COMP3712 ASSIGNMENT 1 DOCUMENTATION

Timothy Foong foon0020

Trie

readInDictionary

Implementation

The implementation of the readInDictionary function was done using the File Class and the Scanner Class. The File class is used to read in the target file, the File Class was picked as it is a simple way of importing files. The Scanner Class was used to separate the file into different lines, the Scanner Class was picked as it contained a simple method of splitting the File into its individual lines.

Testing

The testing for the readInDictionary function was done using the given test. The results of the test were compared to both the given answers as well as the given dictionary.

```
4 data/word-freq.expanded.trim.txt anomaly aba hoodie
Reading in trie...done

testing getNode
anomaly: TrieNode; isTerminal=true, data=33, #children=0
aba: TrieNode; isTerminal=false, data=null, #children=3
hoodie: null

testing get
anomaly: TrieNode; isTerminal=true, data=33, #children=0
aba: null
hoodie: null

Process finished with exit code 0
```

Compared to the given answer

```
4 data/word-freq.expanded.trim.txt anomaly aba hoodie <--
user input
Reading in trie...done

testing getNode
anomaly: TrieNode; isTerminal=true, data=33, #children=0
aba: TrieNode; isTerminal=false, data=null, #children=3
hoodie: null

testing get
anomaly: TrieNode; isTerminal=true, data=33, #children=0
aba: null
hoodie: null
```

getMostFrequentWordWithPrefix

Implementation

The implementation of getMostFrequentWordWithPrefix was done using most of the same code as getAlphabeticalListWithPrefix. It uses the same recursive code with an additional argument "freq" which is a list of integers. This function recursively calls recurseNodes2 which goes through all nodes with the given prefix and finds all words and their frequency. It adds the words to the "words" String List and their frequency to the "freq" Integer List. getMostFrequentWordWithPrefix then finds the position of the highest number in the freq list and uses that position to find the most frequently occurring word. It then returns the most frequent word. This method was chosen as I couldn't figure out a way to keep and update the most frequent word while searching through the prefix nodes. Thus, this solution was used instead which keeps all the words and their frequencies in separate lists but links them through their position in the lists.

Testing

The testing for the getMostFrequentWordWithPrefix function was done using the given test. The results of the test were compared to both the given answers as well as the given dictionary.

```
5 data/word-freq.expanded.trim.txt aba the bbb
Reading in trie...done
PREFIX = aba
Most Frequent Word is abandoned
PREFIX = the
Most Frequent Word is the
PREFIX = bbb
Most Frequent Word is bbb

Process finished with exit code 0
```

Compared to given answer

```
5 data/word-freq.expanded.trim.txt aba the bbb <-- user
input
Reading in trie...done
PREFIX = aba
Most Frequent Word is abandoned
PREFIX = the
Most Frequent Word is the
PREFIX = bbb
Most Frequent Word is bbb
```

