Trie

August 25, 2019

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!

```
In [19]: ## Represents a single node in the Trie
         class TrieNode:
             def __init__(self):
                 self.is_word = False
                 self.children = {}

             def insert(self, char):
                 self.children[char] = TrieNode()

             def suffixes(self, suffix = ''):
                 result = list()
                 self.suffixes_recursive(result)
                 return result

             def suffixes_recursive(self, suffix, word=''):
                 for key in self.children:
                     node = self.children[key]
                     if node.is_word:
                         suffix.append(word + key)
                     node.suffixes_recursive(suffix, word + key)

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

             def insert(self, word):
```

```
        current = self.root
        for char in word:
            if char not in current.children:
                current.insert(char)
            current = current.children[char]
        current.is_word = True

    def find(self, prefix):
        current = self.root
        for char in prefix:
            if char not in current.children:
                return None
            current = current.children[char]
        return current
```

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.)

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.

```
In [20]: MyTrie = Trie()
         wordList = [
             "ant", "anthology", "antagonist", "antonym",
             "fun", "function", "factory",
             "trie", "trigger", "trigonometry", "tripod",
             "panda", "pandin", "pandon",
             "chango", "changuito"
         ]
         for word in wordList:
             MyTrie.insert(word)

In [18]: 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-int
```

In [ ]: