

## 208. Implement Trie (Prefix Tree)

### (/problems/implement-trie-prefix-tree/)

April 11, 2016 | 66.7K views

Average Rating: 4.87 (126 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

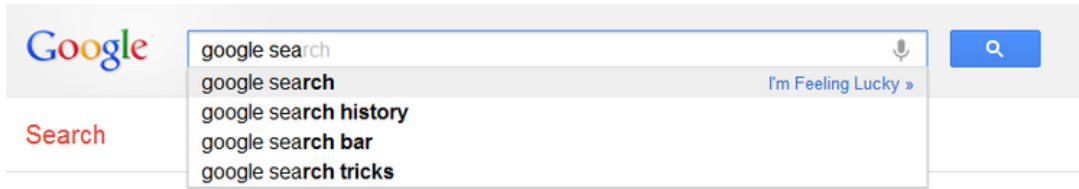
This article is for intermediate level users. It introduces the following ideas: The data structure Trie (Prefix tree) and most common operations with it.

# Solution

## Applications

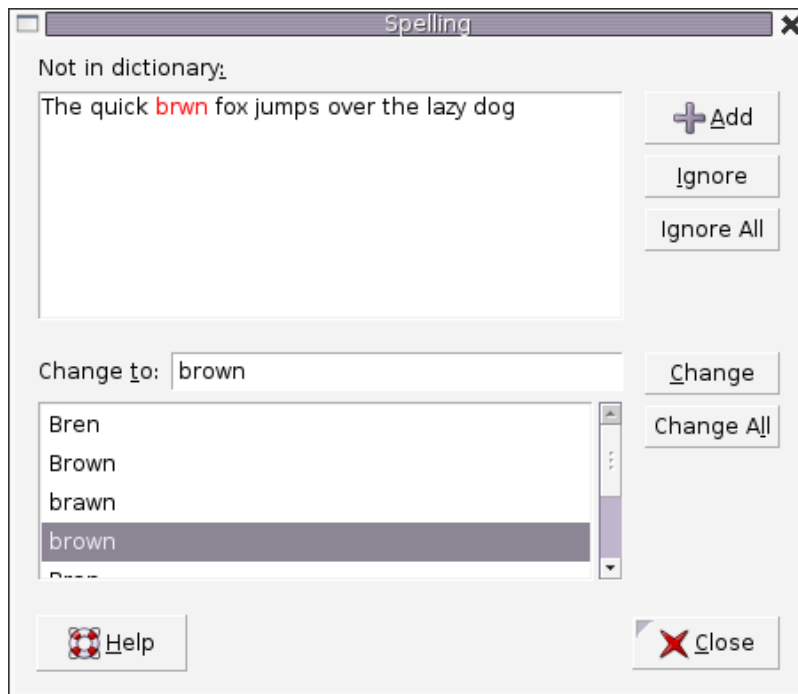
Trie (we pronounce "try") or prefix tree is a tree data structure, which is used for retrieval of a key in a dataset of strings. There are various applications of this very efficient data structure such as :

1. Autocomplete (<https://en.wikipedia.org/wiki/Autocomplete>)



*Figure 1. Google Suggest in action.*

2. Spell checker ([https://en.wikipedia.org/wiki/Spell\\_checker](https://en.wikipedia.org/wiki/Spell_checker))



*Figure 2. A spell checker used in word processor.*

3. IP routing (Longest prefix matching) ([https://en.wikipedia.org/wiki/Longest\\_prefix\\_match](https://en.wikipedia.org/wiki/Longest_prefix_match))

