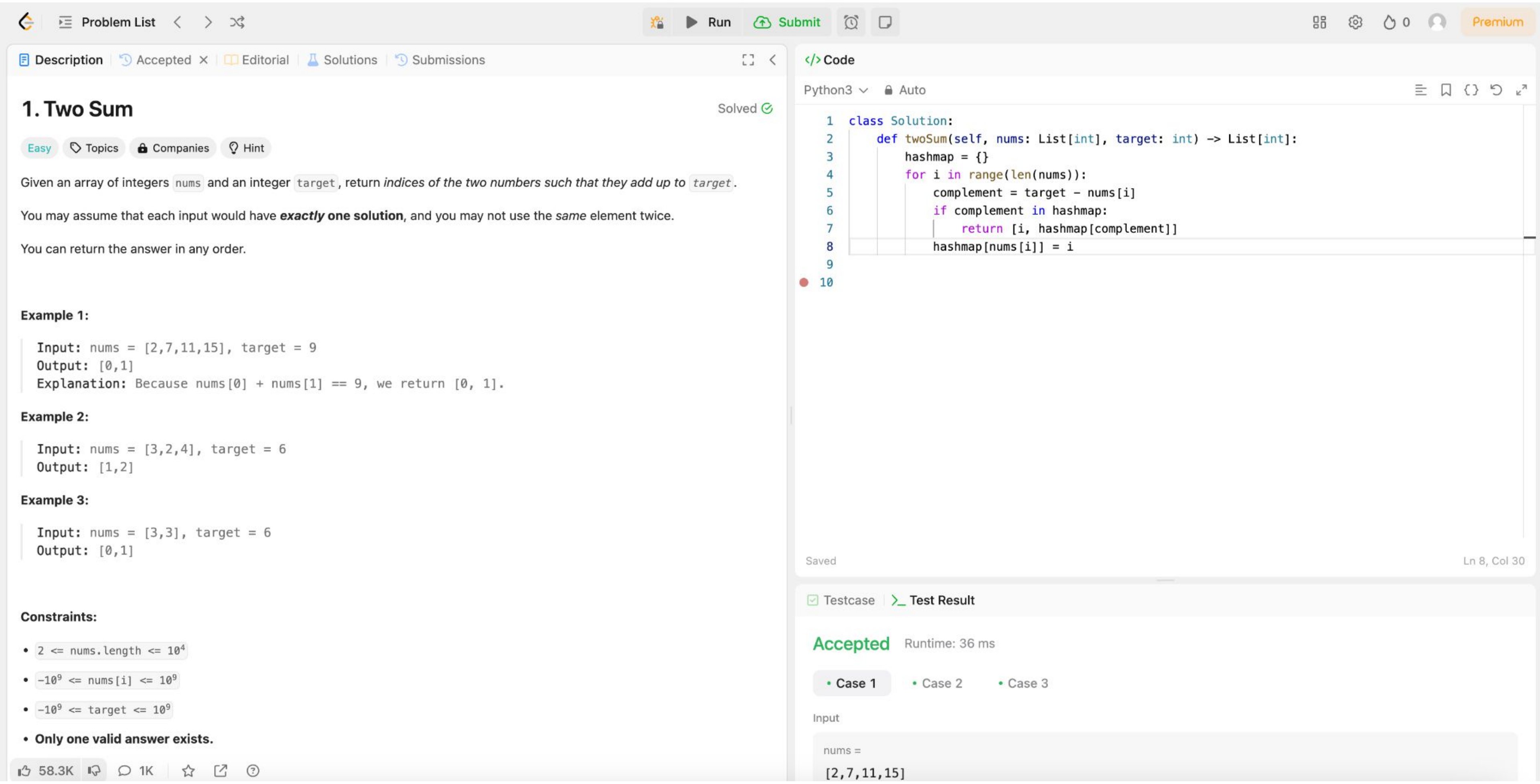
Constraints:

- $2 <= nums.length <= 10^4$
- $-10^9 <= nums[i] <= 10^9$
- $-10^9 <= target <= 10^9$

```
\
```

nums =

```
[2,7,11,15]
```



1. Two Sum

Solved

Easy · Topics · Companies · Hint

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

Constraints:

- $2 <= nums.length <= 10^4$
- $-10^9 <= nums[i] <= 10^9$
- $-10^9 <= target <= 10^9$
- **Only one valid answer exists.**

58.3K · 1K

```python
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        hashmap = {}
        for i in range(len(nums)):
            complement = target - nums[i]
            if complement in hashmap:
                return [i, hashmap[complement]]
            hashmap[nums[i]] = i


