Solution: Replace Words

Let's solve the Replace Words problem using the Trie pattern.

We'll cover the following

Statement

In this problem, we are considering the words that are composed of a prefix and a postfix. For example, if we append a postfix “happy” to a prefix “un”, it forms the word “unhappy”. Similarly, “disagree” is formed from a prefix, “dis” followed by a postfix, “agree”.

You’re given a dictionary, `dictionary`, consisting of prefixes, and a sentence, `sentence`, which has words separated by spaces only. Your task is to replace the words in `sentence` with their prefixes given in `dictionary` (if found) and return the modified sentence.

A couple of points to keep in mind:

- If a word in the sentence matches more than one prefix in the dictionary, replace it with the prefix that has the shortest length. For example, if we have the sentence “iphone is amazing”, and the dictionary {“i”, “ip”, “hone”}, then the word “iphone” has two prefixes in the dictionary “i” and “ip”, but we will replace it with the one that is shorter among the two, that is, “i”.
- If there is no root word against any word in the sentence, leave it unchanged.

Constraints:

- $1 \leq$ `dictionary.length` ≤ 1000
- $1 \leq$ `dictionary[i].length` ≤ 100
- `dictionary[i]` consists of only lowercase letters.
- $1 \leq$ `sentence.length` $\leq 10^3$
- The number of words in `sentence` is in the range $[1, 100]$.
- The length of each word in `sentence` is in the range $[1, 100]$.
- Two consecutive words in `sentence` should be separated by exactly one space.
- All words in `sentence` are lowercase.
- For a word in `sentence`, the length of a prefix can be $[1, 100]$, and the length of a postfix can be $[0, 100]$.

Solution

So far, you’ve probably brainstormed some approaches and have an idea of how to solve this problem. Let’s explore some of these approaches and figure out which one to follow based on considerations such as time

complexity and any implementation constraints.

Naive approach

This problem requires us to observe the leading characters (prefixes) of words in a sentence that can be replaced with the shortest matching prefix from a dictionary.

In the naive approach, we traverse the given sentence, and for each word, we compare it with each prefix in the given dictionary to find the shortest matching prefix against it, if any. To make sure that we get to the right prefix, we store the respective lengths of prefixes with them. Additionally, we keep track of the shortest prefix as well. If, at any point, a match is found, we compare its length with the length of the shortest match found up to that point. If it is shorter than the current shortest prefix, we update the value of the shortest prefix.

In this approach, we iterate through each word in the sentence, which takes $O(n)$ time, where n is the number of words in a sentence. For each word, we iterate over the prefixes in the dictionary, which takes $O(d)$ time, where d is the number of prefixes in the dictionary. Therefore, the overall time complexity for this approach is $O(n \times d)$.

Optimized approach using trie

Searching for the shortest prefixes for all the words in the given sentence requires repeated searches in the dictionary. While this would be an expensive proposition if we keep our dictionary words in data structures like arrays and linked lists, some more advanced data structures, such as tries, help reduce our search time complexity. In this approach, we first store all the dictionary prefixes in a trie and then find the shortest prefix against each word of the sentence so that we can perform the required replacement.

To store the dictionary prefixes in the trie, we start from its root node and perform the following steps repeatedly until we've stored all the words:

- We check if the current character of the dictionary prefix is listed among the immediate children of the current node.
    - If it doesn't exist, we create a new trie node for this character as a child of the current node and move to this new trie node.
    - If it exists among the children, we simply move to that node.
- If the current character is the last character of a given dictionary prefix, we use a flag to mark the trie node holding this last character. This flag represents the completion of a dictionary prefix on the trie. For example, if we have the prefix "dis" in the dictionary, then when we reach "s" and store it in the trie. We mark this node with the flag to represent that the word, "dis", has been completed.

After we are done creating the trie from the dictionary prefixes, we start to locate the matching prefix for each word of the sentence. Once found, we replace the word in the sentence with it. For example, if the input sentence is "fantastic" and the given dictionary is ["fanta", "tic", "fan"], then both "fanta" and "fan" are matching prefixes for "fantastic", but we replace it with "fan" since it is the shortest of all the found matches. Since we are using a trie, our search automatically picks up the shortest matching prefix, and we don't need to do it later.

