527. Word Abbreviation (/problems/word-abbreviation/)

Dec. 16, 2017 | 8.5K views

Average Rating: 4.56 (9 votes)

Given an array of n distinct non-empty strings, you need to generate **minimal** possible abbreviations for every word following rules below.

1. Begin with the first character and then the number of characters abbreviated, which followed by the last character.
2. If there are any conflict, that is more than one words share the same abbreviation, a longer prefix is used instead of only the first character until making the map from word to abbreviation become unique. In other words, a final abbreviation cannot map to more than one original words.
3. If the abbreviation doesn't make the word shorter, then keep it as original.

Example:

```
Input: ["like", "god", "internal", "me", "internet", "interval", "intension", "face", "int
Output: ["l2e","god","internal","me","i6t","interval","inte4n","f2e","intr4n"]
```

Input: ["like", "god", "internal", "me", "internet", "interval", "intension", "face", "intrusion"]
Output: ["l2e","god","internal","me","i6t","interval","inte4n","f2e","intr4n"]

Note:

1. Both n and the length of each word will not exceed 400.
2. The length of each word is greater than 1.
3. The words consist of lowercase English letters only.
4. The return answers should be **in the same order** as the original array.

Approach #1: Greedy [Accepted]

Intuition

Let's choose the shortest abbreviation for each word. Then, while we have duplicates, we'll increase the length of all duplicates.

Algorithm

For example, let's say we have `"aabaaa", "aacaaa", "aacdaa"`, then we start with `"a4a", "a4a", "a4a"`. Since these are duplicated, we lengthen them to `"aa3a", "aa3a", "aa3a"`. They are still duplicated, so we lengthen them to `"aab2a", "aac2a", "aac2a"`. The last two are still duplicated, so we lengthen them to `"aacaaa", "aacdaa"`.

Throughout this process, we were tracking an index `prefix[i]` which told us up to what index to take the prefix to. For example, `prefix[i] = 2` means to take a prefix of `word[0], word[1], word[2]`.

Java Python Copy

```python
class Solution(object):
    def wordsAbbreviation(self, words):
        def abbrev(word, i = 0):
            if (len(word) - i <= 3): return word
            return word[:i+1] + str(len(word) - i - 2) + word[-1]

        N = len(words)
        ans = map(abbrev, words)
        prefix = [0] * N

        for i in xrange(N):
            while True:
                dupes = set()
                for j in xrange(i+1, N):
                    if ans[i] == ans[j]:
                        dupes.add(j)

                if not dupes: break
                dupes.add(i)
                for k in dupes:
                    prefix[k] += 1
                    ans[k] = abbrev(words[k], prefix[k])

        return ans
```

Complexity Analysis

- Time Complexity: $O(C^2)$ where C is the number of characters across all words in the given array.
- Space Complexity: $O(C)$.

Approach #2: Group + Least Common Prefix [Accepted]

Intuition and Algorithm

Two words are only eligible to have the same abbreviation if they have the same first letter, last letter, and length. Let's group each word based on these properties, and then sort out the conflicts.

In each group `G`, if a word `W` has a longest common prefix `P` with any other word `X` in `G`, then our abbreviation must contain a prefix of more than `|P|` characters. The longest common prefixes must occur with words adjacent to `W` (in lexicographical order), so we can just sort `G` and look at the adjacent words.

Java | Python

Copy

```
class Solution(object):
    def wordsAbbreviation(self, words):
        def longest_common_prefix(a, b):
            i = 0
            while i < len(a) and i < len(b) and a[i] == b[i]:
                i += 1
            return i

        ans = [None for _ in words]

        groups = collections.defaultdict(list)
        for index, word in enumerate(words):
            groups[len(word), word[0], word[-1]].append((word, index))

        for (size, first, last), enum_words in groups.iteritems():
            enum_words.sort()
            lcp = [0] * len(enum_words)
            for i, (word, _) in enumerate(enum_words):
                if i:
                    word2 = enum_words[i-1][0]
                    lcp[i] = longest_common_prefix(word, word2)
                    lcp[i-1] = max(lcp[i-1], lcp[i])

            for (word, index), p in zip(enum_words, lcp):
                delta = size - 2 - p
                if delta <= 1:
                    ans[index] = word
```

Complexity Analysis

- Time Complexity: $O(C \log C)$ where C is the number of characters across all words in the given array. The complexity is dominated by the sorting step.
- Space Complexity: $O(C)$.

