Project 1

Adeel Ali

March 4, 2021

Description: Create a Trie data structure and traverse the Trie to simulate an autocomplete search query.

Logic Employed:

Creating the Trie

The idea was to create the Trie with the root being a node containing the empty character "" and the children nodes (which were stored in an array). Every time a word needed to be inserted, the word would recursively be broken down into the first letter which became its own node, then the first and second letters which became its own node, repeating the pattern up until the entire word became its own node. The entire word would be marked with a Boolean value of "true" to keep track for traversal.

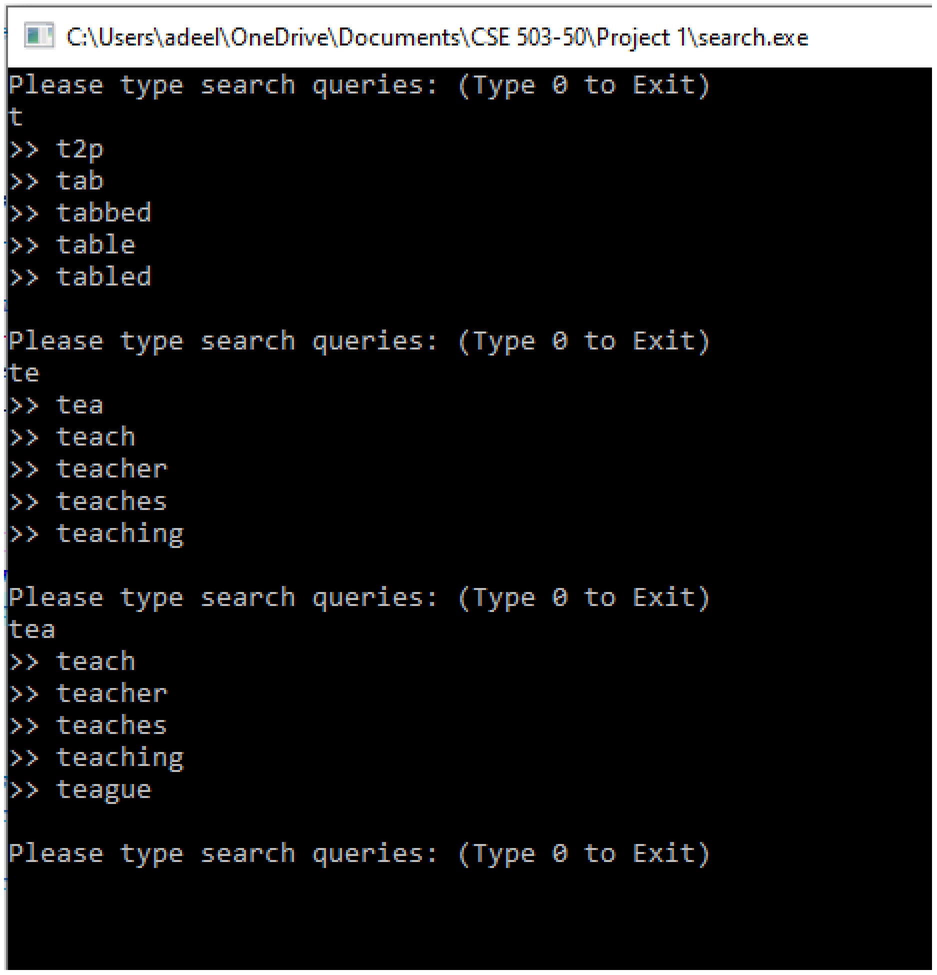
One crucial component was preserving the Trie structure after reaching the deepest level of recursion. As the program exited each layer of recursion, the parent node would be updated to have the previous recursively obtained root as its child. For example, if "te" is the parent node, then "tea" would be assigned as a child of "te". Preservation also involved making sure the Boolean value did not get overwritten if words like "tandem" were added after "tan" (since "tan" is an intermediary step for "tandem" and therefore would not be interpreted as an entire word on the second run). An if-statement was added to correct this.

Finally, there was an overall management component, which involved reading individual words from the input Dictionary.txt file, checking whether a child node already belonged to a parent node (returning the child node if so), increasing the size of the parent's children array, and using an iterator as part of the Trie class to properly index nodes.

