Insertion in a Trie

This lesson defines all the cases needed for inserting a word into a trie, along with the Pythonic implementation.

We'll cover the following

Word Insertion

The insertion process is fairly simple. For each character in the key, we check if it exists at the position we desire. If the character is not present, then we insert the corresponding trie node at the correct index in `children`. While inserting the last node, we also set the value of `isEndWord` to `True`.

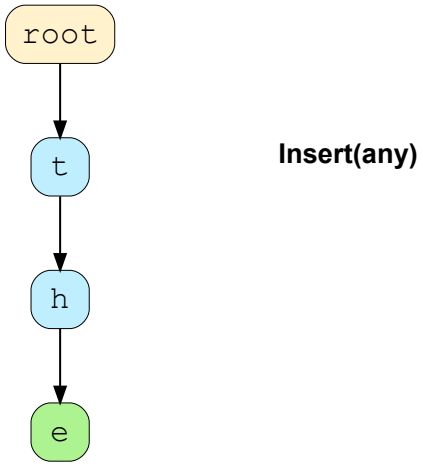
There are three primary cases we need to consider during insertion. Let's discuss them now.

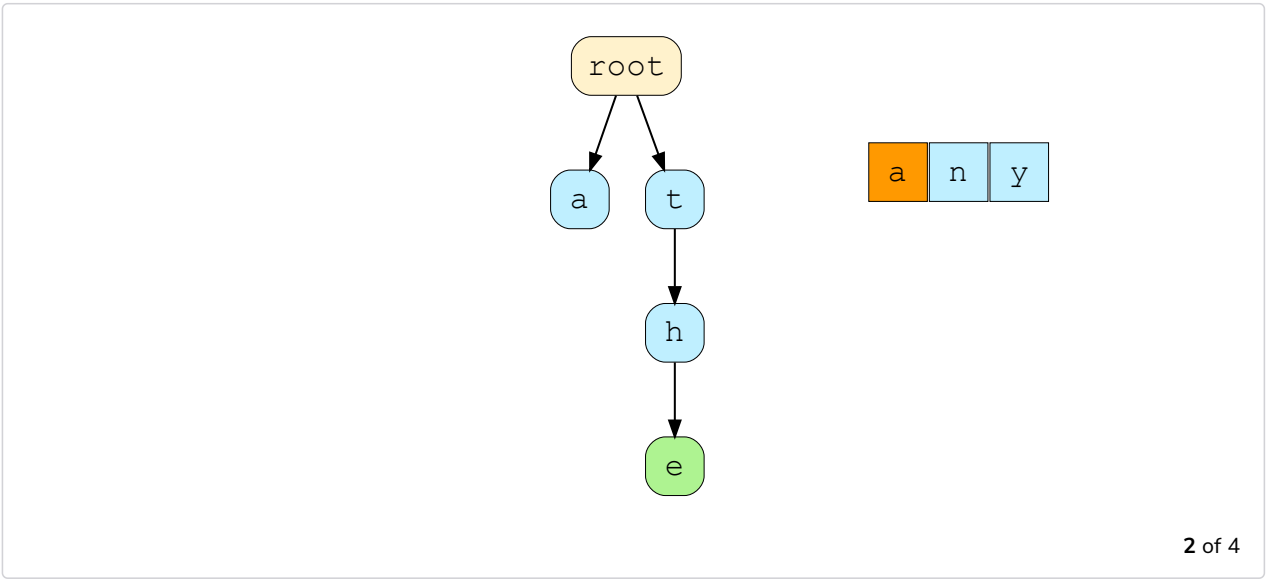
Case 1: No Common Prefix

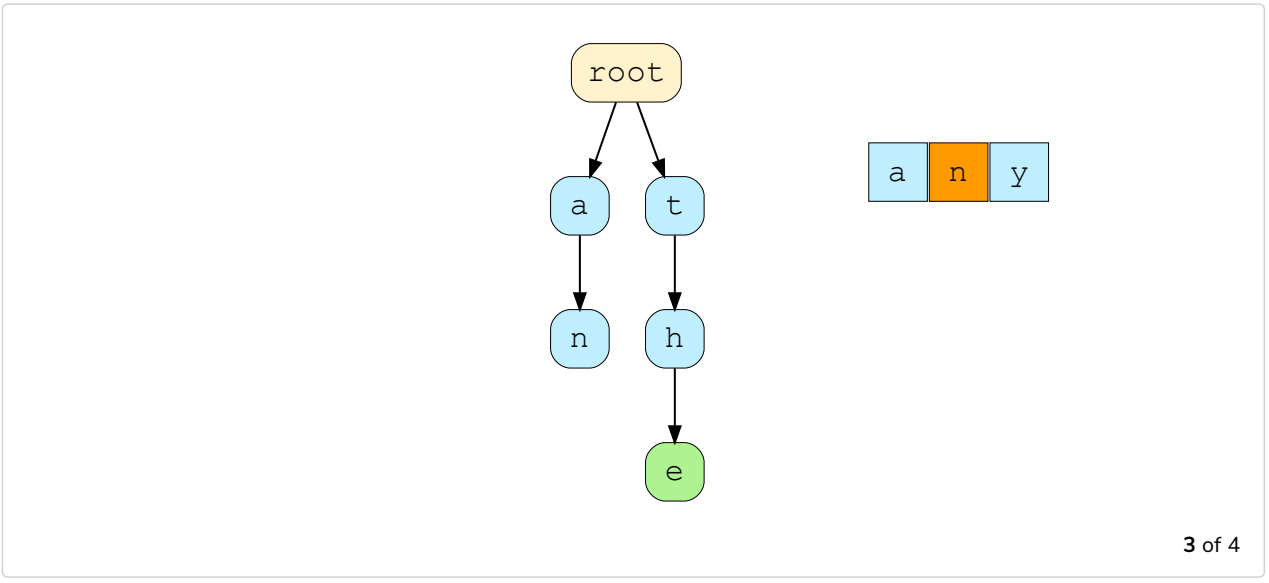
In this situation, we want to insert a word whose characters are not common with any other node path.

