Tries

So, I recently learned a data structure called a Trie. Tries are basically Tree-like non-linear data structures. They are N-ary trees. So, as Bi-nary trees have two children(nodes) same way N-ary trees can have n number of children(nodes). It is used to store words or strings in a way that makes it easy to search for them. Imagine it like a big tree where each branch represents a letter, and together, the branches form words. And trust me, it is way more useful than it sounds—especially if you have ever wondered how your phone or Google knows what you are trying to type before you do.

Working:

1. The root node (starting point) is always kept empty in the Tries data structure. They do not represent any letter.
2. Every node or point (child node) after the root of the tree represents letters of words.
3. If words start with the same letters, they share the same path in the tree until they start to differ.

Why Bother with Tries?

1. Autocomplete Magic: They are great for suggesting possible word completions when you start typing the beginning of a word. Let's take *Google Search*, if we start typing "bo" in Google Search, the autocomplete feature might suggest words like "book," "boss," or "body." This is because Google's system uses something like a Trie to quickly find and suggest words that start with "bo" from a huge list of possible words.
2. Efficient Search: This Data Structure allows us to quickly check if a word or the beginning part of a word (called a prefix) is in a list of words.
3. Small Memory: Instead of storing each word separately, the Tries save memory space by sharing common prefixes (the starting letters of multiple words that are common). For example, for the words "bat" and "bad", the Trie shares the branches for 'b' and 'a'. Only the last letter 't' gets its own branches.

Let's Talk Functions

1. insert() - Adds a word to the Trie by moving down and creating new branches (nodes) for each letter in the word if they do not already exist and then marks the last node as the end of the word.
2. search() - It checks if a word is in the Trie. For this function, just follow the branches. If you can trace the whole word and land on a marked spot at the end, bingo, you have found your word.
3. autoComplete() - This function will search and find all words in the Trie that start with a prefix we give.
4. remove() - This function will remove the word from the Trie. Just traverse to the nodes for each letter in the word. If we reach the end, unmark the last node as the end of the word and if there are no other words using those branches, you can remove them to save space.