ECE 250 - Project 3
Tree
Design Document

Jamie Yen, UW UserID: j6yen

Nov. 7th, 2022

Overview of Classes

Class 1: myTrie

Description:

Represents a trie type 26-array search tree which each node of the tree consists of multiple branches. It can be used for searching, storing vocabulary, erasing keys, checking spelling, etc.

Member variables(all private):

1. TrieNode type pointer root: store the root of the trie
2. TrieNode type pointer root: used in erase function to keep track of which node should be mark as leaf
3. int count: keep track of number of words in the tree
4. string eraseStatus: record the status of deletion of node

Member functions(all public):

1. **insert**: three cases, i. no nodes added and word is already in the list-> failure
ii. no nodes added but word were not in the list-> success
iii. added new nodes-> success
1. **load**: inserts all the words in corpse to the trie i. iterate over the given word ii. the value is nullptr? make a new node and add a branch to it/ traverse to the new node
2. **insert**: i. is the root empty? creates a new node to root ii. if the input is not a lower letter, throw invalid argument and return iii. iterate through the word, add new nodes and branches if there's no branches iv. no nodes get added?is the last node an last character? print failure/ mark it as the last character, print success v. nodes were added? print success
3. **searches**: i. input invalid? throw invalid argument ii. iterate through the trie, find whether the given word exists in trie iii. the ending node is the last character of a word?
4. **erase**: call the removeNode function and prints out the result
5. **removeNode**: i. input invalid? throw invalid argument ii. recursively call removeNode function until reaching the last node to check which scenario it is iii. the deleted word is a prefix of other words in trie? mark the isLastChar false iv. the word doesn't share the same prefix, delete the whole word v. the deleted word shares the same prefix with other words? delete the words other than the prefix
6. **printTrie**: i. is the root empty? do nothing ii. call the printWord function
7. **printWord**: i. travers from the root node and add the parent key of the child node in the "string word" ii. recursively call the printTrie until reaching a leafnode iii. mark the end of the string to escape character
8. **spellcheck**: i. iterate through the word to compare the word with the word in trie ii. no word get matched? print an empty line iii. find the same word? print correct iv. pass the node where the last matched node with the input is to the pirntCheckedWords function, print out the suggested words
9. **printCheckedWords**: i. works similar to printWord but has a prefix passing into it ii. recursively call the printTrie until reaching a leafnode iii. prints prefix+words after the prefix
10. **empty**: i. word count in trie is 0? prints "empty 1" ii. prints "empty 0"

11. **clear**: i. reset cout=0 ii. root empty? pass the whole trie to clear_children
12. **clear_children**: i. step through every node of the trie ii. node is not a leaf? check all its children, call clear_children on its children and recursively delete all its children ii. node not empty? delete the node.
13. **printSize**: i. prints number of words count in the trie

Constructor: creates a TrieNode type new root
Desctructor: call the clear function to deallocate all the nodes and ensure no memory leaks

Class 2: Trienode

Description:

A class representing a node, which consists of an 26 entries array that each entry represents an alphabet of the letter and also two bool values.

Member variables(all privates):

1. bool isLastChar: whether the node is the last character of a word
2. bool isLeaf : whether it is a leaf of the trie tree
3. TrieNode type array pointer alphabets: the entry of the array points to another letter if the array is no

Constructor: initialize all the entries of the alphabets array to null
Destructor: not used since myTrie class is taking care of deallocation

Class 3: illegal_exception

Description:

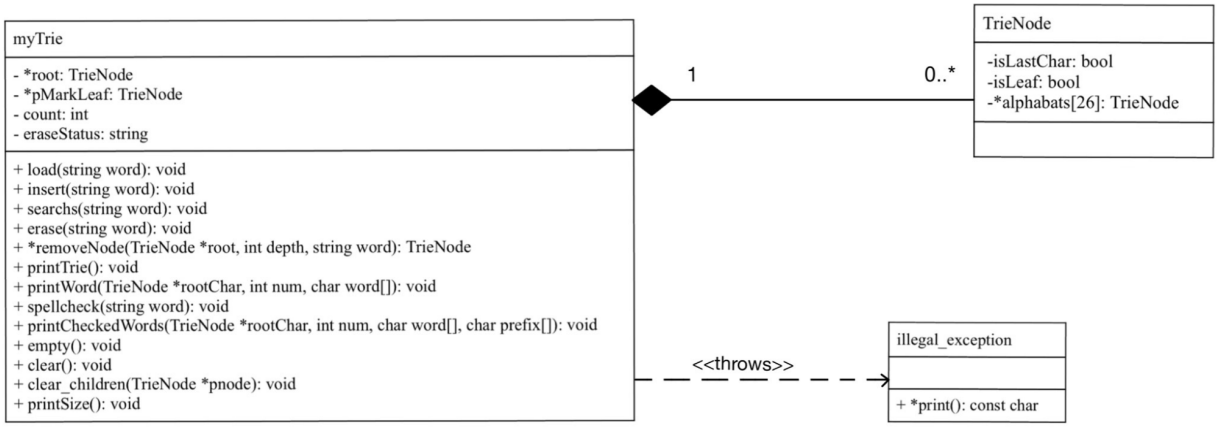
An exception class that inherits the std::exception class to handle input words that contain any characters other than those of the lower-case english alphabet.

Member function(public):

constant char type pointer print: return the string illegal argument

Construtor: not used
Destructor: not used

UML Class Diagram



Test Cases

1. Duplicate key insertion test: insert applepie, apple, any, anything, and insert applepie and apple again, will result in failure insertion, when I print the trie I should see the output to be apple, applepie, any, anything
2. Insertion and spellcheck test: if I do i card, i carman, e carman, p I will be getting card, carman card, if I do e carman, spellcheck carman, I will be getting card
3. Spellcheck test: insert you, young, your, youth, yay, apple and spellcheck yukk, yes, you, boo, I will be getting you young your youth/ yay you young your youth/ correct / (no output)
4. Deletion test: carmen card carmel are already in the trie, if I try to delete carmen, and print the trie, card and carmel should still be in the trie although both those three words share the same prefix, then I will try to delete candy, and print the trie again, then I should see the deletion fails and card carmel get printed

Performance

The expected runtime for insert, searches, erases are O(n), where n is the number of characters in word. This can be achieved since we are using trie to implement these operations, when the above operations were called, we don't need to use overhead of hash functions, or go through the whole trie tree, instead we travel through n branches of nodes, and add them if those branches were not in the trie.
The expected runtime for prints and spellcheck are O(N), where N is the number of words in trie. To ensure this, I make sure that all the nodes in the trie tree are only visited once when I print the whole trie.