Traversing the Trie

The idea was to use the completed Trie to perform a depth first search to find a user input string. If the user input string could not be found, no autocomplete entries could be displayed. Otherwise, once the string was found, a hybrid breadth/depth search would be performed where a node would be traversed depth-wise until an entire word was found (because of the Boolean "true" marking) otherwise it would traverse breadth-wise to find the next node to traverse depth-wise. The iterator was used again to limit the entries outputted to a pre-set amount (in this case 5).

Screenshots and Analysis



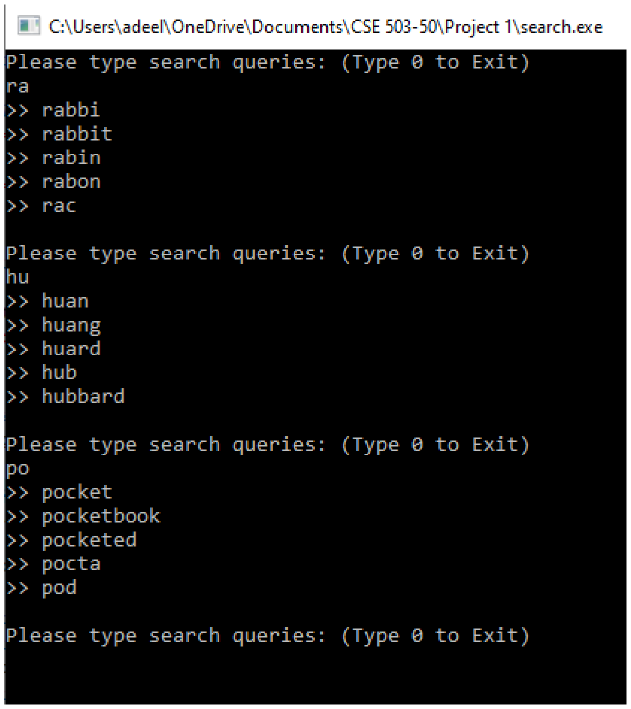
C:\Users\adeel\OneDrive\Documents\CSE 503-50\Project 1\search.exe

```
Please type search queries: (Type 0 to Exit)
t
>> t2p
>> tab
>> tabbed
>> table
>> tabled

Please type search queries: (Type 0 to Exit)
te
>> tea
>> teach
>> teacher
>> teaches
>> teaching

Please type search queries: (Type 0 to Exit)
tea
>> teach
>> teacher
>> teaches
>> teaching
>> teague

Please type search queries: (Type 0 to Exit)
```

The screenshot above shows 3 runs of the search.cpp program. In the first run, the input is taken to be “t” and the program first finds “t” within the Trie, then searches the children of “t” until it finds up to 5 actual words (marked by the Boolean “true”). Notice the first 5 words found are the first 5 words that start with “t” in the Dictionary.txt file.

The next run, the input is taken to be “te” and the program has to first find “t”, then “te” within the Trie. Again the process of searching the children of “te” is done similar to what was done before, but notice the first 5 words displayed are different because we traversed further in the depth of the tree. The options available for autocomplete can no longer be the original 5 words because the word must contain “te” as the first two letters.

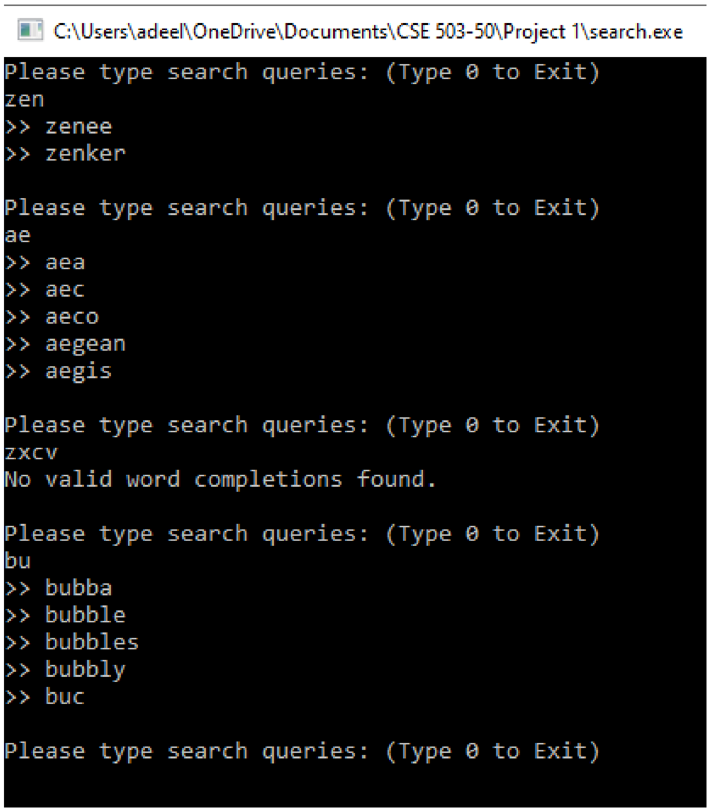
In the final run, the input is taken to be “tea”, so the program finds “t”, “te”, and “tea” then repeats the process of searching the children of “tea”. Since we are further narrowing our search, four of the five words remained the same from the previous run, and we expect the narrower our search, the less new options will be available every iteration.



C:\Users\adeel\OneDrive\Documents\CSE 503-50\Project 1\search.exe

```
Please type search queries: (Type 0 to Exit)
ra
>> rabbi
>> rabbit
>> rabin
>> rabon
>> rac

Please type search queries: (Type 0 to Exit)
hu
>> huan
>> huang
>> huard
>> hub
>> hubbard

Please type search queries: (Type 0 to Exit)
po
>> pocket
>> pocketbook
>> pocketed
>> pocta
>> pod

Please type search queries: (Type 0 to Exit)
```

This above screenshot's purpose is to illustrate that regardless of the input string, the Trie structure is preserved after every iteration of the traversal. Notice "r" appears later than "h" alphabetically, and "p" appears after "h" alphabetically, however, the program works the same for these three cases and displays 5 entries as expected.



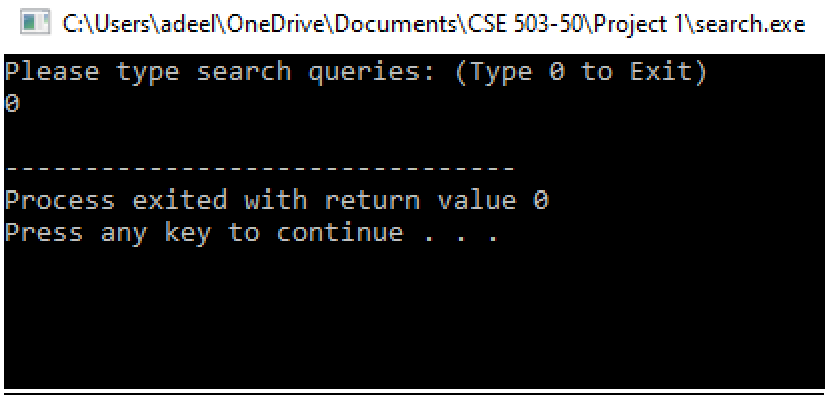
```
C:\Users\adeel\OneDrive\Documents\CSE 503-50\Project 1\search.exe
Please type search queries: (Type 0 to Exit)
zen
>> zenee
>> zenker

Please type search queries: (Type 0 to Exit)
ae
>> aea
>> aec
>> aeco
>> aegean
>> aegis

Please type search queries: (Type 0 to Exit)
zxcv
No valid word completions found.

Please type search queries: (Type 0 to Exit)
bu
>> bubba
>> bubble
>> bubbles
>> bubbly
>> buc

Please type search queries: (Type 0 to Exit)
```

This screenshot's purpose is to illustrate specific cases. The first run shows that if there are less word choices available after a user input string has gotten more specific, then even if the cutoff output entries is 5, the program will only output as many children words as there are available. That's why the only words that can be made from "zen" are "zenee" and "zenker" from the Dictionary.txt file.

The second run shows that the program functions normally even if there were only two output entries instead of five from the previous iteration.

The third run shows that if the user input string has gotten too specific and/or the word cannot exist with the combination of letters inputted, then the program will print the statement "No valid word completions found" to signify the first depth search was never able to identify the input string to be able to search the children of that input string.

The fourth run shows the program functions normally once again even if there weren't any entries found from the previous iteration.



C:\Users\adeel\OneDrive\Documents\CSE 503-50\Project 1\search.exe

```
Please type search queries: (Type 0 to Exit)
0

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

The loop breaks as expected when 0 is typed (shown in above screenshot)

Conclusion

There is enough evidence to suggest the Trie structure was implemented properly because of the speed of the program and the appropriate words we receive as autocomplete entries. The proper safeguards taken also ensure preservation of the Trie after every iteration and only by typing “0” does the loop end.