

1.0 Class Design

TrieNode

- Resize children on init so it is never resized on insert
- Track number of children so inserting is $O(1)$

2.0 UML Diagram

3.0 Function Design

4.0 Runtime

4.1 INSERT

To insert an n class classification, `Trie.insert()`, will have to traverse n nodes. Traversing each node involves searching the children vector for a child with a label that matches the next class, which in the worst case is $O(15)$. If any of the intermediate classes are missing, or once the terminal class is reached, they are added to the tree with `TrieNode.addChild()` which appends to the children vector which is an $O(1)$ operation, in the worst case, this happens n times. The worst case time complexity for INSERT is then $O(n * (1 + 15)) = O(n)$

4.2 CLASSIFY

To classify an input, `Trie.classify()` will traverse through, in the worst case, N nodes and at each node it will call `LabelText` on the input with the current children as the possible classes. Getting the labels of the current children is $O(15)$ in the worst case. The worst-case time complexity of CLASSIFY is then $O(N * (15 + 15)) = O(N)$

4.3 ERASE

To erase an n class classification, `Trie.erase()` will have to traverse n nodes. Once it has reached the terminal node of the classification, it will have to delete the node, which since the node is a terminal node, guarantees deletion will be $O(1)$. To properly delete the node, the pointer in the children vector of the parent node also needs to be set to `nullptr`, which takes $O(15)$ in the worst case. The worst case time complexity of ERASE is then $O(n * 15 + 1 + 15) = O(n)$.

4.4 PRINT

To print a Trie, `Trie.print()` recursively calls `Trie.printHelper()` on the children of each node starting from the root. `printHelper` collects the labels of the nodes that called it and when it reaches a terminal node, appends the string of traversed nodes to the vector of traversal strings and then returns, both operations are $O(1)$. For each classification in the tree, in the worst case `printHelper` is called $N * C$ times where C is the number of classes in the classification. Once all the calls to `printHelper` have returned, `print()` iterates over the vector of traversal strings and prints them out. In the worst case, there are N traversal strings to be printed out. The worst case time complexity for PRINT is then $O(N * C * (1 + 1) + N) = O(N)$.

4.5 EMPTY and SIZE

The size of the Trie, or the number of classifications, is tracked by incrementing and decrementing the size attribute whenever a Trie node becomes a new terminal node or ceases to be one respectively. Thus returning the size of the Trie is $O(1)$ since `Trie.getSize()` just return the attribute value. `Trie.empty()` is also $O(1)$ since it just makes a call and returns the Boolean comparison to 0. The worst case runtime for EMPTY and SIZE then is $O(1)$.

4.6 CLEAR

`Trie.clear()` calls `delete` on the root which calls the `TrieNode` destructor which calls `delete` on each of the children of the `TrieNode`. For a Trie with N nodes, N memory deallocations occur which each take $O(1)$ time. Then worst-case time complexity for CLEAR then is $O(N \cdot 1) = O(N)$