```

Saved · Ln 8, Col 30

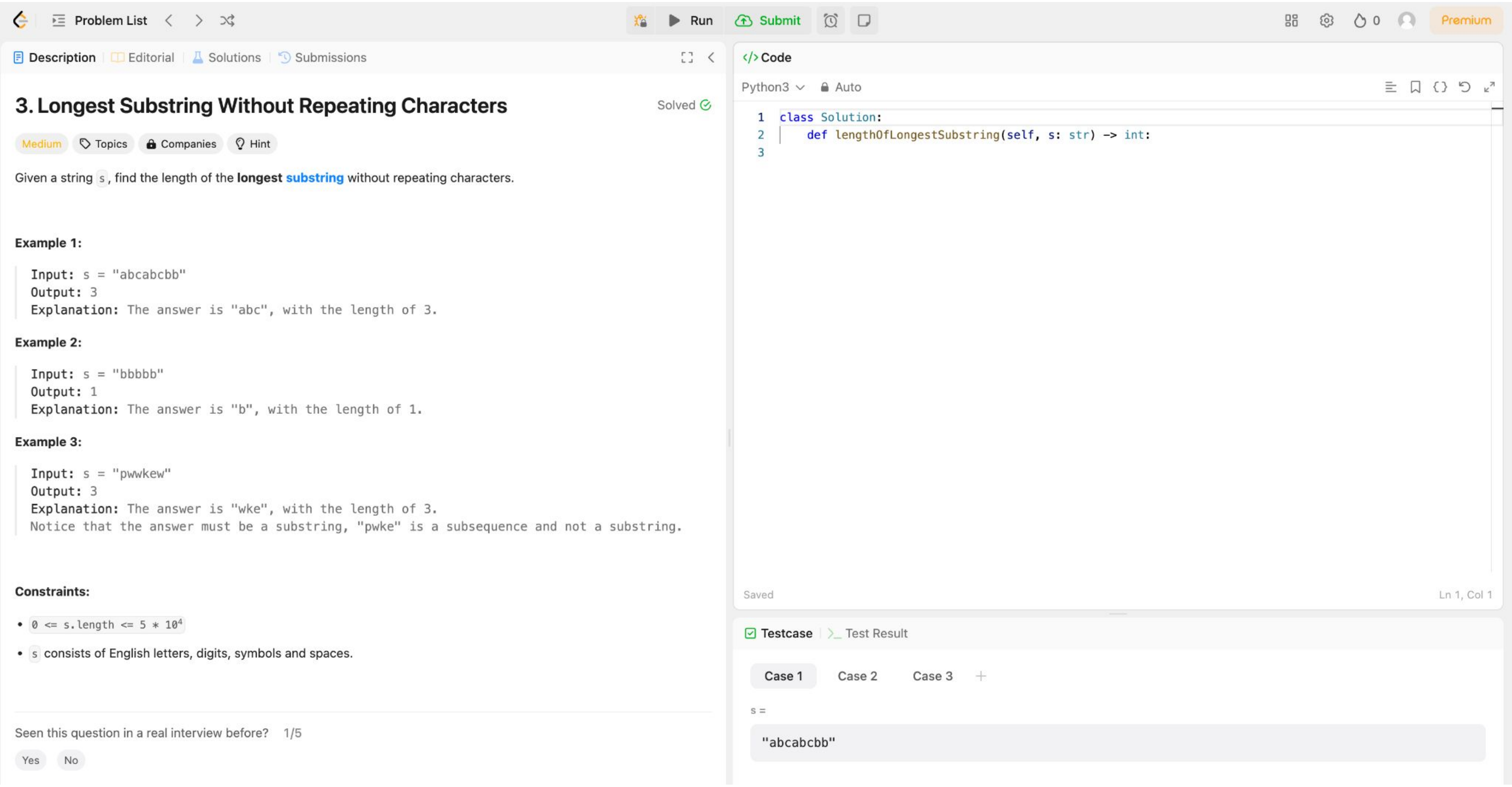
Accepted Runtime: 36 ms

Case 1 · Case 2 · Case 3

Input

nums =

[2,7,11,15]



3. Longest Substring Without Repeating Characters

Solved

Medium · Topics · Companies · Hint

Given a string `s`, find the length of the **longest** **substring** without repeating characters.

Example 1:

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.
```

Constraints:

- `0 <= s.length <= 5 * 10`^{4}
- `s` consists of English letters, digits, symbols and spaces.

Seen this question in a real interview before? 1/5

Yes No

