

Implement Trie Using Array [LeetCode](#)

A trie (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

implements a trie data structure using two classes: **Trie** and **TrieNode**. The trie is a tree-like data structure used for efficient retrieval and storage of strings. Each node in the trie represents a character in a word, and the edges between nodes represent characters. The trie is used to store and efficiently search for words.

TrieNode Class:

- **TrieNode** represents a single node in the trie.
 - Member Variables:
 - **value**: The character value of the node.
 - **children**: An array to store children nodes for each lowercase letter.
 - **isEndOfWord**: A flag indicating if this node marks the end of a word.
- **TrieNode** Constructor:
 - Initializes the node with the given character value.
 - Sets the end of the word flag to false.
 - Initializes the children array to **nullptr** for each lowercase letter.

Trie Class:

- **Trie** represents the trie data structure.
 - Member Variable:
 - **root**: The root node of the trie.
- **Trie** Constructor:
 - Initializes the trie with an empty root node.
- **insert** Function:
 - Inserts a word into the trie.
 - **Time Complexity: $O(m)$, where m is the length of the word.**
 - **Space Complexity: $O(m)$.**

- **search** Function:
 - Searches for a word in the trie.
 - **Time Complexity: $O(m)$, where m is the length of the word.**
 - **Space Complexity: $O(1)$.**
- **startsWith** Function:
 - Checks if there is any word in the trie that starts with the given prefix.
 - **Time Complexity: $O(m)$, where m is the length of the prefix.**
 - **Space Complexity: $O(1)$.**
- **remove** Function:
 - Removes a word from the trie.
 - Calls Helper function for removing a word from the trie.
 - Recursively removes the word from the children of the current node.
 - **Time Complexity: $O(m)$, where m is the length of the word.**
 - **Space Complexity: $O(m)$.**