CS 3345 Hon Data Structures Project 1

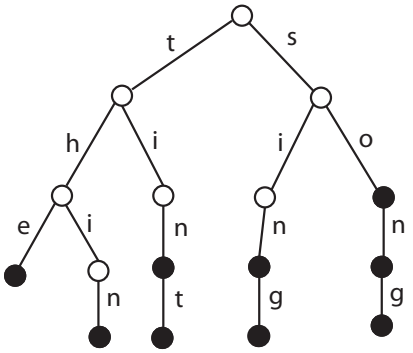
You are going to **write a class** to implement a Trie data structure and a program to test it. You will also **write a class for Node**, a tree node for the Trie.

The trie stores words of up to 20 characters in length, and is used as a spell-checker. To simplify matters, words will be restricted to strings of the 26 lower-case characters (excluding spaces and punctuation marks).

The member functions will be:

```
boolean insert(String s);      // returns false if s is already present, true otherwise
boolean isPresent(String s);   // returns true if s is present, false otherwise
boolean delete(String s);      // returns false if s is not present, true otherwise
int membership();              // returns the number of words in the data structure
void listAll();                // print all members of the Trie in alphabetical order
```

A `Trie` is a tree of degree 26. The following `Trie` contains the words: the, thin, tin, tint, sin, sing, son, song.



There is a boolean variable, `terminal` in each `Node`. In the black `Node`s that variable set to `true` to indicate the end of a word.

Here are the class variables for a Node:

```
class Node {
   boolean terminal;
   int outDegree;
   children Node[];
}
```

Each `Node` includes an array of 26 references to `Node`. When a `Node` is created, all the array elements will be initialized to null and the outDegree will be set to zero. No other variables may be added to the `Node` class.

In the above `Trie`, the root `Node` has only two children, corresponding to characters 's' and 't', or positions 19 and 20 in the array of references.

A search begins at the root `Node` and follows the links, one level for each character in the given word. To find the word "tin" we begin with array element 20 in the root `Node` and, if it is present, we check element 8 in the array of the second level `Node`. Finally, we check element 14 in the array of the third level `Node`. If that `Node` is marked as a terminal, we return "true".

The insert function begins with a search for the given word. If a null reference is found, a sequence of new `Node`s is created corresponding to the missing word ending.

To insert the word "soap" to the above Trie, two new `Node`s would be added, corresponding to 'a' and 'p'. The 'a' in this suffix would be referenced by the third level `Node` corresponding to the prefix "so".

Deletion also begins with a search for the given word. If found, there are two cases. Say we are deleting "sin" from the above `Trie`. Since the terminal character of the given word has children, deletion only requires flipping the boolean variable that indicates the end of a word.

If the word "thin" is deleted, the suffix "in" must be unlinked from the third level `Node`. To do this, when recursing out from the end of the given word, remove links to the characters of that word until a `Node` is found with degree greater than 1, or a terminal `Node` is found. The former case occurs when deleting the word "thin". The word "the" must remain. The latter case occurs when deleting the word "song." The word "so" must remain.

Your program MUST read a text file with name `TrieData.txt` that contains a sequence of one-line commands. In response to each command your program will output one or more lines of text to `System.out` (cout for C++ programmers).

Here are examples of the commands:

```
A soap    // Insert the word ‘‘soap’’
          // Print one of the lines ‘‘soap inserted’’ or ‘‘soap already exists’’
D sin     // Delete the word ‘‘sin’’
          // Print ‘one of the lines ‘sin deleted’’ or ‘‘sin not found’’
S fortune // Search for the word ‘‘fortune’’
          // Print one of the lines ‘‘fortune found’’ or ‘‘fortune not found’’
M         // Print the line ‘‘Membership is ####’’ where #### is the number of words in the Trie
T text    // Where ‘‘text’’ is a sequence of space-separated words, terminated by a newline character.
          // For each word in the sequence, do nothing if the word is found in your trie. For each word not found,
          // print a line, ‘‘Spelling mistake’’ followed by the offending word
L         // Print all the elements of the Trie in alphabetical order, one word per line
E         // The end of the input file
```

Here are sample input and output data files:

Sample Input	Sample Output
A ant	ant inserted
A goat	goat inserted
A frog	frog inserted
A art	art inserted
A goad	goad inserted
A antler	antler inserted
A go	go inserted
A from	from inserted
A part	part inserted
A past	past inserted
A arts	arts inserted
A part	part already exists
A frond	frond inserted
A fries	fries inserted
A text	text inserted
A message	message inserted
A mess	mess inserted
A mean	mean inserted
A goal	goal inserted
A interpretation	interpretation inserted
A coat	coat inserted
M	Membership is 20
D art	art deleted

```
D art                           art not found
D mess                          mess deleted
S art                           art not found
M                               Membership is 18
S antler                        antler found
S part                          part found
S text                          text found
S freis                         freis not found
T pert tacion anteler frog      Spelling mistake pert
                                Spelling mistake tacion
                                Spelling mistake anteler
L                               ant
E                               antler
                                arts
                                coat
                                fries
                                frog
                                frond
                                from
                                go
                                goad
                                goal
                                goat
                                interpretation
                                mean
                                message
                                part
                                past
                                text
```

RULES FOR PROGRAMMING AND SUBMISSION:

1. Write your program as one source file and do not use the"package" construct in your Java source.
2. Name your source file as $N_1N_2F_1F_2$P1.java where your given name begins with the characters N_1N_2 and your family name begins with the characters F_1F_2. For example my name is Ivor Page, so my source file will be called IVPAP1.java. Note that in all but the "java" extension, the characters are upper case.
3. Your program must not output any prompts. Its output must exactly match the format of the Sample Output above
4. Do not use any Java Collection Classes, except Strings and arrays. If in doubt, ask me.
5. Use good style and layout. Comment your code well.
6. Your program MUST read from `TrieData.txt` and write to the System.out (cout for C++ programs.
7. Submit your ONE source code file to the eLearning Assignment drop box for this assignment. Don't submit a compressed file.
8. There will be a 1% penalty for each minute of lateness. After 60 minutes of lateness, a grade of zero will be recorded.

Send me any questions/corrections (ivor@utdallas.edu).