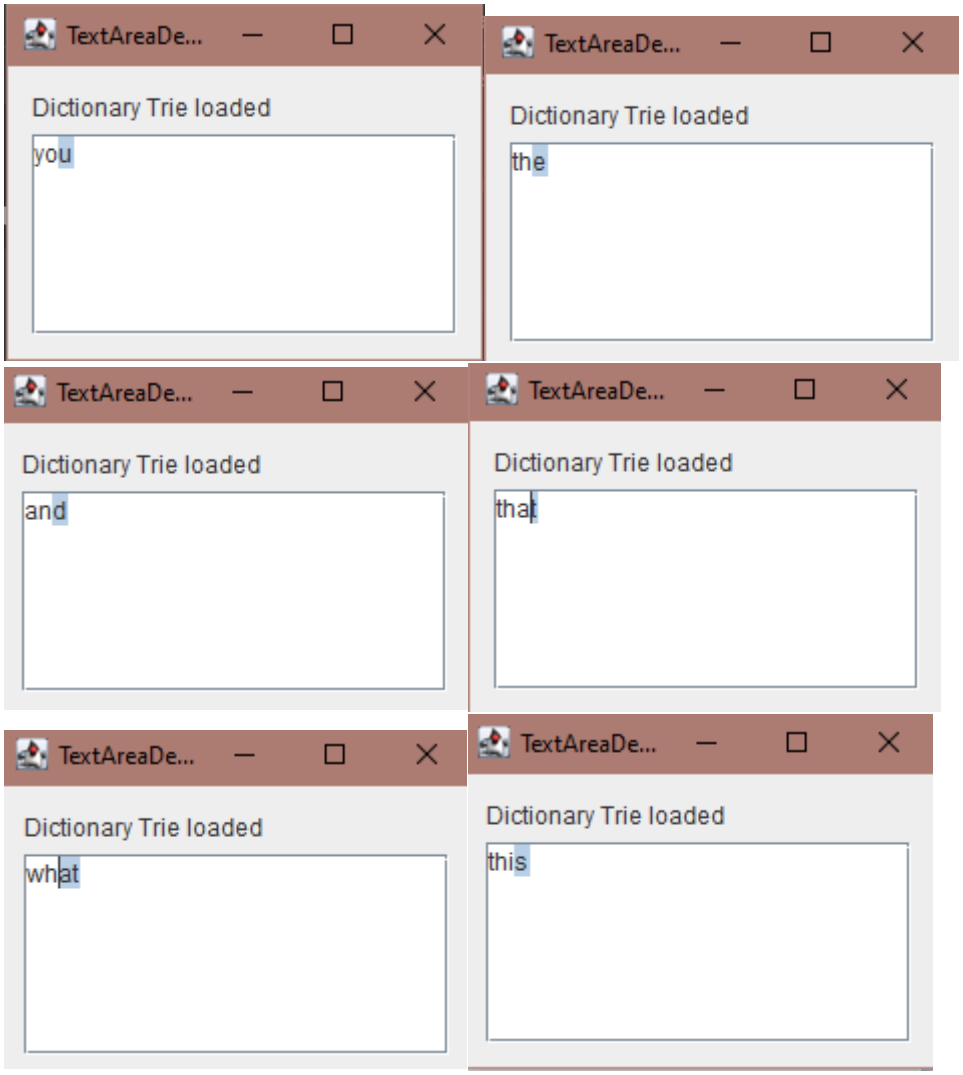
InsertUpdate

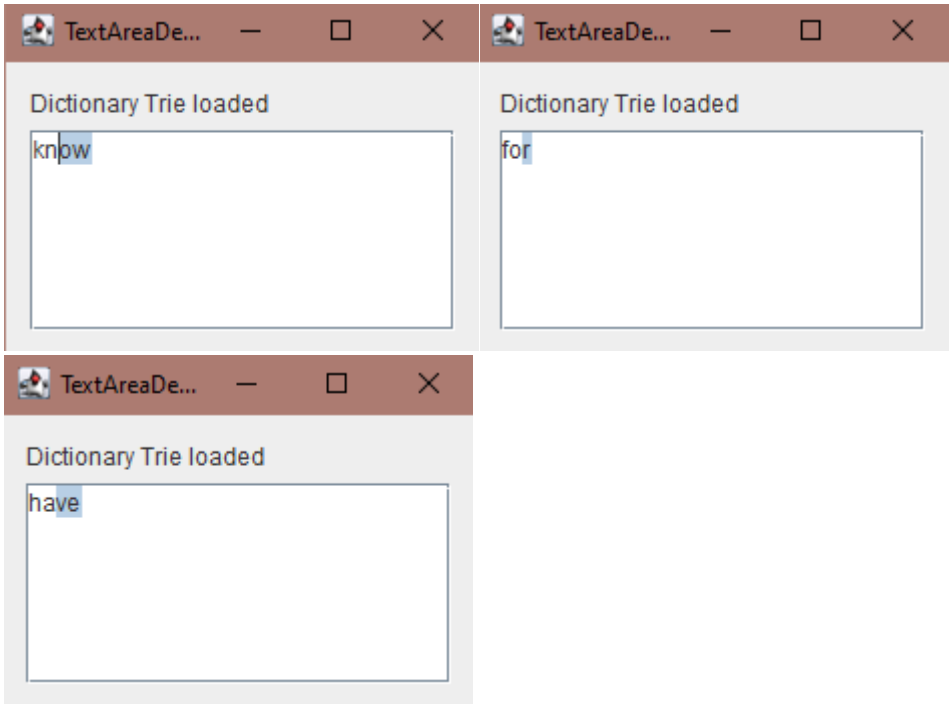
Implementation

With the implementation of both the getMostFrequentWordWithPrefix function and the readInDictionary function complete, the insertUpdate function is working as intended.

