Data Structures

Trie Implementation

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Representation

- One simple way is to have a list of M elements to represent M characters
- Initially, each is just a None, indicating no children
- Each list index will, therefore, represent an edge
- Whether we have such edge, the child[idx] can be assigned a new Trie
- This is similar to the binary tree, but now we have M children

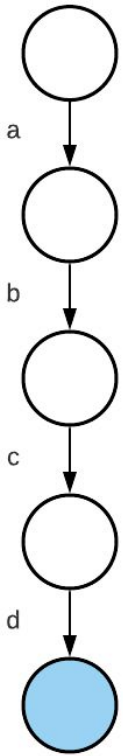
```
class Trie:
    def __init__(self):
        self.child = [None] * 26
        self.is_leaf = False
```

Simple utility

- We will need to convert a character to its idx in the list
- Here is a simple mapping

```
def _to_idx(self, ch):
    # return 0 for 'a', 1 for 'b', ... 25 for 'z'
    return ord(ch) - ord('a')
```

Insertion



- To insert, we iterate through the string letter by letter.
- If an edge doesn't yet exist, we create a new one!
- We always mark the final node (or final character in our string) as a leaf

```
def insert(self, str, idx = 0):
    if idx == len(str):
        self.is_leaf = True
    else:
        cur = self._to_idx(str[idx])
        if self.child[cur] is None:
            self.child[cur] = Trie()
        self.child[cur].insert(str, idx + 1)
```

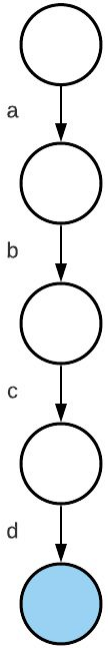
Does a complete word exist?

- You must be able to iterate through the entire length of the string/word
- Its last node must be marked as is_leaf = true

```
def word_exist(self, str, idx = 0):
    if idx == len(str):
        return self.is_leaf # there is a string marked here

    cur = self._to_idx(str[idx])
    if self.child[cur] is None:
        return False    # such path don't exist

    return self.child[cur].word_exist(str, idx + 1)
```



Does a prefix exist?

- The same logic, but returns 'true' as soon as the whole string is successfully iterated through, and found in the tree
- Clearly, all the functions are O(L) time complexity
  - All can be rewritten iteratively to avoid extra O(L) in memory auxiliary space

```
def prefix_exist(self, str, idx = 0):
    if idx == len(str):
        return True

    cur = self._to_idx(str[idx])
    if self.child[cur] is None:
        return False    # such path don't exist

    return self.child[cur].prefix_exist(str, idx + 1)
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."