Approach #3: Group + Trie [Accepted]

Intuition and Algorithm

As in *Approach #1*, let's group words based on length, first letter, and last letter, and discuss when words in a group do not share a longest common prefix.

Put the words of a group into a trie (prefix tree), and count at each node (representing some prefix `P`) the number of words with prefix `P`. If the count is 1, we know the prefix is unique.

Java | Python

Copy

```
class Solution(object):
    def wordsAbbreviation(self, words):
        groups = collections.defaultdict(list)
        for index, word in enumerate(words):
            groups[len(word), word[0], word[-1]].append((word, index))

        ans = [None] * len(words)
        Trie = lambda: collections.defaultdict(Trie)
        COUNT = False
        for group in groups.itervalues():
            trie = Trie()
            for word, _ in group:
                cur = trie
                for letter in word[1:]:
                    cur[COUNT] = cur.get(COUNT, 0) + 1
                    cur = cur[letter]

            for word, index in group:
                cur = trie
                for i, letter in enumerate(word[1:], 1):
                    if cur[COUNT] == 1: break
                    cur = cur[letter]
                if len(word) - i - 1 > 1:
                    ans[index] = word[:i] + str(len(word) - i - 1) + word[-1]
                else:
                    ans[index] = word
        return ans
```

Complexity Analysis

- Time Complexity: $O(C)$ where C is the number of characters across all words in the given array.
- Space Complexity: $O(C)$.

Analysis written by: @awice (https://leetcode.com/awice). Approach #1 inspired by @ckcz123 (https://discuss.leetcode.com/topic/82613/really-simple-and-straightforward-java-solution).

Rate this article:

Previous (/articles/split-array-largest-sum/) Articles > 527. Word Abbreviation Next (/articles/shortest-completing-word/)
Sort By

Comments: 9

Type comment here... (Markdown is supported)

Preview Post

(/iccs)

iccs (iccs) ★ 8 September 13, 2018 1:15 PM

`longestCommonPrefix` is not needed in approach 3.

6 Share Reply

SHOW 2 REPLIES

(/jh_x)

jh_x (jh_x) ★ 7 April 21, 2019 12:11 AM

In the 3rd solution, there is no usage of the function "longestCommonPrefix". .

3 Share Reply

(/gundabathula)

gundabathula (gundabathula) ★ 3 November 5, 2018 9:32 AM

How is the time complexity in first approach O(C*C)?

2 Share Reply

SHOW 1 REPLY

(/crayia)

crayia (crayia) ★ 1 July 29, 2019 11:16 AM

Why is the complexity O(ClogC) for approach 2? The sorting should be O(ClogN), where N is the number of strings, so shouldn't it be O(ClogN)?

1 Share Reply

(/galileo_galilei)

Galileo_Galilei (galileo_galilei) ★ 315 January 5, 2019 3:30 PM

I think the time complexity of 1st approach is O(m^2 * n^2), m is average length of word, n is size of dict. In first for loop we have n, in second while loop, we can have at most m times, and each time will cost us at most nm , so total is O(m^2*n^2).

0 Share Reply

(/zhaomai)

ZhaoMai (zhaomai) ★ 81 May 19, 2019 1:48 PM

many times Trie can be replaced by hash map (but might use more spaces). same as here, you can just a hash map of {'abbr': full word}, and expand the hash map gradually when adding new word (just like grow the trie when adding new word).

0 Share Reply