Tries

Computing Lab
CS 69201
Department of Computer Science and Engineering
IIT Kharagpur

Previously

- We have looked at Trees and Graphs.
- We will start a new topic -

TRIE

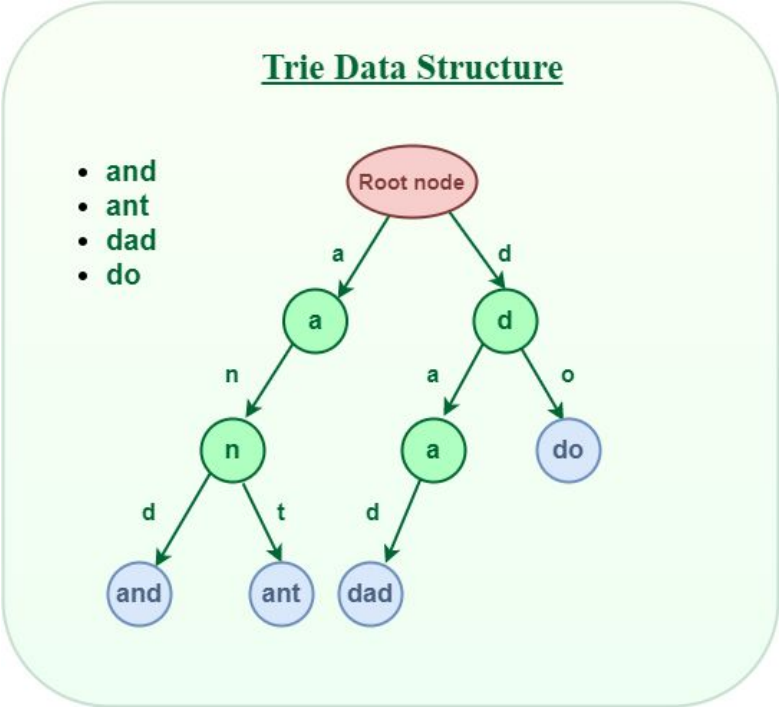
Trie

- Tries are efficient data structures for searching
- If you have a list of words in order of 1B words:
  - Iterative lookup takes O(1B) time
  - Trie reduces to **O(M)** where M is the length of the longest word in the list of words

Trie Structure -Initialization

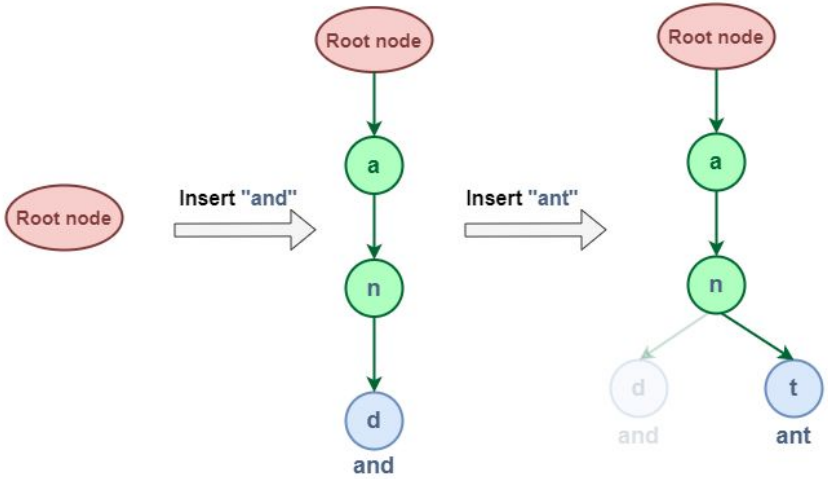
```
struct TrieNode {
   struct TrieNode* child[26];
   bool isEndOfWord;
};

// Function to create a new Trie node
struct TrieNode* getNode() {
   struct TrieNode* node =
     (struct TrieNode*)malloc(sizeof(struct TrieNode));
   node->isEndOfWord = false;
   for (int i = 0; i < 26; i++) {
      node->child[i] = NULL;
   }
   return node;
}
```



