Khadichabonu Valieva

Dr. McDaniel

CSC 307 H001

29 November 2022

Research Essay: Trie Data Structure

1. Overall use and description of the Trie

Trie data structure is defined as a Tree based data structure that is used for storing some collection of strings and performing efficient search operations on them. The word Trie is derived from retrieval, which means finding something or obtaining it. A trie can be used to sort a collection of strings alphabetically as well as search whether a string with a given prefix is present in the trie or not.

2. A use case for the Trie(other than spell checking). Examples:

Auto complete. When the user types in a prefix of his search query, we need to give him all recommendations to auto complete his query based on the strings stored in the Trie. We assume that the Trie stores past searches by the user. In the example, given in the GeeksforGeeks website, if the trie store {“abc”, “abcd”, “aa”, “abbbaba”} and the user types in “ab” then he must be shown {“abc”, “abcd”, “abbbaba”}.

Other uses of Trie data structure: Auto correct, Longest Prefix Matching (also called the Maximum Prefix length math, is used in networking by the routing devices in IP networking), browser history (web browsers keep track of history of websites visited by the user).

3. The overall structure of the Trie (that can store words in the English Language).

Every node of Trie consists of multiple branches. Each branch represents a possible character of keys. In the source, they mark the last node of every key as the end of the word node (**isEndOfWord**). The **isTop** represents only for the special degenerate top-level of the trie.

a. Provide both a visual diagram and a UML table for its class definition.

A structure to represent nodes of the English alphabet can be as follows:

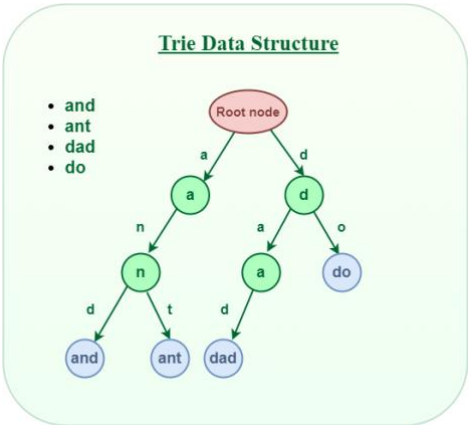

Figure 1. Trie (GeeksforGeeks)

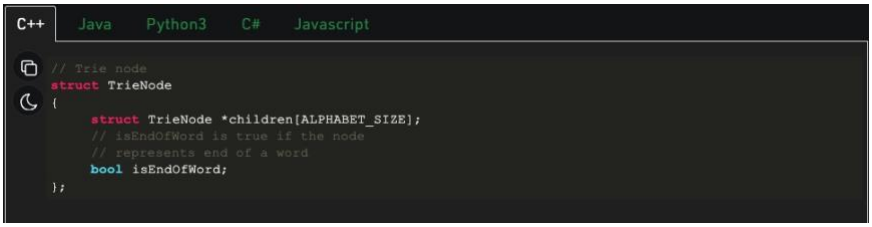

Figure 2. TrieNode (GeeksforGeeks)

Trie Node
+ wordFragment: string + children: [TrieNode] + isEndOfWord: boolean + isTop: boolean

Table 1. UML representation of a single Trie Node (Eukland)

4. The primary operations of the Trie (insert, search, and remove)

Insert Operation:

a. **Purpose:** to insert new strings into the Trie data structure.

b. **Pseudocode:**

```
void insert (string s)
{
        for (every char in string s)
        {
                if (child node belonging to current char is null)
                {
                        child node = new Node();
                }
                current_node = child_node;
        }
}
```

c. **Algorithm:**

- Every character of the input key is inserted as an individual Trie node.
- The key character acts as an index to the array children.
- If the input key is new, construct non-existing nodes of the key, and mark the end of the word for the last node.
- If the input key is a prefix of the existing key in Trie, simply mark the last node of the key as the end of a word. (Singh)

d. **Complexity Analysis:** O(n), where n is the size of the string to be inserted since we must go through n iterations for the whole trie depth.

Search Operation:

a. **Purpose:** to search whether a string is present in the Trie data structure or not.

b. **Pseudocode:**

```
boolean search (string s)
{
        for (every char in string s)
        {
```

```
                if (child node is null)
                {
                return false;
                }
        }
        return true;
}
```

c. **Algorithm:**

- Like the insert operation, but it only compares the characters and moves down.
- If the **isEndOfWord** field of the last node is true, then the key exists.
- The search can terminate due to the end of a string or lack of key in the trie (Singh).

d. **Complexity Analysis:** O(n), where n is the length of the word to be searched since we need to travel down the length of it.

Remove Operation

a. **Purpose:** to delete strings from the Trie data structure.

b. **Pseudocode**

1. If key is not present in trie, then we don't modify trie in any way.
2. If key is not a prefix nor a suffix of any other key, and nodes of key are not part of any other key then:
    a. All the nodes starting from root node(excluding root node) to leaf node of key should be deleted.
3. If key is a prefix of some other key, then:
    a. Leaf node of that key should be marked as "not a leaf node". No node should be deleted in this case.
4. If key is a suffix of some other key, then:

   a. All nodes of key which are not part of the second key should be deleted.

5. If key is not a prefix nor a suffix of any other key but some nodes of key are shared with some other key, then:
   a. Nodes of key which are not common to other keys should be deleted and shared nodes should be kept.

c. Algorithm:

- During this operation, we delete the key in bottom-up manner using recursion.
- Key may not be in the trie, then Remove operation should not modify trie.
- We first make a recursive call to delete the node which is child of the currentNode
- We check if child node was deleted in step#1, then, if it was deleted, we check if this currentNode can be deleted. (Singh)

d. Complexity Analysis: O(n), where n is the length of the string to be deleted since we need to traverse down its length to reach the leaf node.

Works Cited

Eklund, Daniel. “Implementing the Dictionary TRIE: Defining the Data Structures.” *Implementing the Dictionary Trie: Defining the Data Structures*, 8 June 2010, http://merrigrove.blogspot.com/2010/06/implementing-dictionary-trie-defining.html .

“Introduction to Trie - Data Structure and Algorithm Tutorials.” *GeeksforGeeks*, 1 Nov. 2022, https://www.geeksforgeeks.org/introduction-to-trie-data-structure-and-algorithm-tutorials/.

Singh, Shreya. “Applications of Trie Data Structure.” *OpenGenus IQ: Computing Expertise & Legacy*, OpenGenus IQ: Computing Expertise & Legacy, 2 Apr. 2020, https://iq.opengenus.org/applications-of-trie/ .