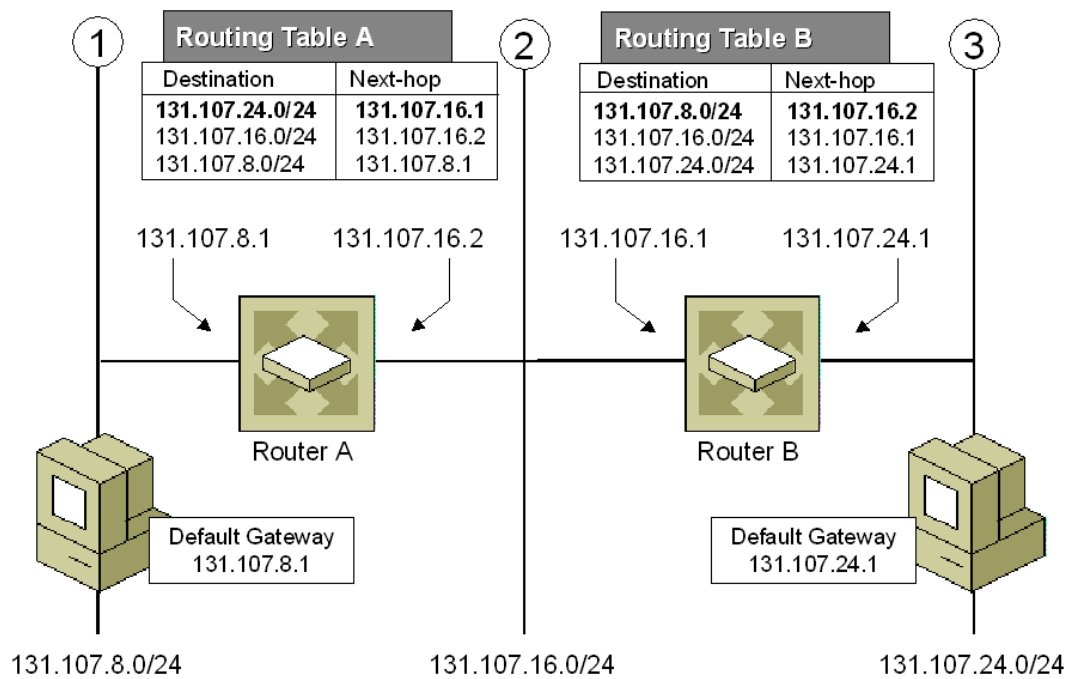


Figure 3. Longest prefix matching algorithm uses Tries in Internet Protocol (IP) routing to select an entry from a forwarding table.

4. T9 predictive text ([https://en.wikipedia.org/wiki/T9\\_\(predictive\\_text\)](https://en.wikipedia.org/wiki/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 (<https://en.wikipedia.org/wiki/Boggle>)



letters.

- Boolean field which specifies whether the node corresponds to the end of the key, or is just a key prefix.