```python
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:

```

Saved Ln 1, Col 1

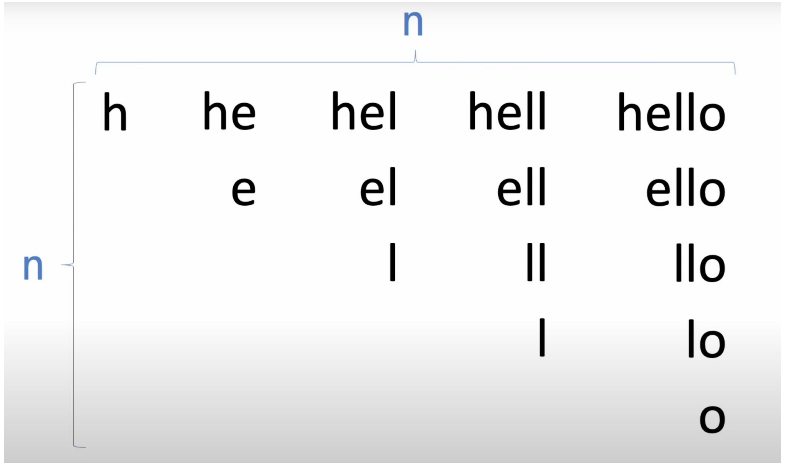
Testcase | Test Result

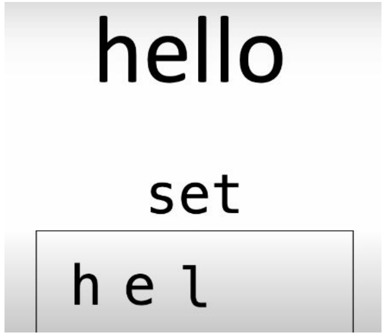
Case 1 Case 2 Case 3 +

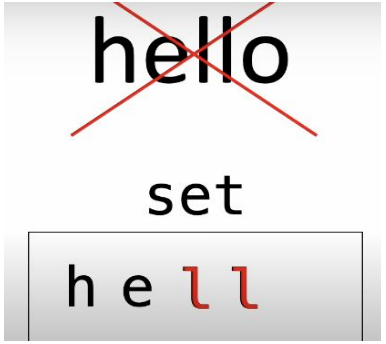
s =

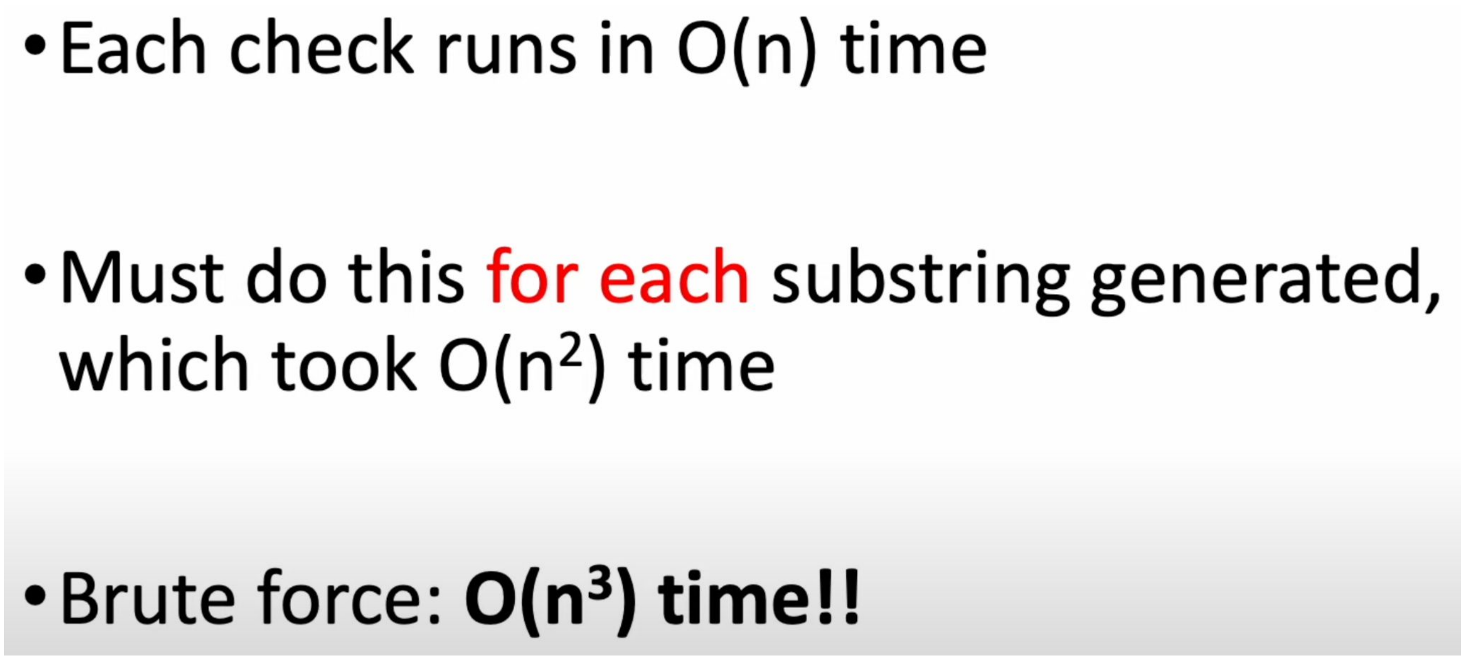
```
"abcabcbb"
```

Input - Hello
Output - Hel



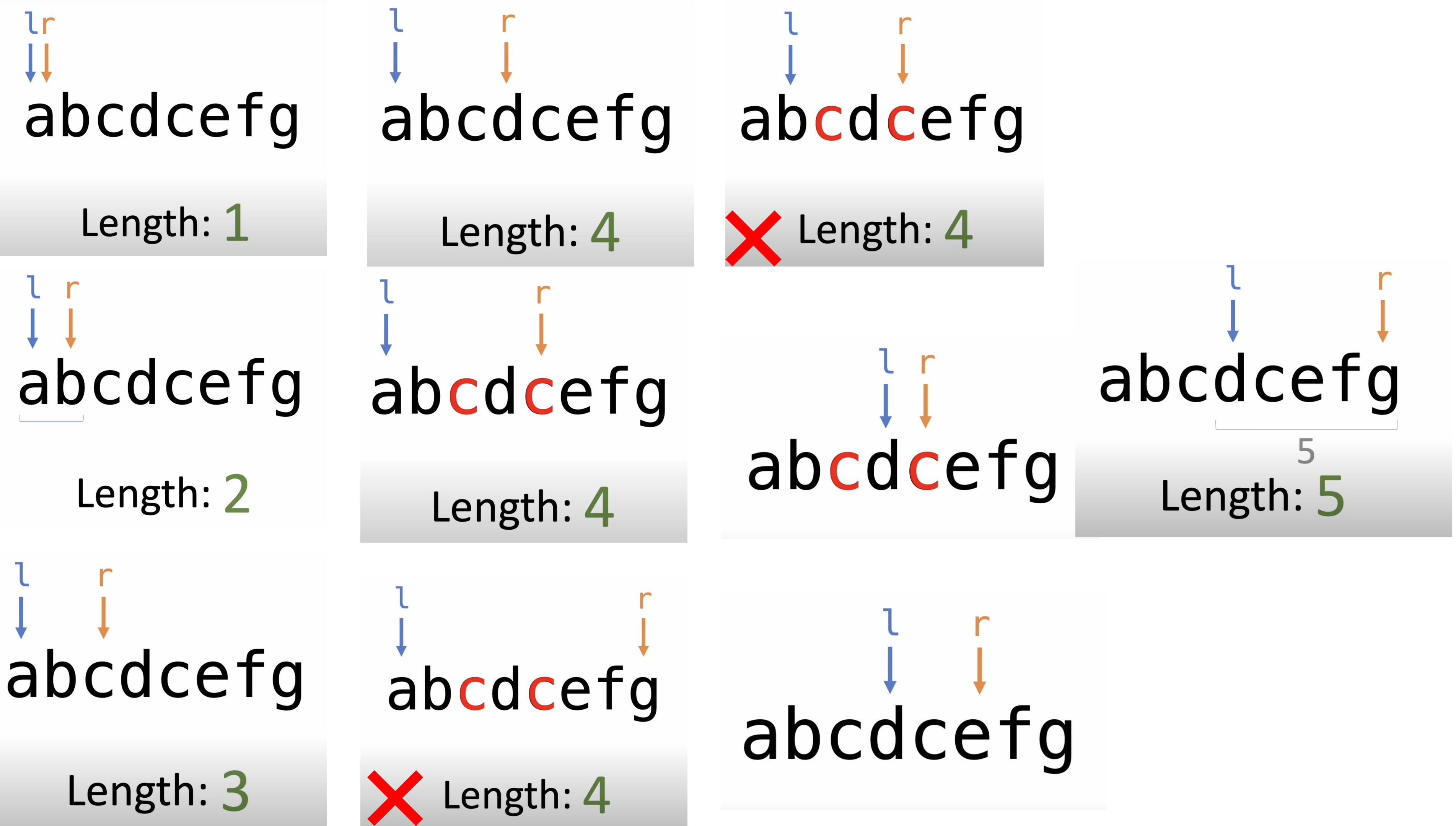






- Each check runs in $O(n)$ time
- Must do this for each substring generated, which took $O(n^2)$ time
- Brute force: **$O(n^3)$ time!!**

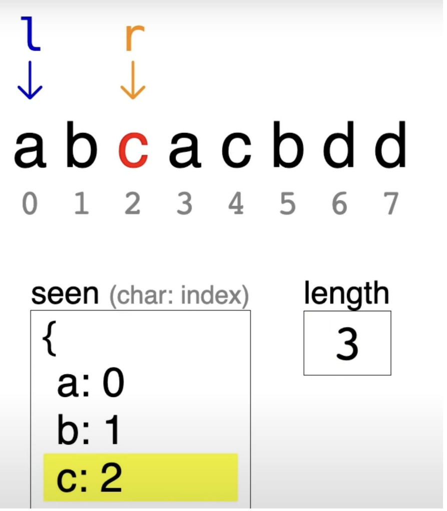
Q - How can we Optimize?

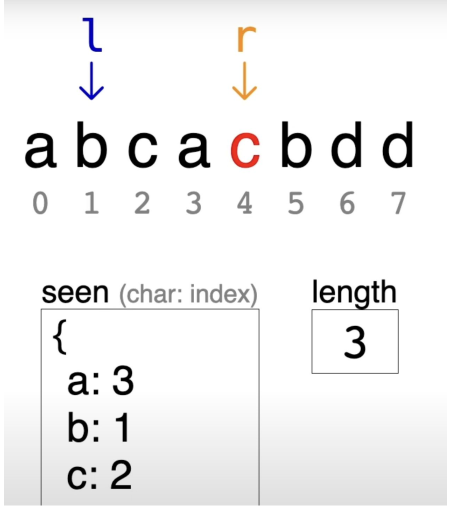


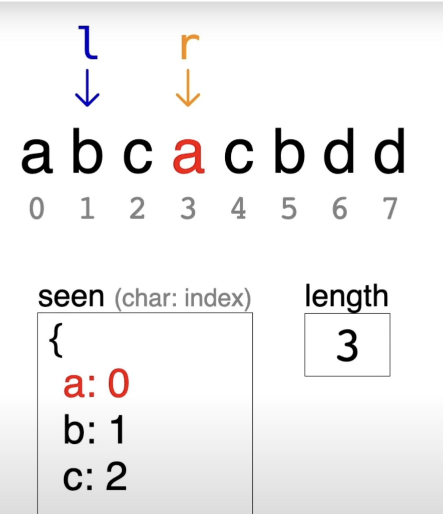
l r
abcdcefg
Length: 1
l r
abcdcefg
Length: 4
l r
abcdcefg
Length: 4
l r
abcdcefg
Length: 2
l r
abcdcefg
Length: 4