Testing

The textbox was tested using the previous function's testing parameters as well as the file "word-freq.expanded.trim.txt". The the top 20 words were entered into the textbox to see if they lined up with the frequency table. Some words had shared prefixes, so it was useful as it was used to verify that the most frequent word appeared rather than the word that was intended to be written.





SuffixTrie

Insert, get & readInFromFile

Implementation

The get function for the suffixTrie is almost identical to the implementation of the prefixTrie's getNode function. The only difference is that the suffixNodes now use a hashmap of String keys to SuffixTrieNode values. This change is because the text being inserted into the SuffixTrie now includes: em-dashes, apostrophes and other Unicode characters. Java is able to handle those characters as a char data type, but it does not like comparisons between char and Unicode characters, thus the decision was made to migrate the children map key from char to String. A hashmap was chosen for its fast get, put, remove and contains key functions as it is only O(1) for those functions as opposed to a TreeMap which is O(log(n)) for those functions. As these functions would called an exponential amount of times I believed that it was worth prioritising speed over keeping insertion order.

The insert function for the suffixTrie uses the same skeleton as the prefixTrie's insert function. The main difference is an additional for loop to insert all possible suffixes into the trie. It also takes in a sentence integer and a character integer to keep track of where the suffix appears in the original text.

The readInFromFile function uses most of the same code as the prefixTrie's ReadInDictionary function. The main differences are that when the function reads in the line it converts it to lower-case and then splits the line by every occurrence of "?", "!" or ".". It then checks if the line isn't blank before inserting it into the suffixTrie. There is an additional variable that tracks the number of sentences that have been processed into the SuffixTrie and inserts it along with the sentence to keep the location of each suffix. The readInFromFile function was originally implemented in

the minimal interpretation but it was found that em-dashes counted as three characters in the testing submission and apostrophes counted as two characters. In the maximal interpretation, those Unicode characters only count as one character.

Testing

The testing for the readInFromFile function was done by reading in the file Mississippi.txt and seeing if it output the correct string.

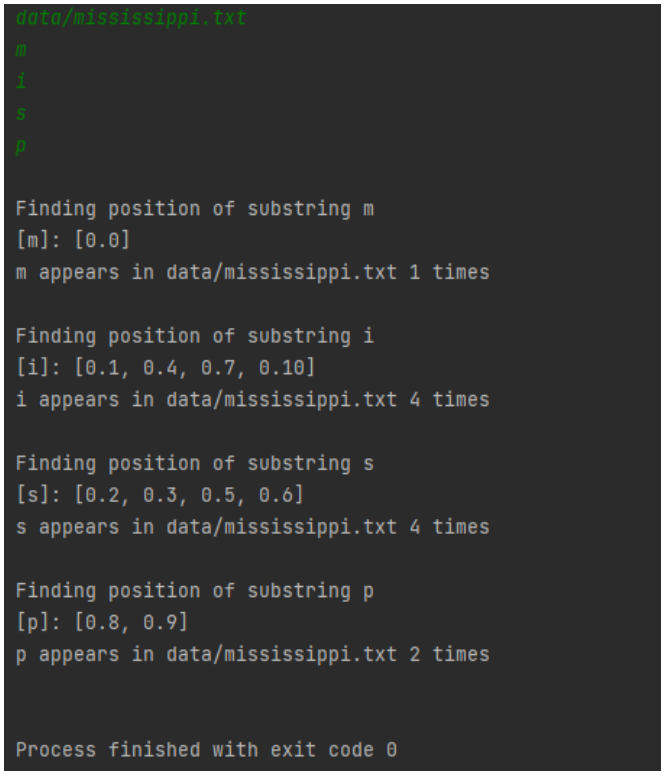
```
data/mississippi.txt
m

mississippi
Finding position of substring m
[m]: null


Process finished with exit code 0
```

