CSE 503 Project 1

Alex Myers

Sun March 2nd, 2024

The goal of this project is to develop a Trie data structure capable of taking in a file, constructing the trie, and displaying a command-line interface for a user to query words stored within the structure. This interface is required to have an auto-completion functionality for incomplete words present in the trie, as well as the capability to recommend words when a query does not match any entry in the structure.

Starting with the definition of the trie's structure in the header file, “trie.hpp”, the node structure has a Boolean member which is used to indicate if a word has occurred in a traversed path. Additionally, an unordered map is utilized to store the children of that node. Opting for a map, as opposed to an array, is more efficient in terms of time complexity. The map operates as a hash table, hashing the key (character) into indices and storing the value (“TrieNode” pointer) accordingly. This choice results in an average time complexity of O(1) for all operations, including search—which is frequently used within the Trie structure. In contrast, an array would have an average time complexity of O(n).

A computer code with text

Description automatically generated with medium confidence

The “Trie” structure has several public members outlining methods for reading, inserting, and searching for words. Note the private members associated with the trie as well. These include a “validWords” unordered set, which temporarily stores words when determining what to display for the autocomplete portion of the assignment. Additionally, there is a “suggestion” unordered set, which temporarily stores words when determining the recommended words to display based on a query.

A computer code with text

Description automatically generated with medium confidence

Part I

The "insertWord" method, designed for the trie to process a file like "dictionary.txt," takes a string as a parameter with a return type of void. It starts by storing the root in a temporary variable named "current." The code then iterates through the string, checking whether the letter in the word is present in the children by using the "find" method on the unordered map.

If the letter is absent in the map, indicating nothing was found, a new TrieNode is added to the children map with the key set to the missing letter.

Regardless of finding the letter or not, the current node is set to the node in the map corresponding to the current letter, effectively moving to the next layer in the trie. This process continues until the entire word is traversed. At the word's end, the current node marks the "complete" member as true, signifying the existence of a word at that path.

A computer code on a white background

Description automatically generated

The "insertFile" method takes a file path (a string) and reads the file. For each line, it adds the information to a string, passing that information into the insert method. This continues until the entire file is processed. If the string/path provided to this method does not exist, a console message displays.

A computer code with text

Description automatically generated with medium confidence

Part II

The "searchTrie" method is implemented to navigate through the Trie, searching for a specific word in the structure, and subsequently displaying potential options for autocompletion. The method has a string as its parameter and a void return type. This method gets its string from the user’s input/query in the console being passed into the method. Like the "insertWord" method, the code starts by storing the root in a temporary variable. It then traverses through the query, checking if the current letter exists in the children of the current node.

Should a letter be absent, a message displays, signaling that the word doesn't exist in the data structure. If the letter does exist, it is appended to a string variable, "path," and the current node is updated to the node stored in the map at that letter. This process repeats until the entire word is traversed. Upon completion, the path is passed into a private helper method labeled "traverseSearch."

A screenshot of a computer program

Description automatically generated

The "traverseSearch" method, a recursive function with a void return type, takes in the path, a node pointer, and an unordered set of strings. In this context, it receives the path just traversed in the "searchTrie" method, the current node where the traversal paused, and the "validWords" unordered set. The base case checks if the node being passed in is null; if not, the method proceeds. It then checks if the node being passed in is marked as a complete word, looking at the "complete" member on the node. If so, the path passed in is inserted into the "validWords" set as an autocomplete word.

Following the above checks, the code iterates through the children of the current node, invoking "traverseSearch" recursively by passing the pointer to the current child node, the path with the child's key appended to it (a character), and the "validWords" set. This recursion continues until the node pointer being passed in is null, ending the call.

A computer code with text

Description automatically generated with medium confidence

After the completion of "traverseSearch," the code returns to the "searchTrie" method, which then iterates through the "validWords" unordered set, displaying each suggested autocompletion word in the console for the user.

A white background with black text

Description automatically generated

Part III

The "didYouMean" method was designed to offer suggestions when a word is not found in the data structure, replacing the message indicating that the word does not exist in the “searchTrie” call. The “didYouMean" method takes in a string and a node. In its initial call, the current path is passed in as the string, and the root is passed in as the node. This code operates similarly to the "searchTrie" method, treating the path as the word and iterating over it to find the corresponding node. Meaning at the end of the for loop the current node (“startNode”) will be where the path ends.

A screenshot of a computer program

Description automatically generated

A computer screen shot of a code

Description automatically generated

Once it lands on this node and stores it in its temporary node variable, the code then calls its private helper method, "didYouMeanHelper."

This method has a void return type and takes in a node, a string labeled "path," a string labeled "word," and an unordered set of strings. In this context, the current node, the current path, the query word, and the unordered set for suggested words are passed into this method. First, the code checks if the current node has a true "complete" member, indicating that the current path is a word, if the path is the same length as the word, and if the path doesn't already exist in the "suggestion" unordered set. If it passes this check, the path is inserted into the set.

A computer screen shot of a code

Description automatically generated

The code then iterates over the children of the current node, recursively calling the "didYouMeanHelper" method and passing that child in along with the current path appended by that child’s key. This continues until the base case is reached. This occurs if the node is null or the set is equal to three. Upon returning from the "didYouMeanHelper" method, the path "pops" or removes the last letter of the path and is then recursively passed into the "didYouMean" method along with the query word. This also continues until the suggestion set contains three words.

Upon returning to the "searchTrie" method, the suggestion set is looped over, and the three suggested words are displayed on the console.

A screenshot of a computer

Description automatically generated