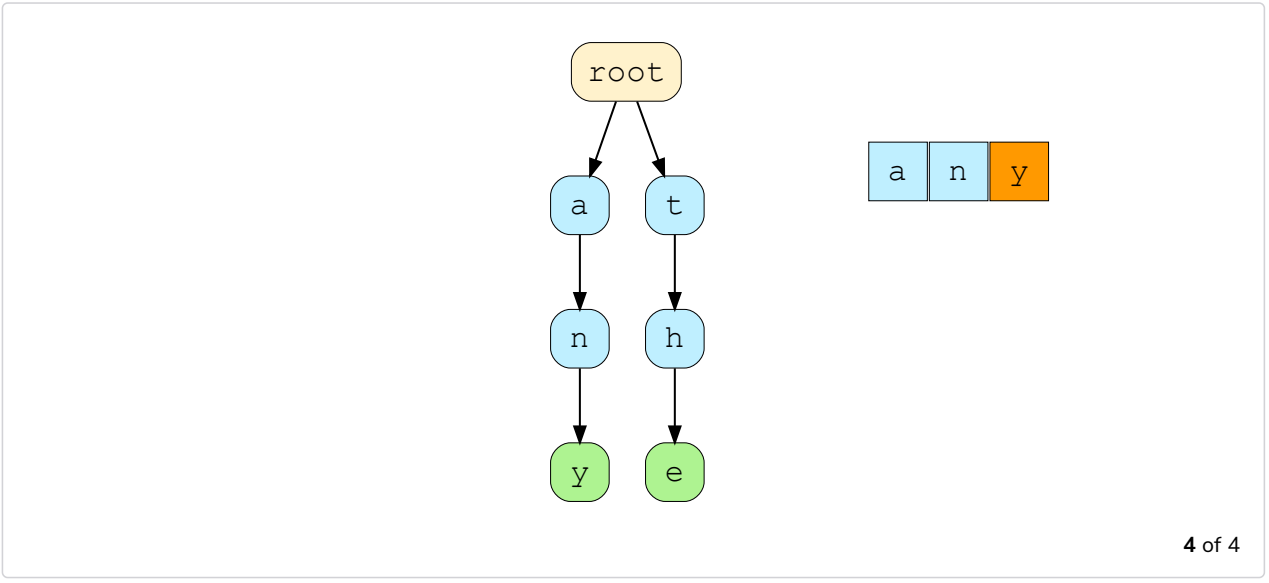
The illustration below shows the insertion of `any` in a trie which consists of only `the`.

We need to create nodes for all the characters of the word `any` as there is no common subsequence between `any` and `the`.







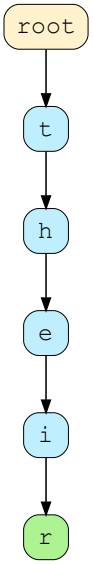


Case 2: Common Prefix

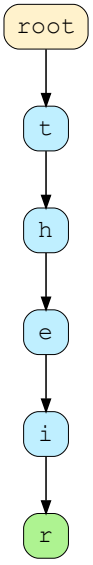
This occurs when a portion of the starting characters of your word already in the trie starting

from the `root` node.

For example, if we want to insert a new word `there` in the trie which consists of a word `their`, the path till `the` already exists. After that, we need to insert two nodes for `r` and `e` as shown below.

