Trie

March 23, 2024

1 Building a Trie in Python

Before we start let us reiterate the key components of a Trie or Prefix Tree. A trie is a tree-like data structure that stores a dynamic set of strings. Tries are commonly used to facilitate operations like predictive text or autocomplete features on mobile phones or web search.

Before we move into the autocomplete function we need to create a working trie for storing strings. We will create two classes: * A `Trie` class that contains the root node (empty string) * A `TrieNode` class that exposes the general functionality of the Trie, like inserting a word or finding the node which represents a prefix.

Give it a try by implementing the `TrieNode` and `Trie` classes below!

[5]:
```python
## Represents a single node in the Trie
class TrieNode:
    def __init__(self):
        self.children = {}  # Initialize this node in the Trie
        self.is_end_of_word = False  # Flag to represent a word's end

    def insert(self, char):
        # If char is not present in children, add it as a new TrieNode
        if char not in self.children:
            self.children[char] = TrieNode()

class Trie:
    def __init__(self):
        self.root = TrieNode()  # Initialize this Trie (add a root node)

    def insert(self, word):
        # Add a word to the Trie
        node = self.root
        for char in word:
            # Insert char in the trie
            node.insert(char)
            # Move to the child node
            node = node.children[char]
        # Mark the end of a word
        node.is_end_of_word = True
```

```
    def find(self, prefix):
        # Find the Trie node that represents this prefix
        node = self.root
        for char in prefix:
            if char in node.children:
                node = node.children[char]
            else:
                return None
        return node
```

2 Finding Suffixes

Now that we have a functioning Trie, we need to add the ability to list suffixes to implement our autocomplete feature. To do that, we need to implement a new function on the `TrieNode` object that will return all complete word suffixes that exist below it in the trie. For example, if our Trie contains the words `["fun", "function", "factory"]` and we ask for suffixes from the `f` node, we would expect to receive `["un", "unction", "actory"]` back from `node.suffixes()`.

Using the code you wrote for the `TrieNode` above, try to add the suffixes function below. (Hint: recurse down the trie, collecting suffixes as you go.)

[6]:
```
class TrieNode:
    def __init__(self):
        ## Initialize this node in the Trie
        self.children = {}
        self.is_end_of_word = False

    def insert(self, char):
        ## Add a child node in this Trie
        if char not in self.children:
            self.children[char] = TrieNode()

    def suffixes(self, suffix = ''):
        ## Recursive function that collects the suffix for
        ## all complete words below this point
        suffixes_list = []

         # If the node marks the end of a word, add the current suffix to the␣
 ↪list
        if self.is_end_of_word and suffix:
            suffixes_list.append(suffix)

        # Recursively search for suffixes in each child node
        for char, node in self.children.items():
            suffixes_list.extend(node.suffixes(suffix + char))

        return suffixes_list
```

3 Testing it all out

Run the following code to add some words to your trie and then use the interactive search box to see what your code returns.

[7]:
```
MyTrie = Trie()
wordList = [
    "ant", "anthology", "antagonist", "antonym",
    "fun", "function", "factory",
    "trie", "trigger", "trigonometry", "tripod"
]
for word in wordList:
    MyTrie.insert(word)
```

[9]:
```
from ipywidgets import widgets
from IPython.display import display
from ipywidgets import interact
def f(prefix):
    if prefix != '':
        prefixNode = MyTrie.find(prefix)
        if prefixNode:
            print('\n'.join(prefixNode.suffixes()))
        else:
            print(prefix + " not found")
    else:
        print('')
interact(f,prefix='');
```

```
interactive(children=(Text(value='', description='prefix'), Output()),␣
↪_dom_classes=('widget-interact',))
```