**ECE250-Project 2**

Shihang Wang

S652wang

Mar 29th,2021

1. **Overview of Classes**

class Node:

Class Node has value of char type and isTerminal of bool type to determine it is word end or not. In addition, an array to store ‘a’ to ‘z’ and also the size which is the child of node. Based on the bool typed functions to determine is empty, is the end of the word, add words, and remove words. Meanwhile, get the char value and get child.

class Root:

Class Root has the array to store ‘a’ to ‘z’ and the size which is the child of node. Based on bool typed functions to determine the operation insert, erase, search, and empty. It prints the tree and the tree after auto complete operation. In addition, it also clears the tree and gets the size of the tree.

class illegal\_exception:

Exception Handling to handle invalid characters in the word which created by commands “i”, “s”, and “e”. When illegal\_exception happens, return the message of the string type “illegal argument”.

Relation: Node class associates and composites the Root class and the multiplicity can be described as “zero to many”, the illegal\_exception class handle the exception during the trie test.

1. **UML class diagram**

Diagram

Description automatically generated

1. **Details on design decision**

Class Node:

The first constructor of class node set the pointer to children to null, value and size to 0, and is Terminal to true. The second constructor of class node takes the str and index, if the string length is 0, call the Node () and return. Then set the pointer to children to null and value to the str with index. If the index is equal to the str length -1, then the size is 0 and is Terminal to true. If not, set the pointer to children and set the is Terminal to false.

The destructor deletes the pointer used for children. And set the size to 0.

getEmpty() return the size==0, getTerminal() return isTerminal, getChild() return the child with index, and getValue() return the value.

addWords() keep matching until the character is not in the chain then create new node. In addition, increase the size.

removeWords() keep matching until the last one is found with the word end is larger than 0, then update the word end to 0 and call getEmpty(). In addition, decrease the size.

Class Root:

The constructor of the class Root set the pointer to children to null and size to 0.

The destructor deletes the pointer used for child and set to null.

insertWord() insert the chain using a new node or not using it calling addWords() , and eraseWord() erase the chain by calling removeWords().It is not just set last node to false, since it will not free the memory until the clear() is called.

searchWord() start at the child of the root using getChild() and match all the way down. If the node is null or less than the chain, return false. Return true if word in the tree. If it exceeds the chain, we only determine node isTerminal(). Print the chain with print().

autoComplete() is similar with searchWord(), but we do not need to determine the word end. It prints the chain with same prefix.

In the end, clear() is called to clear on all sub-trees and delete the node, then set to null.

Class illegal\_exception:

Exception Handling to handle invalid characters in the word.

1. **Test cases**

Basic cases: For any characters which is not ‘a’ to ‘z’(such as space) is consider to be a illegal argument.

When erase a word, the word with same prefix but not same length should consider to be not in the trie tree which should return failure.

The empty tree to do the print and autocomplete operation.

Corner cases: The word length can be any size: The chain can be any length, but the array of child is always 26.

The tire tree is longer until the clear() is called, so we need to free memory when erase words.

To consider about memory, each node has an array of child pointers. To save the memory, the memory should only be used for alphabets instead of null pointers. In addition, after delete the node, it should always set to null to avoid the memory leak.

eraseWord() function should also free memory when erase words

1. **Performance considerations**

It is case sensitive.

The time complexity of creating a tire is O(k\*m). (k is the number of words and m is the average length of each word)

insertWord() & addWords() have a time complexity of O(n), it determined by the length of the word we are inserted. Recursively check or create a node until the end of the chain. It has a space complexity of O(n). In the worst case, the new node we insert do not have same prefix. Which we need to add n nodes.

searchWord() has a time complexity of O(n). It always searches the next word by a while loop. The worst case needs n operation. The space complexity of O(1).

eraseWord() & removeWords() have a time complexity of O(n). it determined by the length of the word we are inserted. Recursively check or update the word end takes O(n) times.

For insert, search and erase funtions will have a tight bound of O(n).

print() has a time complexity of O(N). The for loop takes O(N) times to print.

autoComplete() has a time complexity of O(N) and space complexity of O(1). Using the while loop to get child takes O(N) times.

Clear() has a time complexity of O(N) The for loop takes O(N) times to delete the child and set it to null.

The upper bound of print(), autoComplete(), clear() should be O(N), since we know how many words are in the tree.

empty() has a time complexity of O(1). It returns size==0.

getSize() has a time complexity of O(1). It returns size.

getEmpty(), getTerminal(), \*getChild(), getValue() are all have a time complexity of O(1).