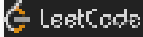


(1) Implement Trie (Prefix Tree) - LeetCode Articles

笔记本: <Inbox>
创建时间: 2020/2/26 14:04
标签: Prefix Tree, 代码收集, 前缀树
URL: <https://leetcode.com/articles/implement-trie-prefix-tree/>

 Explore Problems new Mock Contest Articles Discuss Store Premium

[Articles](#) [208. Implement Trie \(Prefix Tree\)](#)

[Previous](#) [Next](#)

208. Implement Trie (Prefix Tree)

April 11, 2016 | 152.1K views

Average Rating: 4.91 (359 votes)

Implement a trie with `insert`, `search`, and `startsWith` methods.

Example:

```
Trie trie = new Trie();

trie.insert("apple");
trie.search("apple"); // returns true
trie.search("app");   // returns false
trie.startsWith("app"); // returns true
trie.insert("app");
trie.search("app");    // returns true
```

Note:

- You may assume that all inputs are consist of lowercase letters `a-z`.
- All inputs are guaranteed to be non-empty strings.

Summary

This article is for intermediate level users. It introduces the following ideas: The data structure Trie (Prefix tree) and most common operations with it.

Solution

Applications

Trie (we pronounce "try") or prefix tree is a tree data structure, which is used for retrieval of a key in a dataset of strings. There are various applications of this very efficient data structure such as :

1. Autocomplete

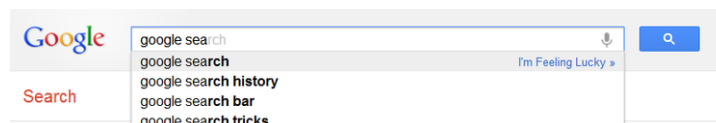


Figure 1. Google Suggest in action.

2. Spell checker

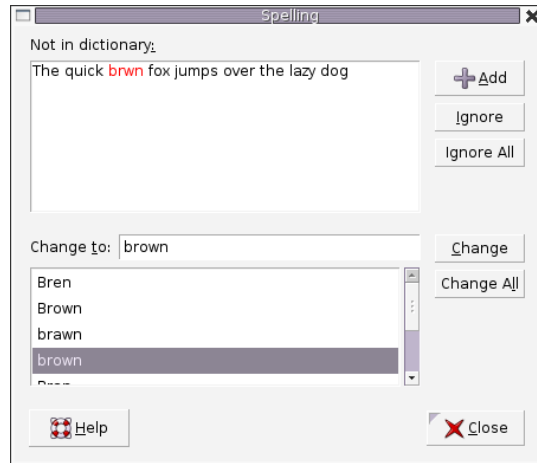


Figure 2. A spell checker used in word processor.

3. IP routing (Longest prefix matching)

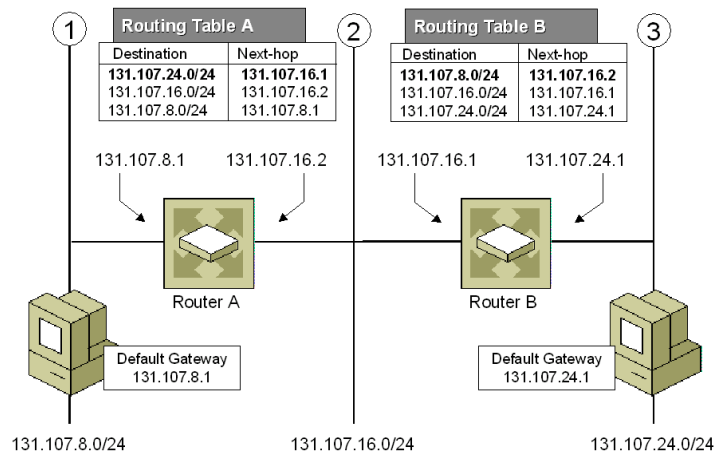


Figure 3. Longest prefix matching algorithm uses Tries in Internet Protocol (IP) routing to select an entry from a forwarding table.

4. T9 predictive text

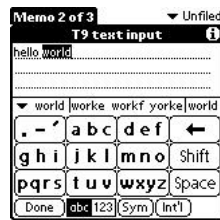


Figure 4. T9 which stands for Text on 9 keys, was used on phones to input texts during the late 1990s.