l r
abcdcefg
l r
abcdcefg
5
Length: 5
l r
abcdcefg
Length: 3
l r
abcdcefg
Length: 4
l r
abcdcefg

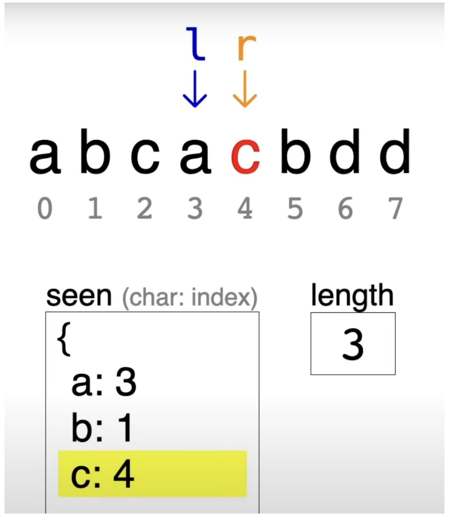
Algorithm - Keep moving the right pointer to extend the substring until we reach a repeated character. At that point, move the left pointer up until the repeated character is gone. We keep repeating the character until the right pointer reaches the end of the string.

How do we know where to update the left pointer when a repeated character is found?

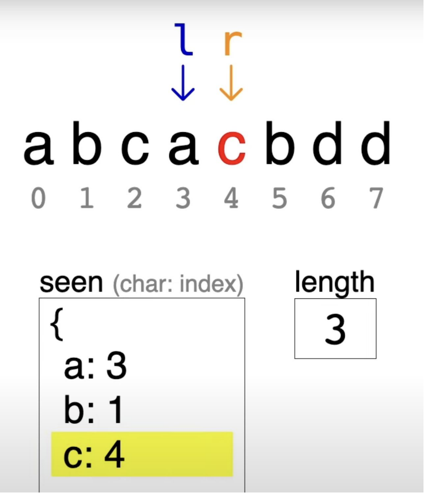


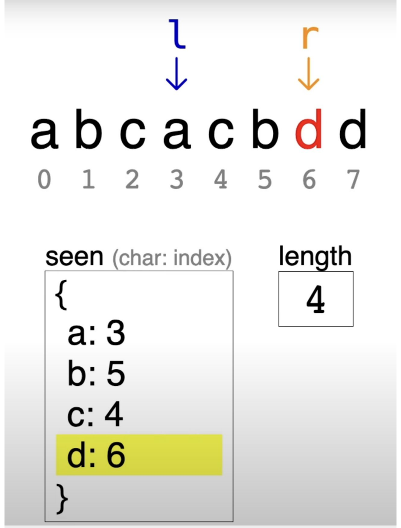


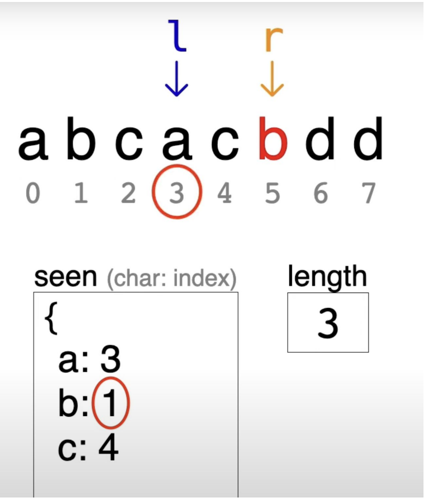


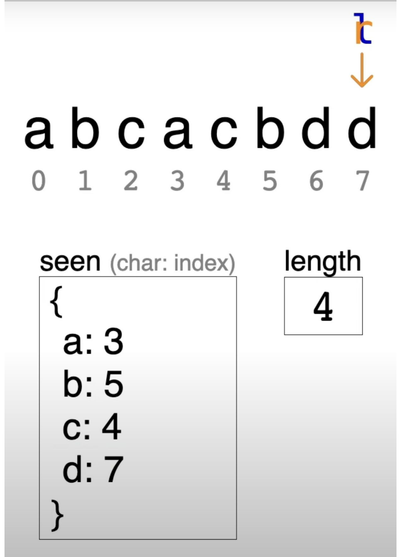


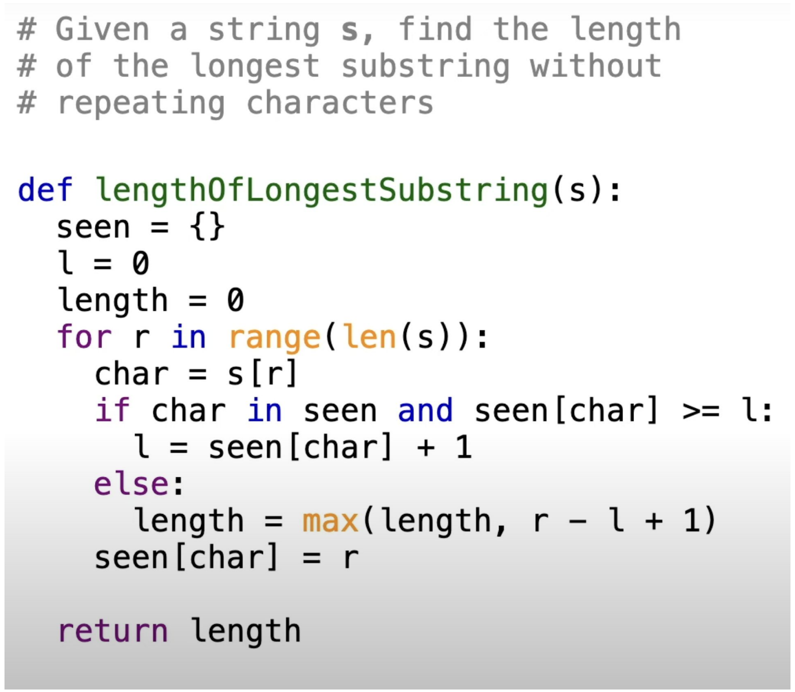
```python
# Given a string s, find the length
# of the longest substring without
# repeating characters