It was then tested using Frank01.txt and Frank02.txt to see if it was able to successfully separate the content into the different sentences.

```
data/Frank01.txt
a

you will rejoice to hear that no disaster has accompanied the commencement of an enterprise which you have regarded with such evil forebodings
i arrived here yesterday, and my first task is to assure my dear sister of my welfare and increasing confidence in the success of my undertaking
Finding position of substring a
[a]: null


Process finished with exit code 0
```

```
data/Frank02.txt
a

you will rejoice to hear that no disaster has accompanied the commencement of an enterprise which you have regarded with such evil forebodings
i arrived here yesterday, and my first task is to assure my dear sister of my welfare and increasing confidence in the success of my undertaking
i am already far north of london, and as i walk in the streets of petersburgh, i feel a cold northern breeze play upon my cheeks, which braces my
do you understand this feeling
this breeze, which has travelled from the regions towards which i am advancing, gives me a foretaste of those icy climes
inspirited by this wind of promise, my daydreams become more fervent and vivid
i try in vain to be persuaded that the pole is the seat of frost and desolation; it ever presents itself to my imagination as the region of beaut
there, margaret, the sun is for ever visible, its broad disk just skirting the horizon and diffusing a perpetual splendour
there—for with your leave, my sister, i will put some trust in preceding navigators—there snow and frost are banished; and, sailing over a calm s
its productions and features may be without example, as the phenomena of the heavenly bodies undoubtedly are in those undiscovered solitudes
what may not be expected in a country of eternal light
i may there discover the wondrous power which attracts the needle and may regulate a thousand celestial observations that require only this voyag
i shall satiate my ardent curiosity with the sight of a part of the world never before visited, and may tread a land never before imprinted by th
these are my enticements, and they are sufficient to conquer all fear of danger or death and to induce me to commence this laborious voyage with
but supposing all these conjectures to be false, you cannot contest the inestimable benefit which i shall confer on all mankind, to the last gene
Finding position of substring a
[a]: null
```

The testing for the insert and get functions was done by reading in the different files and verifying the outputs. First the file Mississippi.txt was read in with the arguments m,i,s and p in order to find out if it could find all instances of each character.



FrankChap02.txt was then read in with the given parameters: and, the, ”, the”, onster, ngeuhhh and “ing? This”. “and” and “the” were used to see if the program could look for whole words, “onster” and “ngeuhhh” were used to see how it would react with strings that do not appear in the text, “, the” and “, as we believed,” was used to see if the program could handle both punctuation and whitespaces, “ing? this” was used to see if the sentences were split correctly, and “(July 31st)” was used to see how it would act with the brackets.

```
data/FrankChap02.txt
and
the
, the
onster
ngeuhhh
, as we believed,
ing? This
(July 31st)

Finding position of substring and
[and]: [2.13, 7.26, 7.86, 8.34, 8.153, 9.14, 11.69, 12.65, 12.147, 13.87, 14.95, 14.119, 14.172, 14.198, 15.16, 17.66,
and appears in data/FrankChap02.txt 368 times

Finding position of substring the
[the]: [6.58, 7.115, 8.51, 8.96, 10.38, 12.35, 12.47, 12.126, 13.0, 13.17, 13.75, 14.0, 14.84, 14.227, 14.248, 15.56, 1
the appears in data/FrankChap02.txt 690 times

Finding position of substring , the
[, the]: [13.15, 34.246, 41.52, 64.214, 75.4, 85.118, 101.70, 120.116, 128.65, 159.16, 177.13, 221.19, 221.71, 227.14,
, the appears in data/FrankChap02.txt 30 times

Finding position of substring onster
[onster]: null

Finding position of substring ngeuhhh
[ngeuhhh]: null

Finding position of substring , as we believed,
[, as we believed,]: [149.7]
, as we believed, appears in data/FrankChap02.txt 1 times

Finding position of substring ing? this
[ing? this]: null

Finding position of substring (july 31st)
[(july 31st)]: [141.12]
(july 31st) appears in data/FrankChap02.txt 1 times
```

FrankChap02.txt was read in with the intent of testing the programs functionality with the Unicode characters that appear in the text, such as the em-dash” —”, the apostrophe ” ’” and the Æ letter.

```
data/FrankChap02.txt
—
’
Æ

Finding position of substring —
[—]: [5.8, 14.5, 14.83, 21.226, 44.53, 60.25, 103.28, 103.91, 115.12, 117.50, 139.14,
— appears in data/FrankChap02.txt 32 times

Finding position of substring ’
[’]: [24.125, 26.150, 71.162, 88.251, 144.11, 182.25, 208.7, 209.48, 269.232, 276.54,
’ appears in data/FrankChap02.txt 18 times

Finding position of substring æ
[æ]: [182.109]
æ appears in data/FrankChap02.txt 1 times


Process finished with exit code 0
```

To verify all these results the FrankChap02.txt file was reformatted by splitting the text whenever there was a “.”, “!” or “?”. This formatting made it much easier to check the exact location of any patterns and verify if they were correct.