5. Solving word games

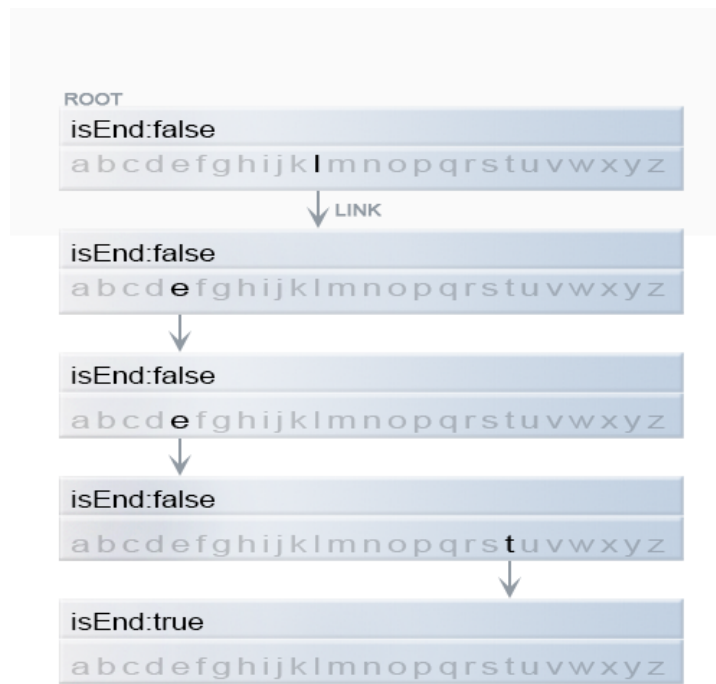


Figure 6. Representation of a key "leet" in trie.

Java

```

class TrieNode {

    // R links to node children
    private TrieNode[] links;

    private final int R = 26;

    private boolean isEnd;

    public TrieNode() {
        links = new TrieNode[R];
    }

    public boolean containsKey(char ch) {
        return links[ch - 'a'] != null;
    }

    public TrieNode get(char ch) {
        return links[ch - 'a'];
    }

    public void put(char ch, TrieNode node) {
        links[ch - 'a'] = node;
    }

    public void setEnd() {
        isEnd = true;
    }

    public boolean isEnd() {
        return isEnd;
    }

}

```

Two of the most common operations in a trie are insertion of a key and search for a key.

Insertion of a key to a trie

We insert a key by searching into the trie. We start from the root and search a link, which corresponds to the first key character. There are two cases :

- A link exists. Then we move down the tree following the link to the next child level. The algorithm continues with searching for the next key character.
- A link does not exist. Then we create a new node and link it with the parent's link matching the current key character. We repeat this step until we encounter the last character of the key, then we mark the

current node as an end node and the algorithm finishes.

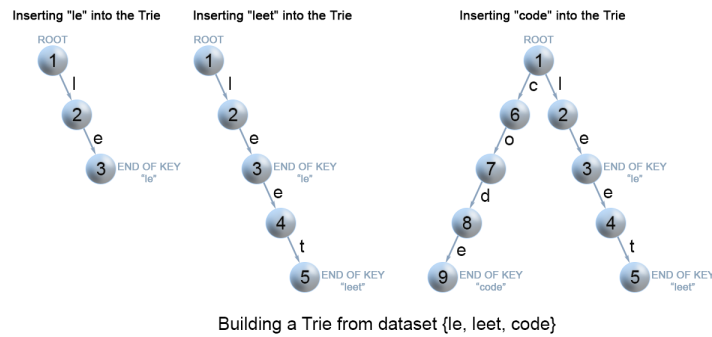


Figure 7. Insertion of keys into a trie.

Java

```
class Trie {
    private TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    // Inserts a word into the trie.
    public void insert(String word) {
        TrieNode node = root;
        for (int i = 0; i < word.length(); i++) {
            char currentChar = word.charAt(i);
            if (!node.containsKey(currentChar)) {
                node.put(currentChar, new TrieNode());
            }
            node = node.get(currentChar);
        }
        node.setEnd();
    }
}
```

Complexity Analysis

- Time complexity : $O(m)$, where m is the key length.

In each iteration of the algorithm, we either examine or create a node in the trie till we reach the end of the key. This takes only m operations.

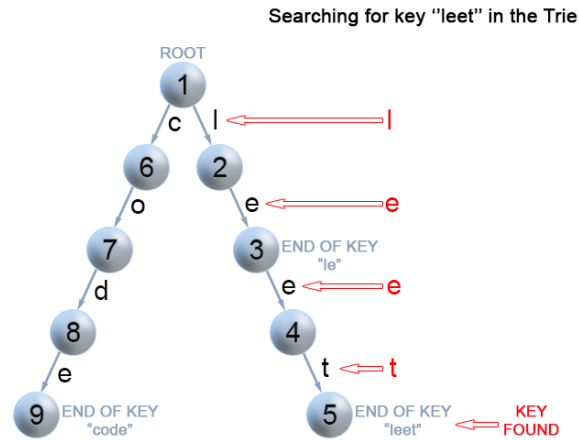
- Space complexity : $O(m)$.