def lengthOfLongestSubstring(s):
  seen = {}
  l = 0
  length = 0
  for r in range(len(s)):
    char = s[r]
    if char in seen and seen[char] >= l:
      l = seen[char] + 1
    else:
      length = max(length, r - l + 1)
    seen[char] = r

  return length
```



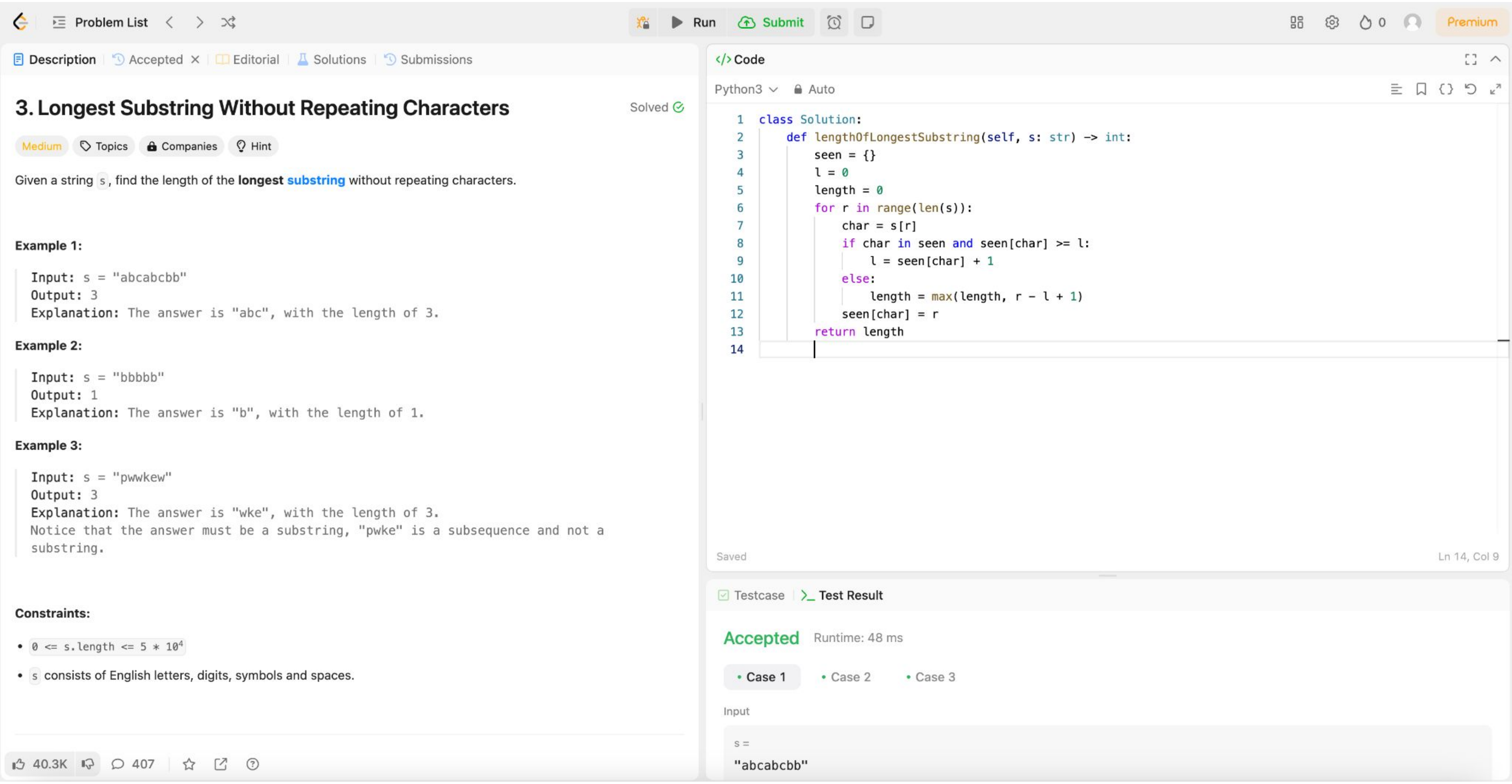








```python
# Given a string s, find the length
# of the longest substring without
# repeating characters

def lengthOfLongestSubstring(s):
  seen = {}
  l = 0
  length = 0
  for r in range(len(s)):
    char = s[r]
    if char in seen and seen[char] >= l:
      l = seen[char] + 1
    else:
      length = max(length, r - l + 1)
    seen[char] = r

  return length
```



3. Longest Substring Without Repeating Characters

Solved

Given a string `s`, find the length of the **longest substring** without repeating characters.

Example 1:

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.
```

Constraints:

- `0 <= s.length <= 5 * 10^4`
- `s` consists of English letters, digits, symbols and spaces.

```python
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        seen = {}
        l = 0
        length = 0
        for r in range(len(s)):
            char = s[r]
            if char in seen and seen[char] >= l:
                l = seen[char] + 1
            else:
                length = max(length, r - l + 1)
            seen[char] = r
        return length

```

Saved

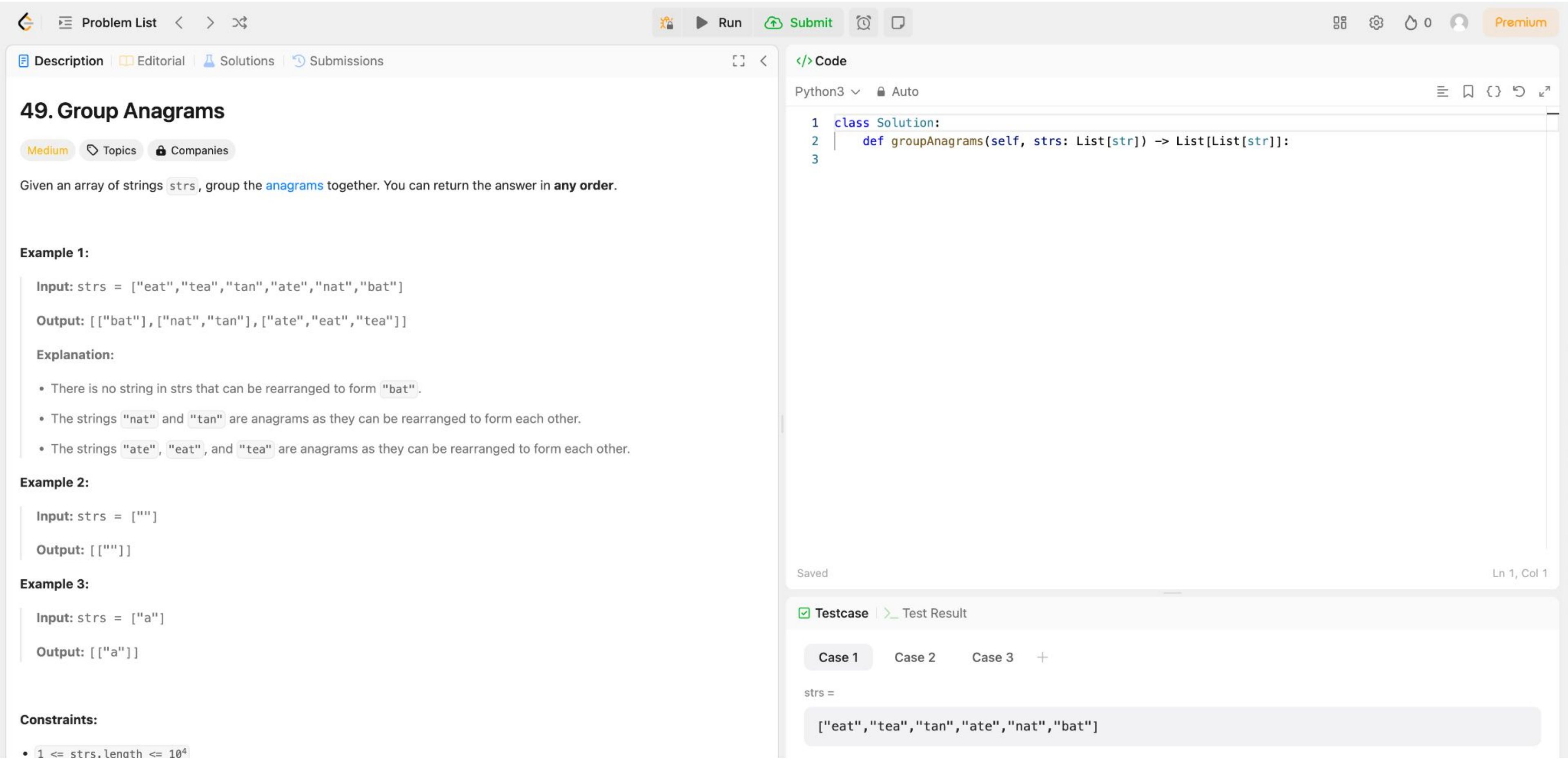
Ln 14, Col 9

Accepted Runtime: 48 ms

Input

s =

"abcabcbb"



49. Group Anagrams

Medium Topics Companies

Given an array of strings `strs`, group the anagrams together. You can return the answer in **any order**.

Example 1:

Input: strs = ["eat","tea","tan","ate","nat","bat"]

Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

Explanation:

- There is no string in strs that can be rearranged to form `"bat"`.
- The strings `"nat"` and `"tan"` are anagrams as they can be rearranged to form each other.
- The strings `"ate"`, `"eat"`, and `"tea"` are anagrams as they can be rearranged to form each other.

