527. Word Abbreviation

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

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

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

```java
class Solution {
    public List<String> wordsAbbreviation(List<String> words) {
        Map<String, List<IndexedWord>> groups = new HashMap();
        String[] ans = new String[words.size()];

        int index = 0;
        for (String word: words) {
            String ab = abbrev(word, 0);
            if (!groups.containsKey(ab))
                groups.put(ab, new ArrayList());
            groups.get(ab).add(new IndexedWord(word, index));
            index++;
        }

        for (List<IndexedWord> group: groups.values()) {
            TrieNode trie = new TrieNode();
            for (IndexedWord iw: group) {
                TrieNode cur = trie;
                for (char letter: iw.word.substring(1).toCharArray()) {
                    if (cur.children[letter - 'a'] == null)
                        cur.children[letter - 'a'] = new TrieNode();
                    cur.count++;
                    cur = cur.children[letter - 'a'];
                }
            }

            for (IndexedWord iw: group) {
```

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).

Comments: 0

Sort By

Type comment here... (Markdown is supported)

Preview Post