Day 1 - 12/3/2016

First hurdle - reading input from command line. Initial plan is to use scanner in.

How to parse string, specifically reading from multiple lines. Would I use next() or nextLine()?

I initially attempted to use both in some double for-loop. So the bigger for loop scans if there's a nextLine(), while the inner loop takes in every word from next().

But that's too taxing. So, I decided to have a loop that takes in the string per line. And then have a separate loop that parses each line to its own word, then to its own character.

Second hurdle - designing the Trie.

Would each node contain an array of 26 children? Or would it contain an arbitrary number of children? Would the nodes be added in alphabetical? Or would the first value added be the first child. What would be the run time differences?

Differences in run time:

(1) If I choose to add per character as a child like a queue, my code would be shorter (no need to convert to ascii, etc), but scanning to see if the other character to be added is already a child of the parent node would spend O(n) time.

(2) on the other hand, if I load each character based on its appropriate spot (i.e. a character c has nth index in the alphabet, it must be stored as the nth child of the parent node), so that when checking for duplicates, I could just call on children[*i*], where *i* is the index of the character to be added. Calling on an array[i] takes O(1) time. But, I'm not entirely sure the run time of converting a character to ascii.

Either way, I realized that choosing the first one would not affect the run time in the long run, since I'd later traverse through the entire Trie in the end. In essence, I would just be adding *n* to the search runtime of a Trie that will consume more than *n* runtime.

But my OCD-ness eventually made me opt for the second choice. Hence, checking for duplicates would no longer require a loop.

Third Hurdle - **How to Iterate**

Well, I first make sure that the characters are stored properly. After doing some trial runs, and applying depth first search to my Trie, it seems that the characters are stored in proper order. Now comes the challenge of outputting the words inputted along with their count.

Day 2 - 12/14/2016

Fourth Hurdle - How to print out the results

So far, doing DFS, I would output the values of each node and then enter one space when the program hits a node that's the end of the word. So example, if the input is HELLO HI, the output is going to be

HELLO - 1

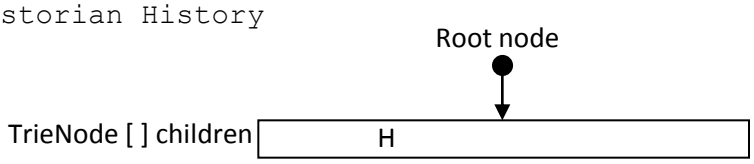
I - 1

Since the root node containing H has already been visited.

Therefore, I resorted to storing in each node the concatenated characters of all its ancestor nodes, and then planning to print out the value of that concatenated string in a node when that node is the end of the word. This would be the final form of my Trie structure.

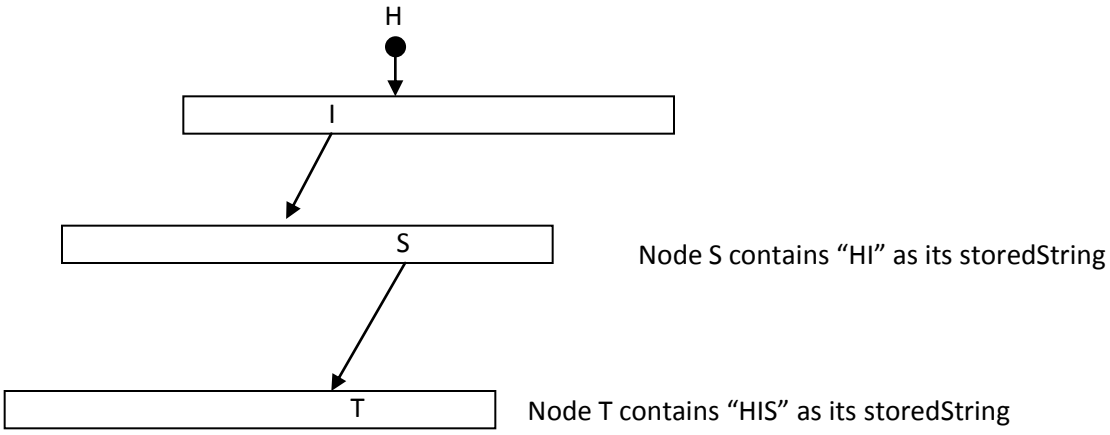
Input:

```
Historian History
```



Remarks: H is in the 7th index of TrieNode. Each character inserted into the children array is inserted in its corresponding index in the alphabet. I did this via converting characters into their ASCII value.

From H, we get the following children



.
.
.

The challenge of outputting the words in alphabetical order became dismissible, as doing a DFS inherently traverses the Trie in alphabetical order - due to the fact that I programmed my Trie to store values in the children node according to their alphabetical index.