Example 2:

Input: strs = [""]

Output: [[""]]

Example 3:

Input: strs = ["a"]

Output: [["a"]]

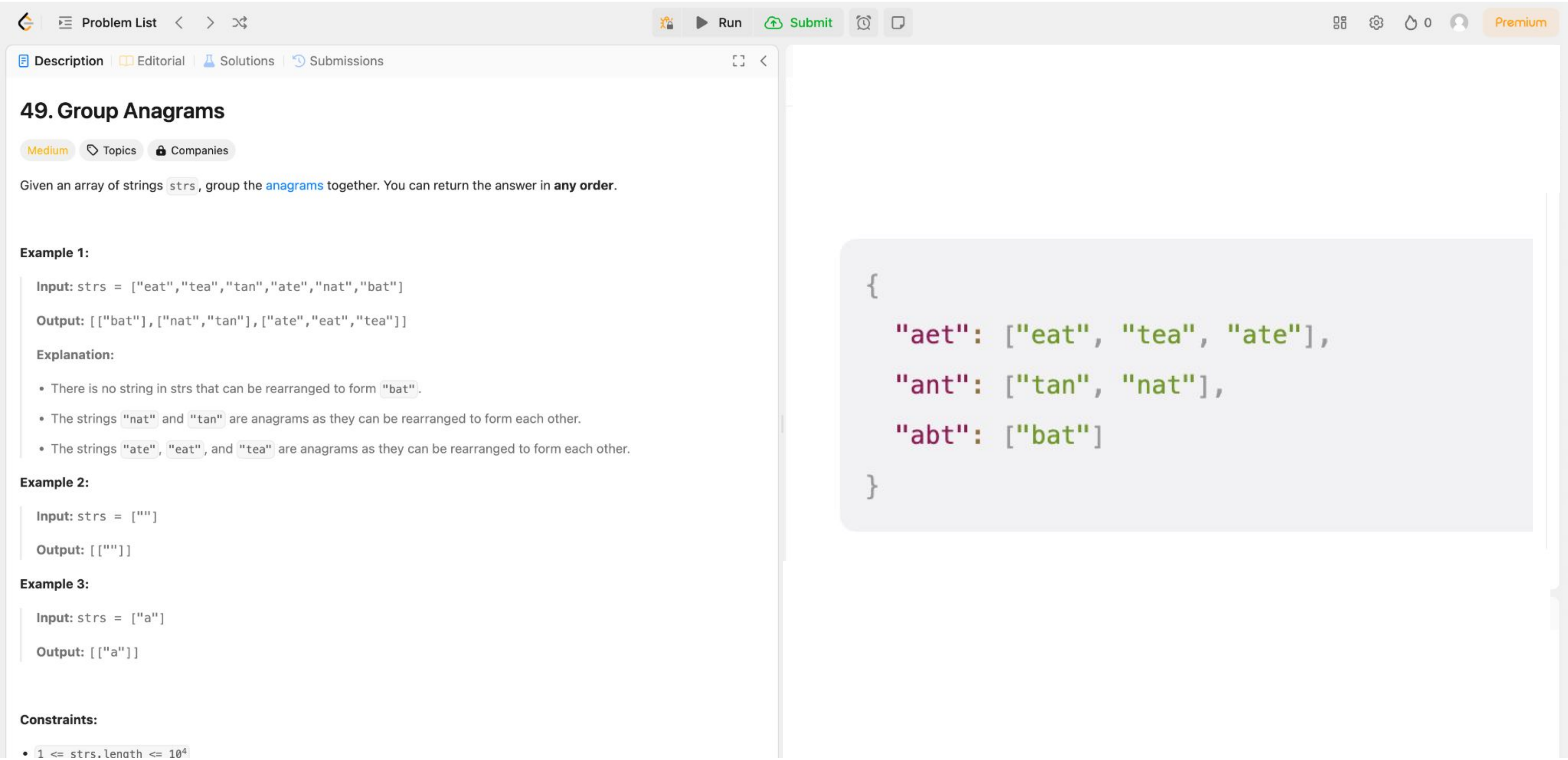
Constraints:

- `1 <= strs.length <= 10`^{4}

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

```

strs =

["eat","tea","tan","ate","nat","bat"]



49. Group Anagrams

Medium Topics Companies

Given an array of strings `strs`, group the anagrams together. You can return the answer in **any order**.

Example 1:

Input: `strs = ["eat","tea","tan","ate","nat","bat"]`

Output: `[["bat"],["nat","tan"],["ate","eat","tea"]]`

Explanation:

- There is no string in strs that can be rearranged to form `"bat"`.
- The strings `"nat"` and `"tan"` are anagrams as they can be rearranged to form each other.
- The strings `"ate"`, `"eat"`, and `"tea"` are anagrams as they can be rearranged to form each other.

Example 2:

Input: `strs = [""]`

Output: `[[""]]`

Example 3:

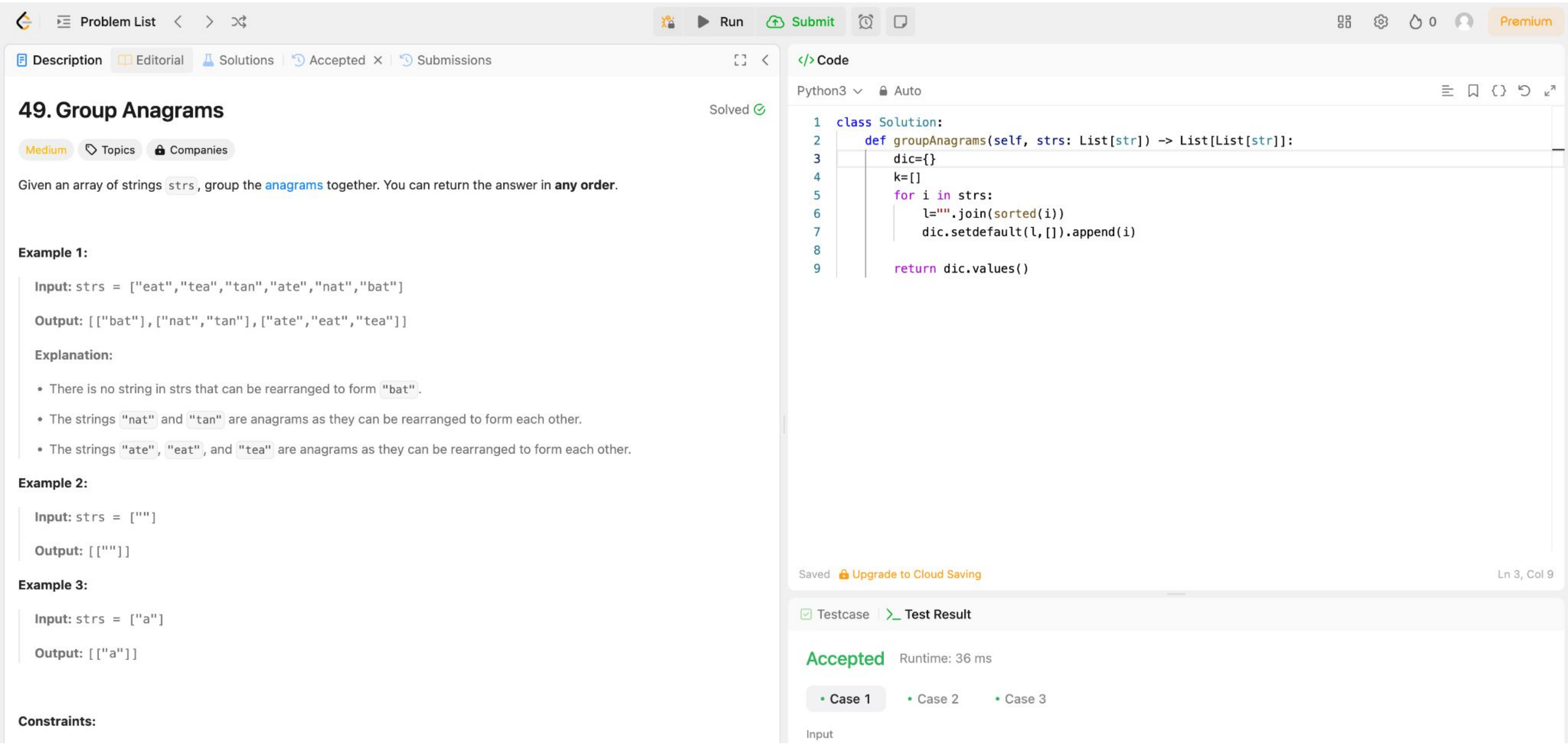
Input: `strs = ["a"]`

Output: `[["a"]]`

Constraints:

- `1 <= strs.length <=` 10^4

