#	Title	Solution	Acceptance	Difficulty	Frequency
408	Valid Word Abbreviation 🔒		28.3%	Easy	
527	Word Abbreviation 🔒		44.0%	Hard	
320	Generalized Abbreviation 🔒		46.3%	Medium	
288	Unique Word Abbreviation 🔒		18.0%	Medium	
411	Minimum Unique Word Abbreviation 🔒		33.8%	Hard	

Question statement (3):

Write a function that takes a string as input and returns an abbreviation of the form: <first letter> <number of omitted letters> <last letter>.

If the candidate solves this quickly, add the following difficulty: Write another function that takes an array of strings as input and return true/false depending on whether this style of abbreviation is unique for this array words. E.g.: [“accessibility”, “automatically”, "airport"] should return false because the first two words both abbreviate to a11y.

Example

"internationalization" -> "i18n", "localization" -> "l10n"

```
boolean p3(String[] array) {
  HashSet<String> set = new HashSet<String>
  for (int i=0; i<array.length; i++) {
    String s = // ith entry in array
    if(set.contains(s))
      return false;
    Set.add(s);
  }
  return
```

Question (3)

I-dentify Problem

) Shorten a string by getting:

-) 1st letter

-) ~~#~~ of ~~duplicates of 1st~~ letters btwn 1st & last letters

-) last letter

Define Goal

The new string should ~~go from being a long string~~ name:

Explore/Anticipate/Set Based or Duke's Approach

1) workout examples by hand:

-) "internationalization"

-> i18n

-) "localization"

-> l10n

-) "sequestered"

-> s9d

-) "obtuse"

o 4 e

-) "red"

r 1 d

-) "accessibility"

→

a 11 y

-) "if" ~~what to~~

→

i 0 f

-) "I"

→

I 0 I ?

Q what happens if 2 letters?

Q what happen if 1 letter?

2) write down what I did

•) look/"store" first letter

•) count up to 2nd to last letter. & write-down/"store" the ~~##~~

•) look/"store" at the last letter

3) Any patterns?

.) First & last letters always looked at

4) tracing by hand,

.) done in step 1

5) translate to code

6) Run test cases

7) Debug failed Test cases