**TRIES DATA STRUCTURE**

**Introduction to TRIE Advanced Data Structure**

A TRIE (pronounced as "try") is an advanced tree-based data structure used for efficient information retrieval. It is also known as a prefix tree because it stores words such that common prefixes are shared among multiple entries. Unlike binary search trees (BSTs) that store keys in a sorted order, TRIE structures characters level by level, making operations like searching and prefix matching significantly faster. TRIEs are particularly useful for applications involving dictionaries, autocomplete systems, and IP routing.

**Important Operations on TRIE**

1. **Insertion:**
   * A word is inserted by traversing the TRIE character by character, creating new nodes if necessary.
   * Time Complexity: O(L), where L is the length of the word.
2. **Search:**
   * A word is searched by checking for the presence of each character in the TRIE.
   * Time Complexity: O(L).
3. **Deletion:**
   * A word is removed by traversing down the nodes and deleting nodes if they are no longer part of another word.
   * Time Complexity: O(L).
4. **Prefix Matching (Auto-complete):**
   * Searching for words starting with a given prefix involves traversing to the prefix node and collecting all its descendants.
   * Time Complexity: O(P + W), where P is the prefix length and W is the number of matched words.
5. **Word Count and Frequency Count:**
   * TRIE nodes can store additional information such as word frequency, useful in NLP applications.

**Complexity Analysis for Storing a Large Number of Words**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Structure** | **Insertion Complexity** | **Search Complexity** | **Space Complexity** |
| Array | O(1) | O(N) | O(N \* L) |
| Linked List | O(1) | O(N) | O(N \* L) |
| BST | O(log N) | O(log N) | O(N \* L) |
| TRIE | O(L) | O(L) | O(N \* A) |

Where:

* N is the number of words.
* L is the average length of words.
* A is the alphabet size.

TRIE offers faster search operations compared to arrays and linked lists, and it avoids the balancing issues of BSTs. However, its space consumption can be high, especially when storing sparse words.

**How to Use TRIE as a Map?**

A TRIE can function as a map (key-value store) by associating values with words. Instead of marking only the end of a word, a TRIE node can store a corresponding value. For example:

* Insert("apple", 100) → Associates the word "apple" with value 100.
* Search("apple") → Retrieves 100.

This method is useful for implementing dictionary-based key-value storage systems efficiently.

**Real-World Applications of TRIE**

1. **Autocomplete and Spell Checking:**
   * TRIE is widely used in search engines and text editors for predicting user input based on prefixes.
2. **IP Routing (Longest Prefix Matching):**
   * Routers use TRIE-based structures to match IP addresses to their longest common prefixes for efficient packet forwarding.
3. **Dictionary and Lexicographic Sorting:**
   * TRIE enables storing words in sorted order and facilitates fast dictionary lookups.
4. **Genome Sequencing:**
   * TRIE helps in storing and querying DNA sequences efficiently.
5. **Data Compression (T9 Keyboard, Huffman Coding):**
   * TRIE can be used in text prediction and encoding schemes to reduce storage space.
6. **Security (Keyword Filtering, Spam Detection):**
   * TRIE can quickly detect forbidden words or spam patterns in real-time systems.
7. **Search Engine Indexing:**
   * Search engines use TRIE for efficient keyword-based indexing and retrieval of documents.