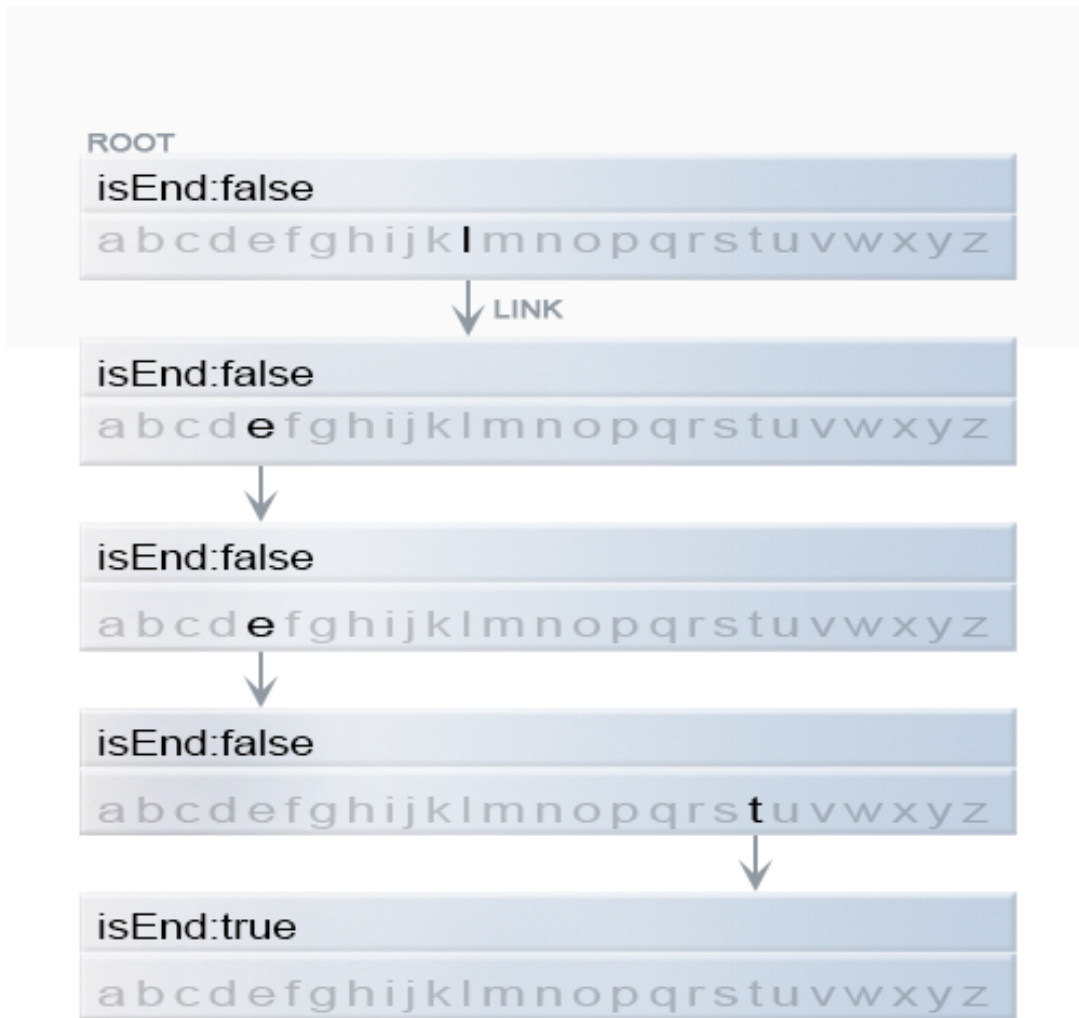


Figure 6. Representation of a key "leet" in trie.

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.

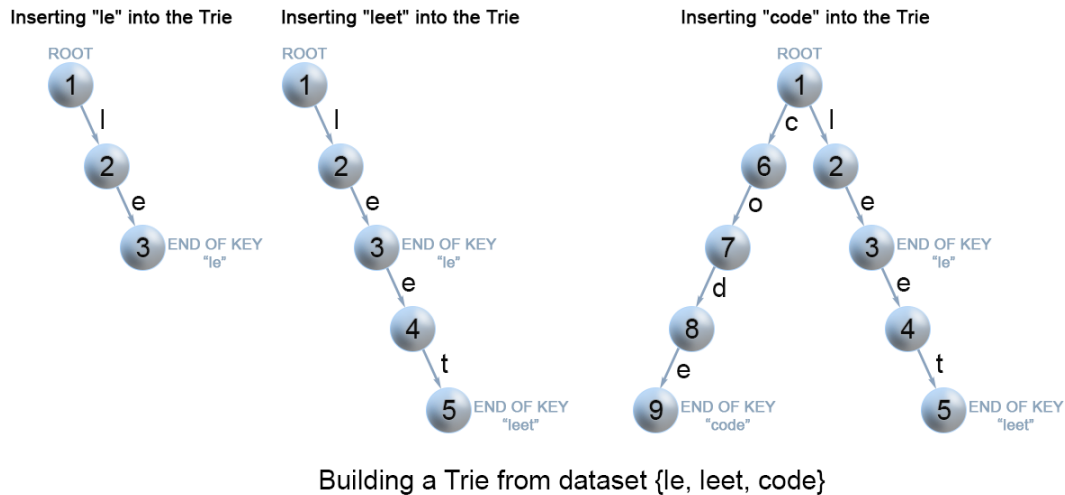


Figure 7. Insertion of keys into a trie.

## 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.

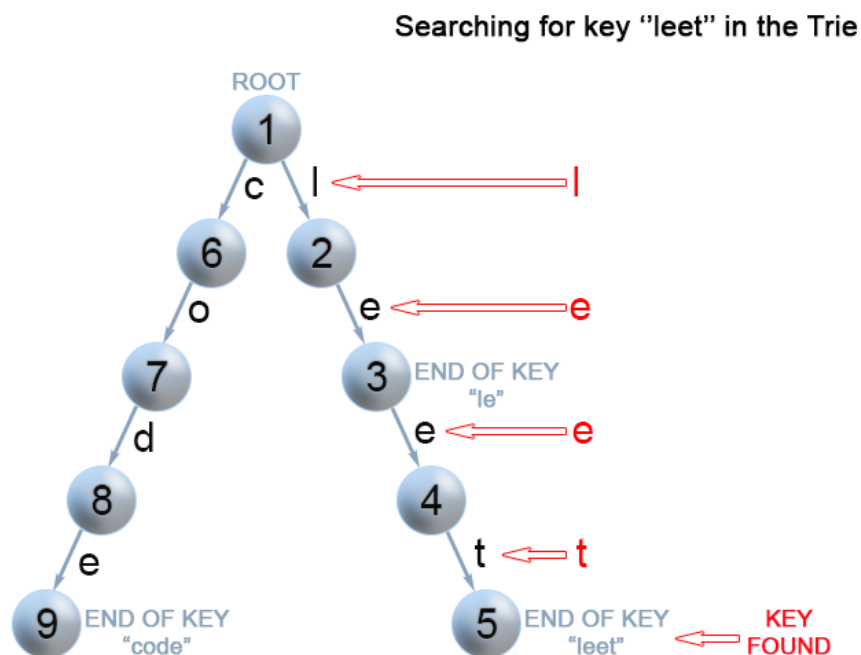
- Space complexity :  $O(m)$ .