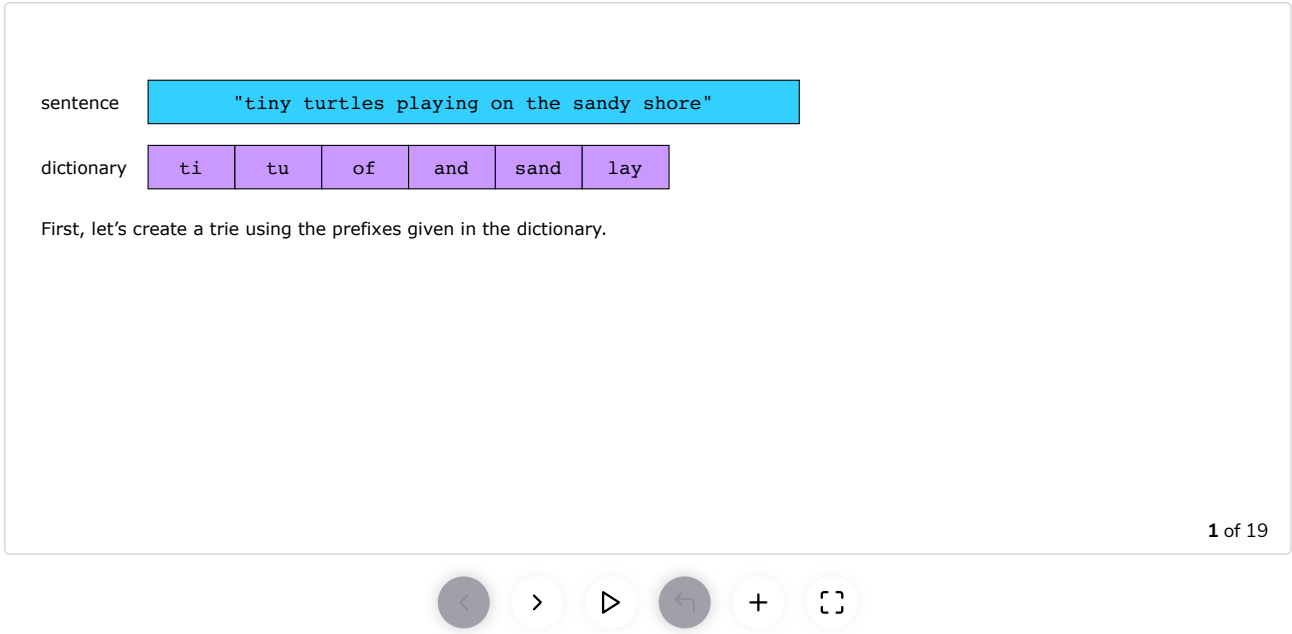
To traverse all the words in the sentence, we store them in a list. Then, for each word, w_i, perform the following:

- Start with the first character, $w_i[0]$, and look at whether the same character exists in any of the children of the root node of the trie.

- If it exists, it implies that the prefix for w_i exists, and we can continue as follows:
  - For remaining characters, keep looking for the matching character in the trie one by one.
  - If at any point while searching and matching, we reach a node that is flagged for being the last character, we stop and return this prefix as the shortest matching prefix for w_i.
- Otherwise, the prefix against w_i doesn't exist, so just move to the next word of the sentence.

Once found, replace the original word w_i in the list with its matched prefix. Keep doing this until we process the last word in the sentence. In the end, change the list of words back to the string representing the modified sentence and return it as the output.

The slides below illustrate the steps of the solution in detail:



Let's look at the code for this solution below:

Java

main.java

TrieNode.java

Trie.java

```java
class ReplaceWords {

    public String replaceWords(String sentence, List<String> dictionary) {
        Trie trie = new Trie();
        trie.root = new TrieNode();

        // iterate over the prefixes in the dictionary, and
        // insert them in the trie

        // NOTE: insertInTrie function is implemented in the file, Trie.java
        for (String prefix: dictionary) {
            trie.insertInTrie(prefix);
        }

        // store each word of the sentence in a new list
        List<String> sentanceList = Arrays.asList(sentence.split(" "));

        // this newList is intended to return the final sentence
        // after all possible replacements have been made
```

```
20          List<String> newList = new ArrayList< >();
21
22          // iterate over all the words in the sentence
23          for (String str: sentanceList) {
24
25              // replace each word in new_list with the
26              // shortest prefix found from the trie
27
28              // NOTE: replace function is implemented in the file  Trie.java
```

Replace Words

Let's review the optimized approach we have discussed above.

- Create a trie and store all the dictionary prefixes in it.
- Traverse the sentence and use the trie to find the shortest matching prefix for each word of the sentence.
- Once found, replace the original word of the sentence with the matched prefix. We do this for all the words in the sentence.
- If no matching prefix is found, leave the word as it is and move to the next word in the sentence.
- Return the modified sentence as the output.

Time complexity

It takes $O(m)$ time to create a trie from the given dictionary prefixes, where m is the sum of the number of characters of all the prefixes in the dictionary.

Then, it takes $O(n)$ time to iterate over each word in the sentence and find its matching prefix in the trie, where n is the number of words in the sentence.

Therefore, the overall time complexity for this approach is $O(n + m)$.

Space complexity

The space complexity is $O(m)$, where m is the sum of the number of characters of all the prefixes in the dictionary.

Back
Replace Words

Next
Design Add and Searc...

Mark as Completed