```
{
  "aet": ["eat", "tea", "ate"],
  "ant": ["tan", "nat"],
  "abt": ["bat"]
}
```



49. Group Anagrams

Solved

Medium · Topics · Companies

Given an array of strings `strs`, group the anagrams together. You can return the answer in **any order**.

Example 1:

Input: `strs = ["eat","tea","tan","ate","nat","bat"]`

Output: `[["bat"],["nat","tan"],["ate","eat","tea"]]`

Explanation:

- There is no string in strs that can be rearranged to form `"bat"`.
- The strings `"nat"` and `"tan"` are anagrams as they can be rearranged to form each other.
- The strings `"ate"`, `"eat"`, and `"tea"` are anagrams as they can be rearranged to form each other.

Example 2:

Input: `strs = [""]`

Output: `[[""]]`

Example 3:

Input: `strs = ["a"]`

Output: `[["a"]]`

Constraints:

Python3 · Auto

```python
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        dic={}
        k=[]
        for i in strs:
            l="".join(sorted(i))
            dic.setdefault(l,[]).append(i)

        return dic.values()
```

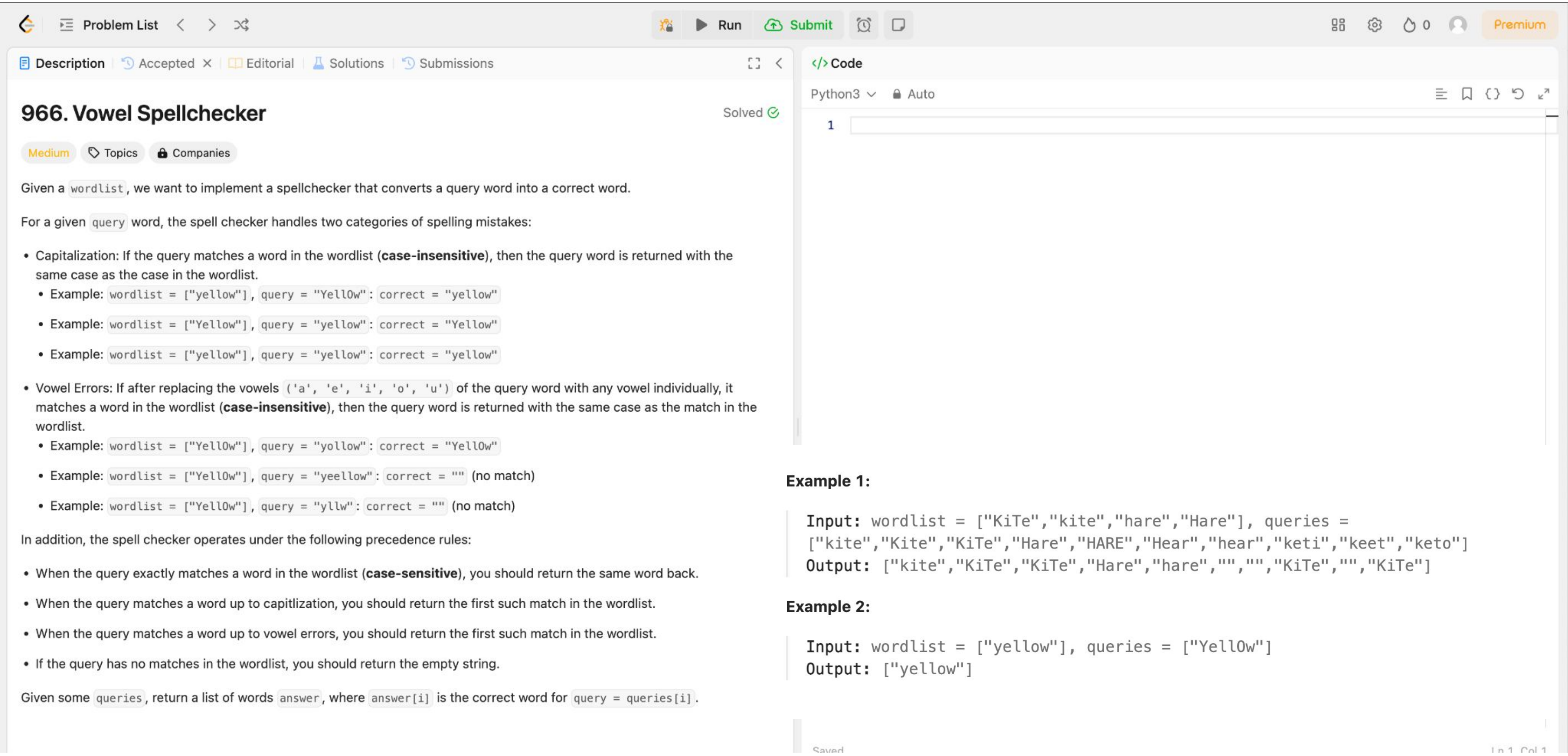
Saved · Upgrade to Cloud Saving · Ln 3, Col 9

Testcase · Test Result

Accepted Runtime: 36 ms

Case 1 · Case 2 · Case 3

Input



966. Vowel Spellchecker

Given a `wordlist`, we want to implement a spellchecker that converts a query word into a correct word.

For a given `query` word, the spell checker handles two categories of spelling mistakes:

- Capitalization: If the query matches a word in the wordlist (**case-insensitive**), then the query word is returned with the same case as the case in the wordlist.
  - Example: `wordlist = ["yellow"]`, `query = "YellOw"`: `correct = "yellow"`
  - Example: `wordlist = ["Yellow"]`, `query = "yellow"`: `correct = "Yellow"`
  - Example: `wordlist = ["yellow"]`, `query = "yellow"`: `correct = "yellow"`
- Vowel Errors: If after replacing the vowels `('a', 'e', 'i', 'o', 'u')` of the query word with any vowel individually, it matches a word in the wordlist (**case-insensitive**), then the query word is returned with the same case as the match in the wordlist.
  - Example: `wordlist = ["YellOw"]`, `query = "yollow"`: `correct = "YellOw"`
  - Example: `wordlist = ["YellOw"]`, `query = "yeellow"`: `correct = ""` (no match)
  - Example: `wordlist = ["YellOw"]`, `query = "yllw"`: `correct = ""` (no match)

In addition, the spell checker operates under the following precedence rules:

- When the query exactly matches a word in the wordlist (**case-sensitive**), you should return the same word back.
- When the query matches a word up to capitlization, you should return the first such match in the wordlist.
- When the query matches a word up to vowel errors, you should return the first such match in the wordlist.
- If the query has no matches in the wordlist, you should return the empty string.

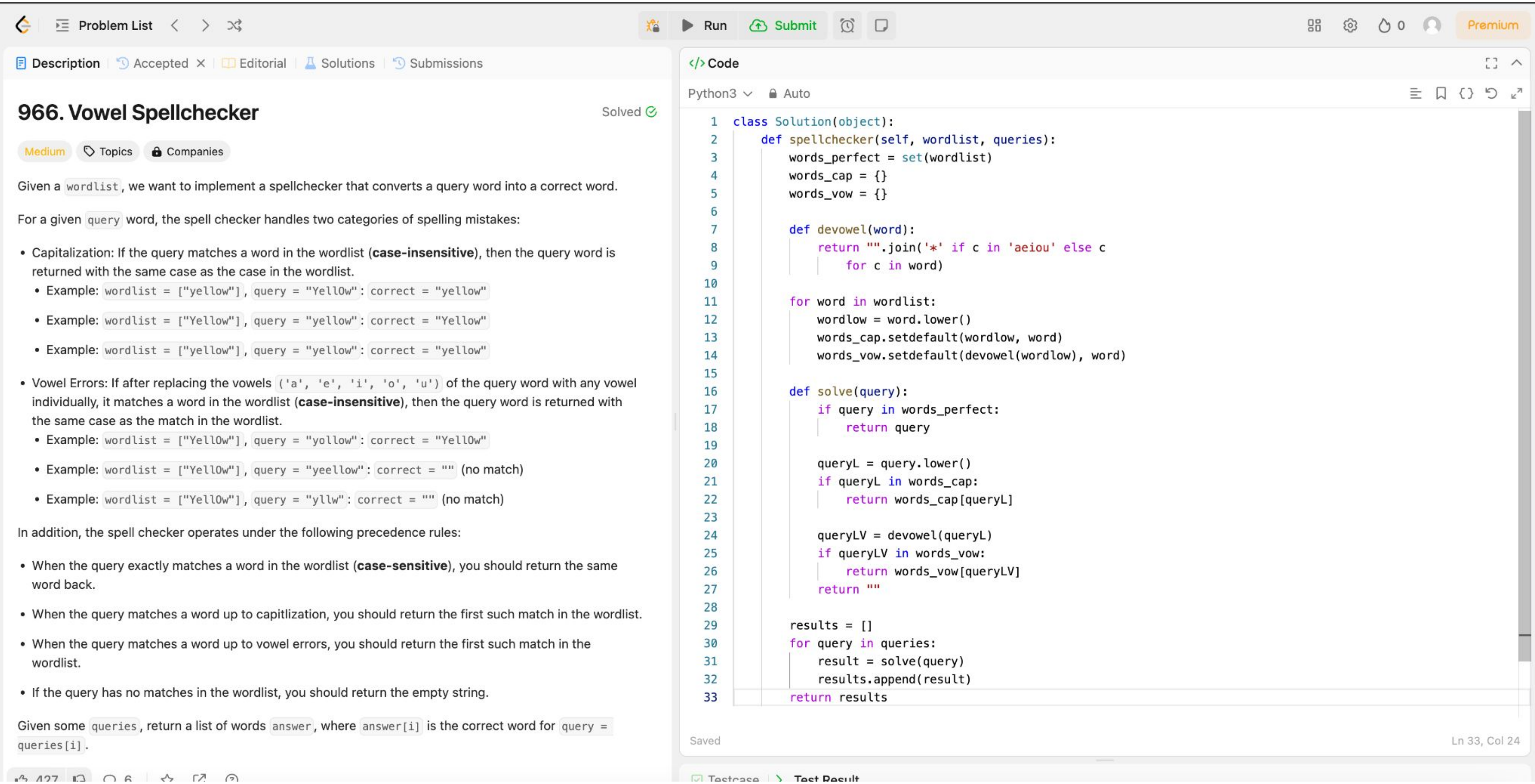
Given some `queries`, return a list of words `answer`, where `answer[i]` is the correct word for `query = queries[i]`.

Example 1:

