TRIE (PREFIX TREE) AND REAL-TIME USE IN AUTOCOMPLETE AND SPELL CHECK

- A Trie, also known as a Prefix Tree, is a tree-like data structure used for efficient storage and retrieval of strings. It is particularly useful for operations involving prefixes, such as autocomplete and spell checkers.
- Derive from "retrieval."

Representation of Trie Node

- Trie data structure consists of nodes connected by edges.
- Each node represents a character or a part of a string.
- The root node acts as a starting point and does not store any character.

Node declaration in Trie

```
class TrieNode:
    def __init__(self):
        self.children = [None] *
        26 self.isEndOfWord =
        False
```

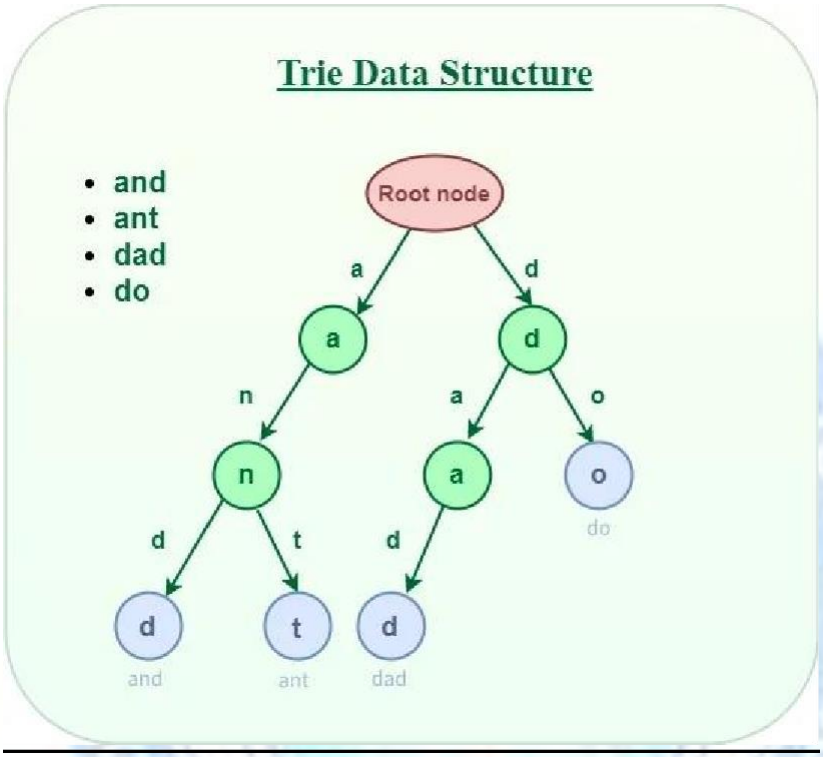
Structure

Nodes: Each node in a Trie represents a single character.

Root Node: The root node typically represents an empty string.

Children: Each node contains a collection (often a dictionary or array) of child nodes, where each child corresponds to the next character in a potential word.

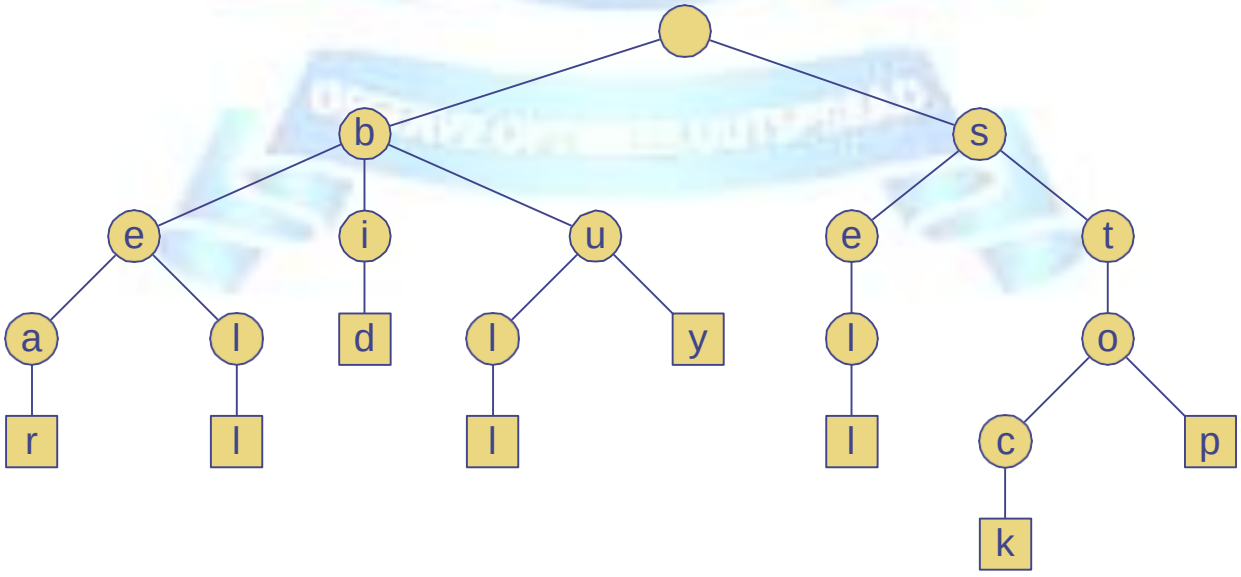
End of Word Flag: Each node also contains a boolean flag indicating whether the path leading to that node represents a complete word.



Tries Example

Example: standard trie for the set of strings

S = { bear, bell, bid, bull, buy, sell, stock, stop }



Operations:

insert(word):

Traverses the Trie, creating new nodes for characters not yet present and marking the final node of the word as
is_end_of_word = True.

search(word):

Traverses the Trie based on the input word. Returns True if the entire word is found and its last character's node is marked as an end of word; otherwise, returns False.

starts_with(prefix):

Traverses the Trie based on the input prefix. Returns True if all characters of the prefix are found in the Trie; otherwise, returns False.

Real time applications - Auto-autocomplete:

Search Engines:

- As users type a search query, autocomplete provides instant suggestions based on popular searches, user history, and trending topics, accelerating the search process and guiding users to relevant results.

Text Editors and IDEs:

- In programming environments, autocomplete suggests code snippets, variable names, and function calls, reducing typing errors and speeding up development.

Forms and Applications:

- When filling out online forms, autocomplete can pre-fill information like addresses, email addresses, and names based on previously entered data, saving time and ensuring accuracy.

Messaging Apps:

- Autocomplete suggests words and phrases as users type, making

communication faster and more convenient, especially on mobile devices.

Content Creation Platforms:

- For bloggers, writers, and content creators, real-time spell check is crucial for producing polished, professional content.

Educational Tools:

- Spell check assists students and language learners in identifying and correcting errors, aiding in the development of accurate writing skills.

Real time applications - Spell Check

Word Processors and Email Clients:

- Real-time spell check highlights misspelled words as they are typed, offering immediate corrections and improving the quality of written communication.

Web Browsers:

- Many browsers include built-in spell checkers that function in text fields on websites, ensuring that online content and messages are free from common errors.