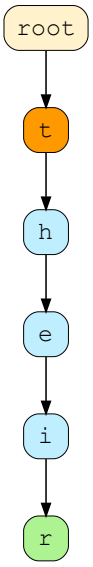


1 of 8



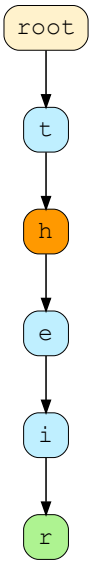
Insert 'there'

2 of 8



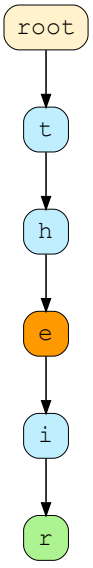
Path till 'the' is same

3 of 8

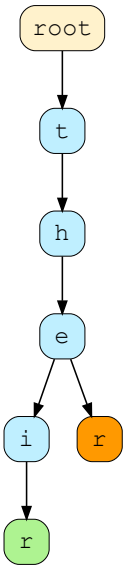


Path till 'the' is same

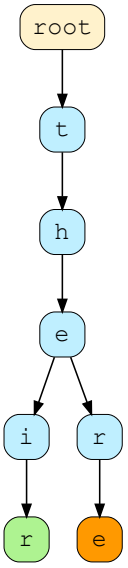
4 of 8

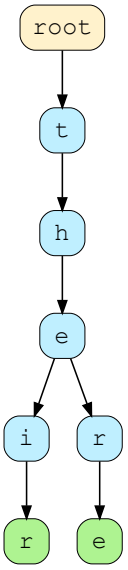


Path till 'the' is same



Now, creating new nodes for 'r' and 'e'



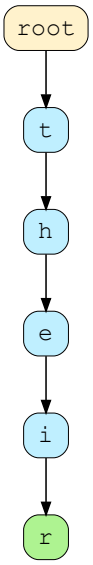


Marking 'e' as the end of the word

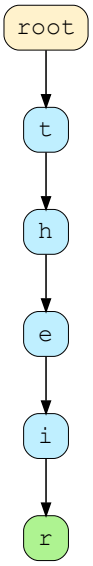
Case 3: Word Exists

This occurs if your word is a substring of another word that already exists in the trie.

For example, if we want to insert a word `the` in the trie which already contains `their`, the path for `the` already exists. Therefore, we simply need to set the value of `isEndWord` to true at `e` in order to represent the end of the word for `the` as shown below.

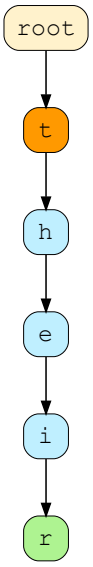


1 of 6



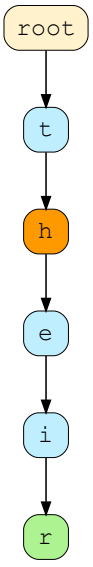
Insert 'the'

2 of 6



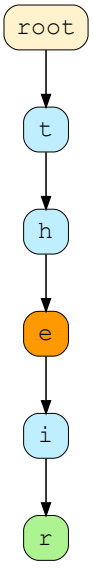
Path for 'the' is same

3 of 6

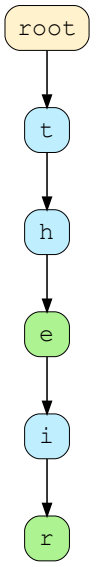


Path for 'the' is same

4 of 6



Path for 'the' is same



Mark 'e' as the end of the node

6 of 6

Implementation

Here is the implementation of the `insert` function based on the three cases we've seen.

Trie.py

TrieNode.py

```
from TrieNode import TrieNode


class Trie:
    def __init__(self):
        self.root = TrieNode()  # Root node

    # Function to get the index of character 't'
    def get_index(self, t):
        return ord(t) - ord('a')

    # Function to insert a key into the trie
    def insert(self, key):
        # None keys are not allowed
        if key is None:
            return

        key = key.lower()  # Keys are stored in lowercase
        current_node = self.root
        index = 0  # To store the character index

        # Iterate the trie with the given character index,
        # If the index points to None
        # simply create a TrieNode and go down a level
        for level in range(len(key)):
            index = self.get_index(key[level])

            if current_node.children[index] is None:
                current_node.children[index] = TrieNode(key[level])
                print(key[level] + " inserted")

            current_node = current_node.children[index]

        # Mark the end character as leaf node
        current_node.mark_as_leaf()
```

```
36          print("'" + key + "' inserted")
37
38      # Function to search a given key in Trie
39      def search(self, key):
40          return False
41
42      # Function to delete given key from Trie
43      def delete(self, key):
44          return
45
46
47  # Input keys (use only 'a' through 'z')
48  keys = ["the", "a", "there", "answer", "any",
49          "by", "bye", "their", "abc"]
50  output = ["Not present in trie", "Present in trie"]
51
52  t = Trie()
53  print("Keys to insert: ")
54  print(keys)
55
56  # Construct Trie
57  for i in range(len(keys)):
58      t.insert(keys[i])
59
```

The function takes in a string `key` indicating a word. `None` keys are not allowed, and all keys are stored in *lowercase*.

We simply iterate over the characters in `key` and for each character we generate an index using `get_index()`.

The next step is to check the child of `current_node` at that particular index and if it is `None`, we simply create a new `TrieNode` at that index.

We mark the last node as leaf since the word has ended.

Time Complexity

For a key with **n** characters, the worst case time complexity turns out to be *O(n)* since we need to make **n** iterations.

We have learned how to insert a word in a trie. Now, let's see how we can search for a particular word in a trie.

Back

Structure of a Trie

Next

Search in a Trie

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/insertion-in-a-trie__trie__data-structures-for-coding-interviews-in-python)