Boris Jancic

20908506

ECE 250 Project 3

To implement the trie I create a Trie class to handel all the functionality as given by the requirements. High level functions to use the trie were made public, while low level functionality, as well as member variables, were made private to create this ADT. In the main file there exists a loop which prompts the user for inputs. This loop is not apart of the Trie, as there are situations where you could use one without such a loop.

Class Trie:
Contains all the functionality to manipulate and use the trie, as well as necessary data members. High level functions are public, while low level functions and members are made private to ensure proper use. The implementation comes in the form of a head array of letters (Node) to store which letters (and consequently which words) are stored in the trie. Letters are given by another array linking to the next letter(s), while ends are denoted by a nullptr. The Trie also contains a size varialbe to allow for quick access of the number of words stored. Upon the creation of a Trie, the head is initialized and the size is set to zero. In order to prevent memory leaks, the head is deleted upon destruction. As the Node destructor is recursive, destroying the Trie will also destroy all the childerent of the head node and consequently all dynamically allocated memory.

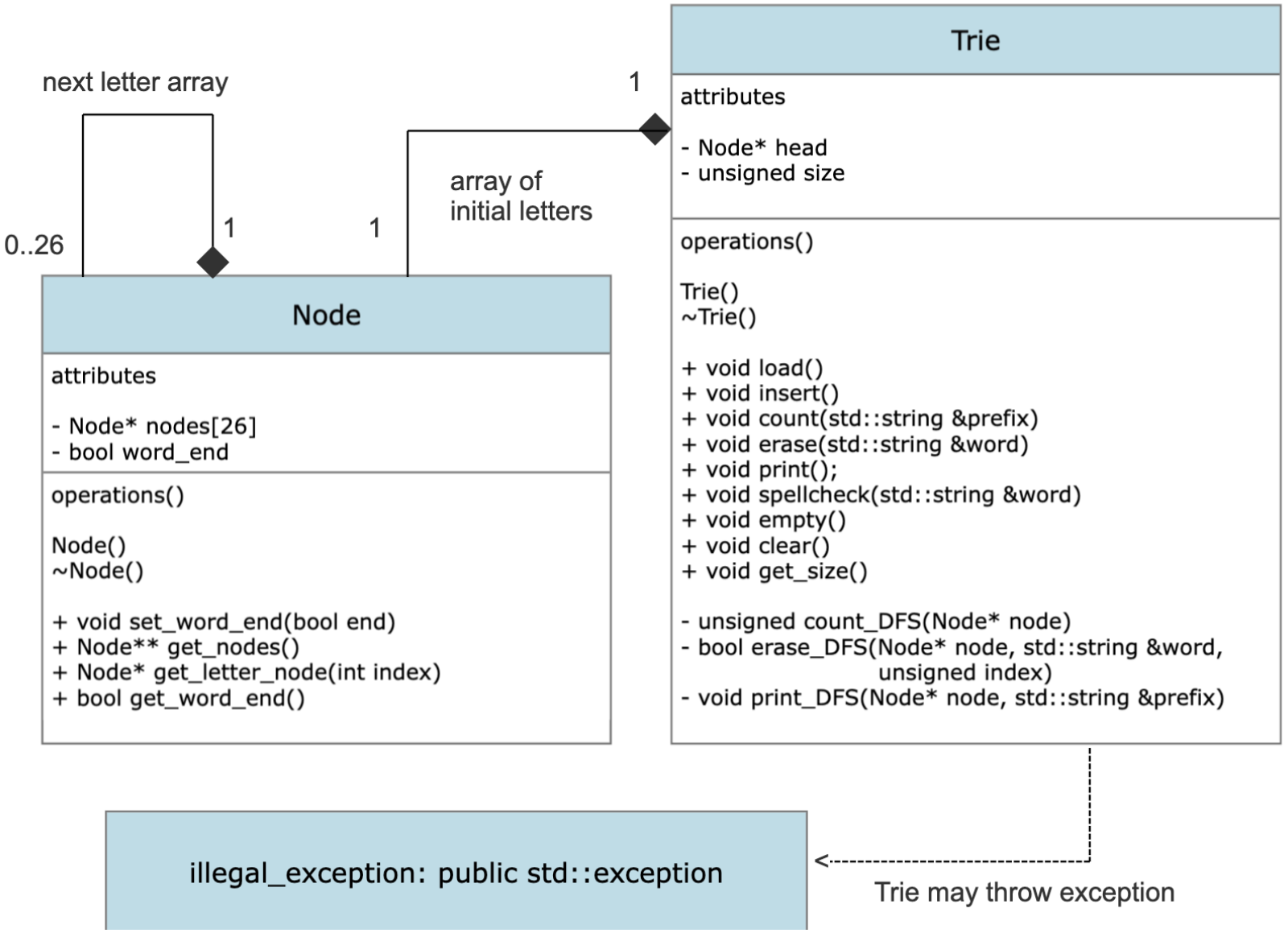
Class Node:
Stores an array of Nodes for each letter in the alphabet to denote all next letter(s) for the given word(s). A value of nullptr in the array represents an end to such continuation of the word in such direction. There also exists a boolen flag to mark weather the current node represents the end of a word in order to differentiate words which are also prefixes to other words. To ensure proper use, relevant public getter and setter methods are used, while data members are kept private. Upon creation, all values in the next letter array are initialized to nullptr. In order to prevent memory leaks, upon destruction, all values in the next letter array are destroyed. This distruction is recursive by nature, and will therefore also delete all child elements and so forth.