Extension

Wildcard suffix and prefix search

The wildcard character that has been chosen is the "*" character. This character can be used at the start of the word, to find all words that have that prefix, the end of the word, to find all words that have that suffix, and in the middle of the word, to find all words that start and end with a given suffix and prefix. Wildcard searches should be formatted as thus.

Prefix Search: "[chars]*"

Suffix Search: "*[chars]"

Body search: "[chars]*[chars]"

The actual implementation does not require quotation marks and the system can take any number of characters as either the suffix or prefix search, however it cannot do both a suffix and prefix search at the same time e.g. "*[chars]*".

Implementation

Wildcard suffix searches was an extension that I tried to implement. I tried to implement full wildcard searches but was only able to implement prefix and suffix wildcard searches for full words. The suffix wildcard was implemented by creating a suffix trie that contained all the full words in the text. The suffix tree reads in the file by separating the text into full words and inserting them into the suffix tree. All punctuation, except the apostrophe, is replaced by whitespace and then each non-whitespace string is individually inserted into the suffix tree. Each suffix leaf contains a hashset of the words that have that suffix and when the get function is called it returns the list of words. The node stores the words in a hashset to remove duplicate words that may be read from the file. The collection of words is then put into a list so it can be sorted, and the results are run through the original suffixTrie to find all the word's positions, the results are then printed into the console. The driver checks if the search term starts with a * character and runs the suffix search function.

Wildcard prefix searches were implemented through a prefix tree. The prefix tree reads in the file by separating the text into full words and inserting them into the prefix tree. All punctuation, except the apostrophe, is replaced by whitespace and then each non-whitespace string is individually inserted into the prefix tree. The getAlphabeticalListWithPrefix function from part 1 of the assignment is called in to get a list of the words that start with the given prefix.

After some thought I was able to figure out body wildcard searches, albeit it is a little limited in what it can search for. The body search can take in a prefix and a suffix separated by the wildcard operator, and it will return a list of words that start and end with the given prefix and suffix. This works by performing a prefix search with the given prefix and a suffix search with the given suffix. The program then eliminates any word that does not appear on both lists and returns the list of words.

Testing

The suffix and prefix wildcards were tested by the creation of the test.txt file. This file only contains each unique word in the Frank01.txt file. I manually checked the file and compared it to the output of the function.

```
data/Frank01.txt
a*
*a

Finding words starting with a
[accompanied]: [0.46]
accompanied appears in data/Frank01.txt 1 times
[an]: [0.52, 0.78, 1.26, 1.86]
an appears in data/Frank01.txt 4 times
[and]: [1.26, 1.86]
and appears in data/Frank01.txt 2 times
[arrived]: [1.2]
arrived appears in data/Frank01.txt 1 times
[assure]: [1.50]
assure appears in data/Frank01.txt 1 times
There are 5 words that start with a

Finding words ending in a
There are 0 words that end in a


Process finished with exit code 0
```

Unfortunately, an error that I was unable to fix was with words that are also substrings of other words, such as “an”. “an” only appears in text once as a full word, appears twice as part of “and” and once as part of “accompanied”. I was unable to figure out how to separate out those results and only display the times where it appeared as a full word.

The body wildcard search was tested in a similar way to the prefix and suffix search, with the results being compared to a list of unique words in the Frank02.txt file.

```
data/Frank02.txt
a*d

Finding words Starting with a and ending in d
[accompanied]: [0.46]
accompanied appears in data/Frank02.txt 1 times
[and]: [1.26, 1.86, 2.34, 2.153, 3.14, 5.69, 6.65, 6.147, 7.87,
and appears in data/Frank02.txt 20 times
[arrived]: [1.2]
arrived appears in data/Frank02.txt 1 times
There are 3 words that start with a and ending in d
```

This was then tested against the FrankChap02.txt file to see if it could fulfill a more specific search query.

```
data/FrankChap02.txt
en*and

Finding words Starting with en and ending in and
[england]: [2.9, 59.9, 114.9, 118.23, 120.109, 138.9]
england appears in data/FrankChap02.txt 6 times
There are 1 words that start with en and ending in and


Process finished with exit code 0
```

Frequency of words

Implementation

This is a function that returns the number of times a word appears in the text. It works by counting the length of the starIndexes arraylist. This was extended to work with the wildcards to show how many unique words either start with the prefix or end with the suffix. It works with the wildcard searches by simply counting the number of words each search returns.

Testing

The number of times a word appeared in text was verified by using a text editor program to search for the pattern and see how many times it appears.