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.



```

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();
    }
}

```

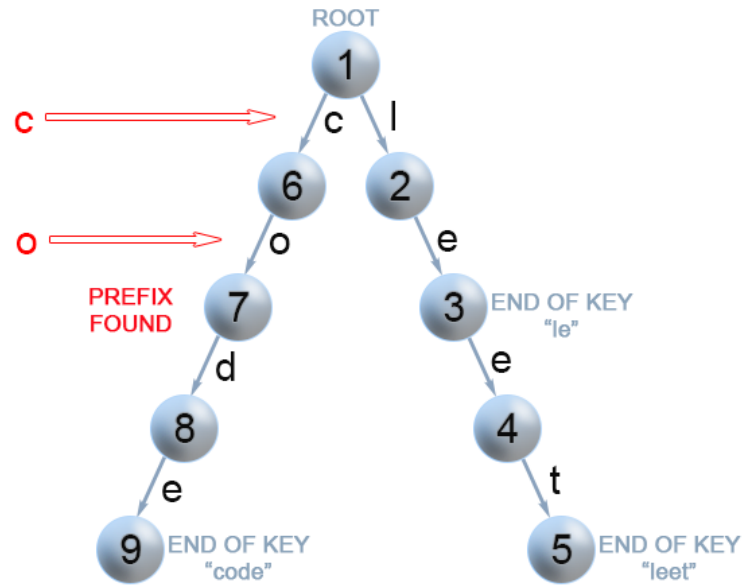
## Complexity Analysis

- 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 (<https://leetcode.com/problems/add-and-search-word-data-structure-design/>) - Pretty much a direct application of Trie.
2. Word Search II (<https://leetcode.com/problems/word-search-ii/>) - Similar to Boggle.

Analysis written by: @elmirap.

## Rate this article:

◀ Previous (</articles/meeting-rooms/>)

Next ▶ (</articles/implement-stack-using-queues/>)

## Comments: 12

Sort By ▼

Type comment here... (Markdown is supported)

👁 Preview

Post

sgsfak (/sgsfak) ★ 1 ⌚ September 14, 2018 3:48 AM



My implementation in C uses a Left-child right-sibling ([https://en.wikipedia.org/wiki/Left-child\\_right-sibling\\_binary\\_tree](https://en.wikipedia.org/wiki/Left-child_right-sibling_binary_tree)) structure instead of a hash table and with a runtime of 28ms appears to beat all other submissions. The submission is here (<https://leetcode.com/submissions/detail/175957876/>) -- I am not sure whether you can see it though.

Read More

0 ^ v | 📄 Share | ↩ Reply

abduere11 (/abduere11) ★ 3 ⌚ August 14, 2018 10:37 PM



amazing explanation

1 ^ v | 📄 Share | ↩ Reply

st4rgut (/st4rgut) ★ 3 ⌚ August 8, 2018 3:15 PM



doesn't space complexity depend on the number of strings in the trie, worst case potentially being n strings hence  $O(n)$ ?





0 ^ v | 📄 Share | ↩ Reply



SHOW 1 REPLY

yaosongding (/yaosongding) ★ 1 ⌚ June 30, 2018 2:29 AM









man I love u, nice guide for me

1    Share  Reply





Subhadeep2704 (/subhadeep2704) ★ 61  June 20, 2018 12:29 PM 



Good Stuff.

0    Share  Reply



HaochenPan (/haochenpan) ★ 9  April 12, 2018 8:49 PM 



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

9    Share  Reply

frostzyh (/frostzyh) ★ 17  April 5, 2018 12:55 PM 



It seems very similar to B/B+ tree. What are the difference between them?

0    Share  Reply

Skyfacon (/skyfacon) ★ 4  October 12, 2017 2:25 AM 

Really amazing

0    Share  Reply

vishwakarma.iiita (/vishwakarmaiiita) ★ 2  September 30, 2017 6:49 AM 

Very well written.

2    Share  Reply

bharath8 (/bharath8) ★ 0  July 22, 2017 1:55 PM 

That is what forums are for right. share your ideas, opinions even with people whom you dont know? :).  
anyways, what about the worst case?

0    Share  Reply

SHOW 2 REPLIES

[Contact Us \(/support/\)](/support/) | [Students \(/students/\)](/students/) | [Frequently Asked Questions \(/faq/\)](/faq/) | [Terms of Service \(/terms/\)](/terms/) |

[Privacy Policy \(/privacy/\)](/privacy/)

 [United States \(/region/\)](/region/)