Structure of a Trie

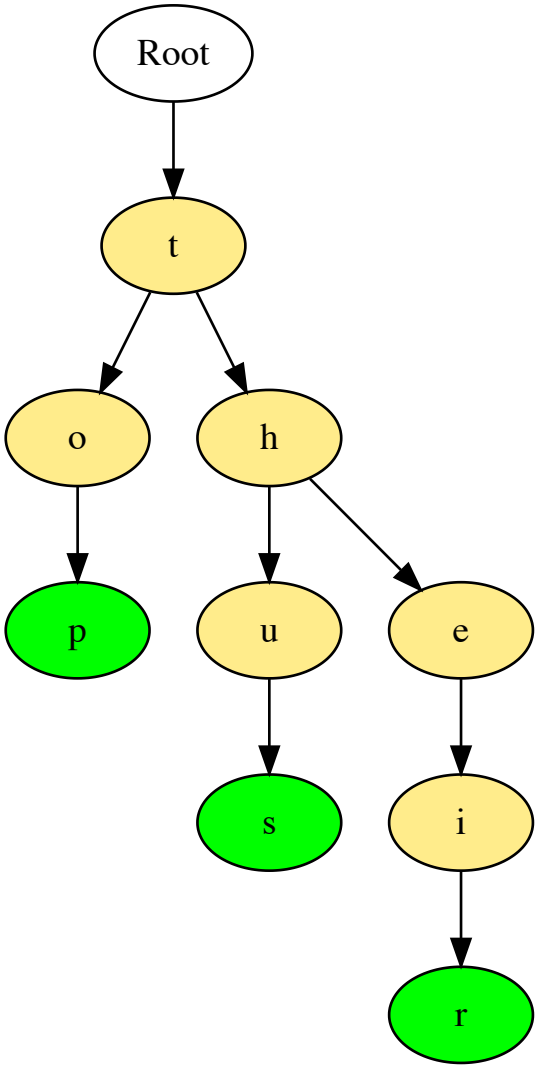
This lesson covers the structure of the Trie class in Python.

We'll cover the following

- Introduction
- The Trie Node Class
- The Trie Class

Introduction

In this lesson, we will take a look at the basic structure of a trie and then build a class in Python based on what we've studied.



Trie containing "top", "thus" and "their".

The Trie Node Class#

The node of a trie represents a letter. For example, if you want to insert *“hello”* in the trie, we will need to add 5 nodes, one for each letter. A typical node in a trie consists of three data members:

- `char` : This stores the character in the node.
- `children` : An array which consists of pointers to children nodes. The size of this array depends on the size of the alphabet, which is 26 for English.
- `is_end_word` : A flag to indicate the end of a word. It is set to `False` by default and is only updated when a word ends during insertion. When

this flag is `True`, the node is treated as a **leaf**.

Here is the implementation in Python:

TrieNode.py

```
class TrieNode:
    def __init__(self, char=''):
        self.children = [None] * 26  # This will store pointers to the children
        self.is_end_word = False  # true if the node represents the end of word
        self.char = char  # To store the value of a particular key

trie_node = TrieNode('a')
print(trie_node.char)
```

The Trie Class#

The `Trie` will be implemented using the `TrieNode` class. As discussed above, a trie node represents one letter which keeps pointers to its children nodes. Each node can have at max 26 children if we are storing English words.

A `root` node is placed at the top and contains 26 pointers (one per letter). These pointers hold either `None` or another `trie_node`. The `root` is similar to the `head_node` from linked lists.

All the words are stored in a top-bottom manner. While storing the last character, we should always set the `is_end_word` flag as `True` to indicate the end of a word. This technique helps us in searching for a word to see if it even exists.

A typical trie class looks like this in Python:

Trie.py

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

    # Function to insert a key in the Trie
    def insert(self, key):
        pass

    # Function to search a given key in Trie
    def search(self, key):
        return False

    # Function to delete given key from Trie
    def delete(self, key):
        pass
```

Here is the complete implementation in Python:

Trie.py

TrieNode.py

```
class TrieNode:
    def __init__(self, char=''):
        self.children = [None] * 26  # This will store pointers to the children
        self.is_end_word = False  # true if the node represents the end of word
        self.char = char  # To store the value of a particular key
```

Back

What is a Trie?

Next

Insertion in a Trie

Completed

Report an Issue