Author: Blake McGahee

Course: COP3530

Date: 08/07/2025

Tries: Searches and Inserts for Tries - Storing Words

When I first heard the name "Trie," I thought it may have just been a typo for "Tree," and honestly, it kind of is. Tries are sometimes called prefix trees or digital trees for a reason. They are a tree-based data structure specifically designed for storing strings. But unlike a typical binary search tree where you have nodes with a left and right child, a Trie organizes its nodes based on the characters of a word.

Imagine a digital filing system where you have a folder for every letter of the alphabet, and inside the 'C' folder, you have sub-folders for 'A', 'E', 'O', and 'U', and so on. This super-organized, character-by-character structure is what makes Tries so powerful, especially for things like autocomplete on your phone or a spell checker.

At its core, a Trie is built from nodes. Each node can represent a single character, and the entire path from the very first node (the root) down to a specific node represents a word or a word prefix. The root node itself is typically empty and doesn't represent any character. Each node also has a way to point to its children, usually an array or a hash map where the index or key is a character. A crucial part of each node is a Boolean flag, like isEndOfWord, which tells you if the path to that node completes a valid word in the dictionary.

Operation 1: Inserting a Word

Inserting a word into a Trie is a lot like following a map. Let's say we want to insert the word "CAT." We start at the root node.

1. **First Character ('C'):** We look at the first character, 'C'. Does our current node (the root) have a child node for 'C'? Since this is a new Trie, probably not. So, we create a new node for 'C' and link it to the root. We then move our focus to this new 'C' node.
2. **Second Character ('A'):** Now we're at the 'C' node. We look at the next character, 'A'. Does the 'C' node have a child for 'A'? No. So, we create a new node for 'A', link it to 'C', and move our focus there. The path from the root now spells "CA."

Author: Blake McGahee

Course: COP3530

Date: 08/07/2025

3. **Third Character ('T'):** We repeat the process. From the 'A' node, we look for a child for 'T'. We create a new node for 'T', link it to 'A', and move our focus to the 'T' node. The path now spells "CAT."
4. **Marking the End:** We've processed the entire word. At this final 'T' node, we set its isEndOfWord flag to true. This is how the Trie knows that "CAT" is a complete and valid word, not just a prefix.

The great thing about this process is that if we later wanted to insert "CAR," we would follow the exact same path for 'C' and 'A', but at the 'A' node, we'd create a new child for 'R' instead of 'T'. This sharing of prefixes makes Tries so memory efficient when you have many words that start with the same letters.

Operation 2: Searching for a Word

Searching for a word is the reverse of inserting it. We're essentially just retracing our steps through the tree. Let's check if the word "CAT" exists.

1. **Follow the Path:** We start at the root node. We look for a child node for 'C'. It exists, so we move there. From the 'C' node, we look for 'A'. It exists, so we move there. From the 'A' node, we look for 'T'. It exists, so we move to the final 'T' node.
2. **Check the Flag:** We've successfully navigated the entire path for "CAT." The last step is to look at the isEndOfWord flag on the final 'T' node. Since we set it to true when we inserted the word, our search is successful, and we can confidently return true.

Now, what if we tried to search for a word that isn't there, like "DOG"? We'd start at the root, look for a child for 'D', but since we never inserted a word starting with 'D', no such child would exist. At this point, the search fails immediately, and we return false.

What about a word that is a prefix, but not a full word, like "CA"? We would successfully find the node for 'C' and then the node for 'A'. But after we've processed all the characters in "CA," we check the isEndOfWord flag on the 'A' node. Since we only inserted "CAT," the 'A' node's flag is still false. So, even though the prefix exists, the word "CA" does not, and we return false.

Why Tries are a Big Deal

The elegance of a Trie is in its efficiency. Both searching for a word and inserting a word have a time complexity that's directly related to the length of the word, which is

Author: Blake McGahee

Course: COP3530

Date: 08/07/2025

super-fast. This makes Tries a go-to for applications that deal with large numbers of strings, especially when prefixes are important. You get the benefit of super-fast lookups without the possibility of hash collisions that you might find in a hash map. It's a great example of a data structure that sacrifices a little more memory for some serious speed and functionality gains.