In the worst case newly inserted key doesn't share a prefix with the the keys already inserted in the trie. We have to add m new nodes, which takes us $O(m)$ space.

Search for a key in a trie

Each key is represented in the trie as a path from the root to the internal node or leaf. We start from the root with the first key character. We examine the current node for a link corresponding to the key character. There are two cases :

- A link exist. We move to the next node in the path following this link, and proceed searching for the next key character.
- A link does not exist. If there are no available key characters and current node is marked as `isEnd` we return true. Otherwise there are possible two cases in each of them we return false :
 - There are key characters left, but it is impossible to follow the key path in the trie, and the key is missing.
 - No key characters left, but current node is not marked as `isEnd` . Therefore the search key is only a prefix of another key in the trie.



Searching for a key in a Trie from dataset {le, leet, code}

Figure 8. Search for a key in a trie.

Java

```
class Trie {
    ...

    // search a prefix or whole key in trie and
    // returns the node where search ends
    private TrieNode searchPrefix(String word) {
        TrieNode node = root;
        for (int i = 0; i < word.length(); i++) {
            char curLetter = word.charAt(i);
            if (node.containsKey(curLetter)) {
                node = node.get(curLetter);
            } else {
                return null;
            }
        }
        return node;
    }

    // Returns if the word is in the trie.
    public boolean search(String word) {
        TrieNode node = searchPrefix(word);
        return node != null && node.isEnd();
    }
}
```

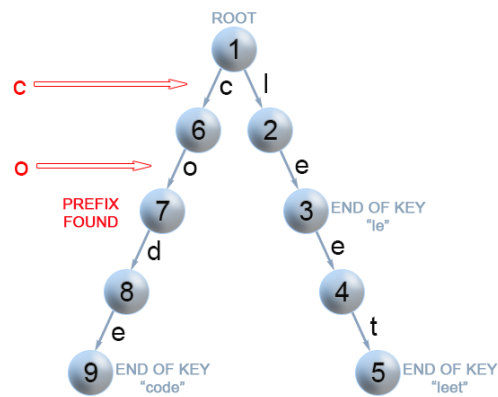
Complexity Analysis

- Time complexity : $O(m)$ In each step of the algorithm we search for the next key character. In the worst case the algorithm performs m operations.
- Space complexity : $O(1)$

Search for a key prefix in a trie

The approach is very similar to the one we used for searching a key in a trie. We traverse the trie from the root, till there are no characters left in key prefix or it is impossible to continue the path in the trie with the current key character. The only difference with the mentioned above `search for a key` algorithm is that when we come to an end of the key prefix, we always return true. We don't need to consider the `isEnd` mark of the current trie node, because we are searching for a prefix of a key, not for a whole key.

Searching for "co" in the Trie



Searching for a prefix in a Trie from dataset {le, leet, code}

Figure 9. Search for a key prefix in a trie.

Java

```
class Trie {
    ...

    // Returns if there is any word in the trie
    // that starts with the given prefix.
    public boolean startsWith(String prefix) {
        TrieNode node = searchPrefix(prefix);
        return node != null;
    }
}
```

Complexity Analysis

- Time complexity : $O(m)$
- Space complexity : $O(1)$

Practice Problems

Here are some wonderful problems for you to practice which uses the Trie data structure.

1. [Add and Search Word - Data structure design](#) - Pretty much a direct application of Trie.
2. [Word Search II](#) - Similar to Boggle.

Analysis written by: @elmirap.

Rate this article:

[Previous](#)

[Next](#)

Comments: 50





HaochenPan 100 April 13, 2018 11:49 AM

Thank you so much, I learned a new data structure!

100



vishwakarma.iita 36 September 30, 2017 9:49 PM

Very well written.

35



abduere11 17 August 15, 2018 1:37 PM

amazing explanation

15



yaosongding 12 June 30, 2018 5:29 PM

man I love u, nice guide for me

10



821705834 2 July 27, 2019 2:15 AM

Really need codes for python and C++.

2

[SHOW 4 REPLIES](#)



wintop6211 309 June 26, 2019 2:07 AM

The application part is amazing, we need more solutions that come with it.

1



hhtagels 65 July 20, 2019 2:30 AM

Thanks for the detailed explanation! Amazing stuff!

0



danielle44 0 July 13, 2019 5:34 PM

Great! Thanks!

0



yunmeng 0 July 1, 2019 9:24 AM

Thank you so much for the clear explanation!

0



davidelena 7 June 23, 2019 1:11 AM

Very clear and easy to understand, Thank your for your answer~

0

1