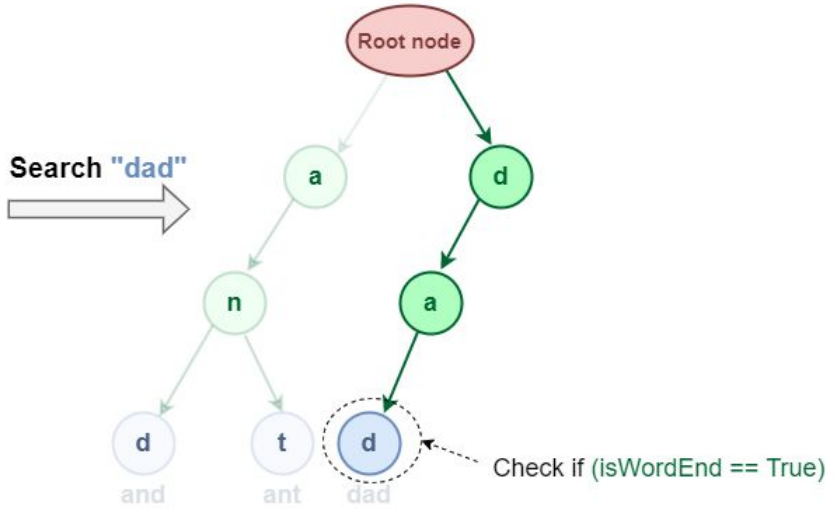
Trie Structure -Insertion

```
// Function to insert a key into the Trie
void insert(struct TrieNode* root, const char* key)
    struct TrieNode* curr = root;
    while (*key) {
        int index = *key - 'a';
        if (!curr->child[index]) {
            curr->child[index] = getNode();
        }
        curr = curr->child[index];
        key++;
    }
    curr->isEndOfWord = true;
}
```



Trie Structure -Search

```
// Function to search a key in the Trie
bool search(struct TrieNode* root, const char* key) {
    struct TrieNode* curr = root;
    while (*key) {
        int index = *key - 'a';
        if (!curr->child[index]) {
            return false;
        }
        curr = curr->child[index];
        key++;
    }
    return (curr != NULL && curr->isEndOfWord);
}
```



Levenshtein Distance using a Trie

Levenshtein Distance same till a common prefix

		k	a	t	e
	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Going from cat to “cats”

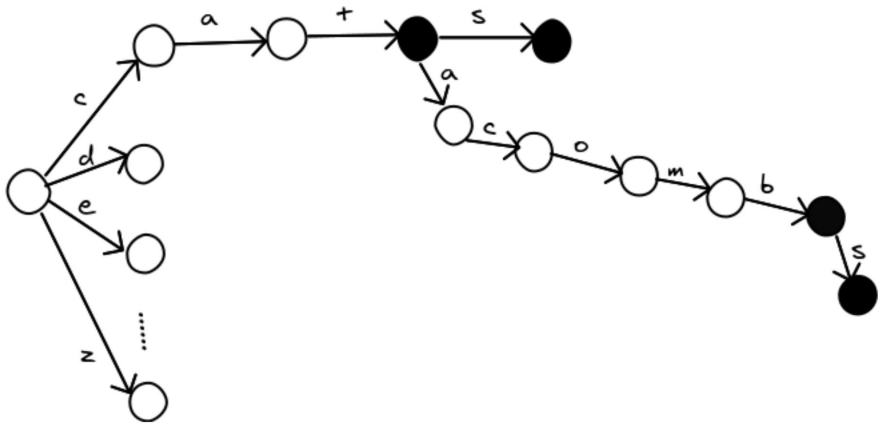
Only the last row changes.

		k	a	t	e
	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2
s	4	4	3	2	2

Solution: Use Trie

We can avoid a lot of work if we can process the words in order, so we never need to repeat a row for the same prefix of letters.

The **trie** data structure is perfect for this! (Here's one with the words cat, cats, catacomb, and catacombs)



Recursive Algo

Input: A target word, and a distance threshold
Output: list of words

1. Construct a trie using the entries from the file
2. Build the first row of edit distance matrix considering empty string
3. Recursively search each branch of the trie
    a. Build one row for the letter, with a column for each letter in the target word
    b. If the last entry in the row indicates the optimal cost is less than the maximum cost, and trie node is a word, then add it.
    c. if any entries in the row are less than the maximum cost, then recursively search each branch of the trie