Replace Words

In English, we have a concept called `root`, which can be followed by some other words to form another longer word - let's call this word `successor`. For example, the root `an`, followed by `other`, which can form another word `another`.

Now, given a dictionary consisting of many roots and a sentence. You need to replace all the `successor` in the sentence with the `root` forming it. If a `successor` has many `roots` can form it, replace it with the root with the shortest length.

You need to output the sentence after the replacement.

Example 1:

```
Input: dict = ["cat", "bat", "rat"]
sentence = "the cattle was rattled by the battery"
Output: "the cat was rat by the bat"
```

Note:

1. The input will only have lower-case letters.
2. 1
3. 1
4. 1
5. 1

Solution 1

Why it is a hard problem? Anyway...

```
public class Solution {
    public String replaceWords(List<String> dict, String sentence) {
        if (dict == null || dict.size() == 0) return sentence;

        Set<String> set = new HashSet<>();
        for (String s : dict) set.add(s);

        StringBuilder sb = new StringBuilder();
        String[] words = sentence.split("\\s+");

        for (String word : words) {
            String prefix = "";
            for (int i = 1; i <= word.length(); i++) {
                prefix = word.substring(0, i);
                if (set.contains(prefix)) break;
            }
            sb.append(" " + prefix);
        }

        return sb.deleteCharAt(0).toString();
    }
}
```

written by shawngao original link here

Solution 2

The code could be shorter but I wanted to add two performance optimizations.

1. You do not need to build the entire trie for roots in the dictionary. You can stop at the shortest root.
2. The optimized root function returns the root or the entire word (till the space). If it returns the root, the calling function need to skip to the next space. That way, we only go through the sentence once.

```
class trie {
    bool isRoot = false;
    trie* l[26] = {};
public:
    void insert(string& word, int ch, int sz) {
        isRoot |= ch == sz;
        if (!isRoot) { // stop at the shortest root.
            if (l[word[ch] - 'a'] == nullptr) l[word[ch] - 'a'] = new trie();
            l[word[ch] - 'a']->insert(word, ch + 1, sz);
        }
    }
    int root(string& word, int st, int ch, int sz) {
        if (st + ch == sz || word[st + ch] == ' ' || this->isRoot) return ch;
        if (l[word[st + ch] - 'a'] == nullptr) { // root was not found
            while (st + ch < sz && word[st + ch] != ' ') ++ch; // skipping the enti
re word
            return ch;
        }
        return l[word[st + ch] - 'a']->root(word, st, ch + 1, sz);
    }
};
string replaceWords(vector<string>& dict, string snt) {
    trie t;
    string res;
    for (auto s : dict) t.insert(s, 0, s.size());
    for (int i = 0; i < snt.size(); ) {
        if (snt[i] == ' ') res += snt[i++];
        auto chars = t.root(snt, i, 0, snt.size());
        res += snt.substr(i, chars);
        for (i += chars; i < snt.size() && snt[i] != ' '; ++i);
    }
    return res;
}
```

written by votrubac original link here

Solution 3

The only modification to the standard Trie, is that we need a function that returns the shortest prefix of the given word in the trie, if the shortest prefix exists or return the original word. Once we have this, all we have to do is iterate through the sentence and replace each word with the shortest prefix in the trie.

```java
public String replaceWords(List<String> dict, String sentence) {
    Trie1 trie = new Trie1(256);
    dict.forEach(s -> trie.insert(s));
    List<String> res = new ArrayList<>();
    Arrays.stream(sentence.split(" ")).forEach(str -> res.add(trie.search(str)));
    return res.stream().collect(Collectors.joining(" "));
}

class Trie1 {
    private int R;
    private TrieNode root;

    private class TrieNode {
        private TrieNode[] next = new TrieNode[R];
        private boolean isWord;
    }

    public Trie1(int R) {
        this.R = R;
        root = new TrieNode();
    }

    // Inserts a word into the trie.
    public void insert(String word) {
        insert(root, word);
    }

    private void insert(TrieNode root, String word) {
        if(word.isEmpty()) {
            root.isWord = true;
            return;
        }
        if(root.next[word.charAt(0)] == null) {
            root.next[word.charAt(0)] = new TrieNode();
        }
        insert(root.next[word.charAt(0)], word.substring(1));
    }

    // Returns the shortest prefix of the word that is there in the trie
    public String search(String word) {
        int len =  search(root, word, -1);
        if(len < 1) return word;
        return word.substring(0, len);
    }

    private int search(TrieNode root, String word, int res) {
        if(root == null)
            return 0;
        if(word.isEmpty())
            return 0;
        if(root.isWord) {
            return res + 1;
        }
        return search(root.next[word.charAt(0)], word.substring(1), res+1);
    }
}
```

written by johnyrufus16 original link here

From Leetcoder.