```
Input: wordlist = ["KiTe","kite","hare","Hare"], queries =
["kite","Kite","KiTe","Hare","HARE","Hear","hear","keti","keet","keto"]
Output: ["kite","KiTe","KiTe","Hare","hare","","","KiTe","","KiTe"]
```

Example 2:

```
Input: wordlist = ["yellow"], queries = ["YellOw"]
Output: ["yellow"]
```



966. Vowel Spellchecker

Solved

Medium Topics Companies

Given a `wordlist`, we want to implement a spellchecker that converts a query word into a correct word.

For a given `query` word, the spell checker handles two categories of spelling mistakes:

- Capitalization: If the query matches a word in the wordlist (**case-insensitive**), then the query word is returned with the same case as the case in the wordlist.
  - Example: `wordlist = ["yellow"]`, `query = "YellOw"`: `correct = "yellow"`
  - Example: `wordlist = ["Yellow"]`, `query = "yellow"`: `correct = "Yellow"`
  - Example: `wordlist = ["yellow"]`, `query = "yellow"`: `correct = "yellow"`
- Vowel Errors: If after replacing the vowels `('a', 'e', 'i', 'o', 'u')` of the query word with any vowel individually, it matches a word in the wordlist (**case-insensitive**), then the query word is returned with the same case as the match in the wordlist.
  - Example: `wordlist = ["YellOw"]`, `query = "yollow"`: `correct = "YellOw"`
  - Example: `wordlist = ["YellOw"]`, `query = "yeellow"`: `correct = ""` (no match)
  - Example: `wordlist = ["YellOw"]`, `query = "yllw"`: `correct = ""` (no match)

In addition, the spell checker operates under the following precedence rules:

- When the query exactly matches a word in the wordlist (**case-sensitive**), you should return the same word back.
- When the query matches a word up to capitlization, you should return the first such match in the wordlist.
- When the query matches a word up to vowel errors, you should return the first such match in the wordlist.
- If the query has no matches in the wordlist, you should return the empty string.

Given some `queries`, return a list of words `answer`, where `answer[i]` is the correct word for `query = queries[i]`.

```python
class Solution(object):
    def spellchecker(self, wordlist, queries):
        words_perfect = set(wordlist)
        words_cap = {}
        words_vow = {}

        def devowel(word):
            return "".join('*' if c in 'aeiou' else c
                for c in word)

        for word in wordlist:
            wordlow = word.lower()
            words_cap.setdefault(wordlow, word)
            words_vow.setdefault(devowel(wordlow), word)

        def solve(query):
            if query in words_perfect:
                return query

            queryL = query.lower()
            if queryL in words_cap:
                return words_cap[queryL]

            queryLV = devowel(queryL)
            if queryLV in words_vow:
                return words_vow[queryLV]
            return ""

        results = []
        for query in queries:
            result = solve(query)
            results.append(result)
        return results
```

Saved Ln 33, Col 24