Class illegal_exception: public std::exception:
Contains all the members of std::exception. Effectively acts as a name change in its current form, but could act as an extention to the standard libraries exception handling if modified and added on to. When an illegal exception occurs, an error will be thrown, and handled by a try/catch. This is functionality is used in the Trie class and main loop.

Runtimes:

- Load: O(n), n is the number of letters is the corpus
  This is because we must insert every letter of every word into the trie (see Insert below)
- Insert: O(n), n is the number of characters in the word
  To insert a word we must loop through each letter and traverse the tree, updating any new letters and setting an end word flag at the end. As we must loop though n elements, and letter/end word updates are constant, we get O(n) runtime.
- Count: O(N), N is the number of words in the trie
  In the worst case, if all the words stored in the trie have the given prefix, we must traverse through each word to tally the count using DFS, this gives O(N) as there are N possible words to traverse.
- Erase: O(n), n is the number of characters in the word
  Inorder to erase a word we must traverse all n elements. N letters gives n traversals, each traverse involves one delete (or simply a end word flag change). This gives is O(n) runtime.
- Print: O(N), N is the number of words in the trie
  Inorder to print every word, we must traverse every word (and every letter in every word) using DFS, if we let the max letters in a word be a constant, and as printing is constant, we get O(N) runtime (or O(n) for each letter in the trie).
- Spellcheck: O(N), N is the number of words in the trie
  In the worst case, if ll the words in the trie have the prefix of the last correct substring, we must print out all words in the trie, N words and constant print time gives O(N) runtime.
- Empty: O(1)
  To check if empty, we simply have to check weather the number of words is 0. As we actively store and update the current number of words, this function runs in constant time as all tht is required is a single memory access, and no computation.
- Clear: O(N), N is the number of words in the trie
  To clear the trie, we must travere each letter in the trie. If we let the max word length be a constant, this gives us O(N) time, as to delete every letter we must use DFS on every letter gring O(n) or O(N) for each word, and each deletion takes constant time.
- Size: O(1)
  As we actively store and update the current number of words, this function runs in constant time as all tht is required is a single memory access, and no computation.

UML Diagram


Trie
attributes
- Node* head
- unsigned size
operations()
Trie()
~Trie()
+ void load()
+ void insert()
+ void count(std::string &prefix)
+ void erase(std::string &word)
+ void print();
+ void spellcheck(std::string &word)
+ void empty()
+ void clear()
+ void get_size()
- unsigned count_DFS(Node* node)
- bool erase_DFS(Node* node, std::string &word, unsigned index)
- void print_DFS(Node* node, std::string &prefix)
1
array of initial letters
1
next letter array
0..26
1
Node
attributes
- Node* nodes[26]
- bool word_end
operations()
Node()
~Node()
+ void set_word_end(bool end)
+ Node** get_nodes()
+ Node* get_letter_node(int index)
+ bool get_word_end()
illegal_exception: public std::exception
Trie may throw exception