What I do.

Thursday, December 24, 2020 11:05 PM

Week2:
Main goal: calculate number of sentences/words/Syllables in a given text.
What I did:
The main practice is around String.Split method. How to get the correct substrings to calculate sentences/words and Syllables. The first two were relatively easy. The last one has a few more requirements including checking if a lone e is at the end. I implemented some helper functions checkLoneEIfEnd, startWithCheck(to check if a word start with a vowel) and ifVowel(to check if is vowel), the basic syllable split regular experissions I use is: word.toLowerCase().split("[^aeiou what this line doing is generate a string array of strings splitted by the vowels. The only proble implemention is that if the original word not start with vowel, it will generate one more syllabl an empty first string. Pay attention to this. Then the other thing is to check if the ending lone e

The 2nd goal of this week is to implement the flesche score. which is very easy because the ha calculating numbers of sentences/words/Syllables has been done already. For this task, we just put these numbers into Flesche fomular. The only thing I need to notice is that if I want to get a value by integer/another integer, just simply add integer * 1.0 /another integer then the answe be double.

week 3:
Mainly 2 things:

first one is implementing the "EffiecientDocument", the difference from the BasicDocum implemented above is this time, read the numSentences/numWords/numSentences at t time by calling getText(), not a big deal.
Second thing is to write a benchmarking processing time analysis for efficient and basic Document. The built-in java method System.nanoTime is called by both before and after series of operations.

One thing to note: `estTime = (endTime - startTime) / 100000000.0;`

week 4:
Implementing my linkedList.
C2 w4 programming for myLinkedList and the JUnit Test(SO FRUSTRATING taking so much time

100000000.0;

week 4:

Implementing my linkedList.

C2 w4 programming for myLinkedList and the JUnit Test(SO FRUSTRATING taking so much time on that. Hopefully I can learn more form this session), also, the mylinkedList function are full of pitfalls, be careful next time to implement it.

LLNode<E>

MyLinkedList<E> extends AbstractList<E>

{

LLNode<E> head;

LLNode<E> tail;

int size;

```
public MyLinkedList()
{
// TODO: Implement this method
    head = new LLNode<E>(null);
    tail = new LLNode<E>(null);
    size = 0;
    head.next = tail;
    tail.prev = head;
}
```

Pay attention to those nodes' next and prev whenever you are trying to add or remove an element, you have to change their next/prev and probably their next' pre and pre's next

}

2nd part of this week's project is implementing Markov text processor which I have experience at Duke's Java Course, so this is not a big deal, what we do is for a given paragraph, for each word, we map this word to its following words. Then when we are generating Markov Text, Just randomly picking up words from its followers.

Week 5

1st part is C2 w5 Programming part 1 Spell Checking. which I have to implementing the Dictionary balanced binary tree and dictionary Linked List. Which is just using Java built-in data structures of TreeSet() and LinkedList along with their methods.

2nd part of this week is implementing DictionaryTrie, which is a little difficult than the

1st part, but the most difficult part has been implemented already: The TrieNode class define. So technically speaking, what I do for DictionaryTrie is based on what we already have. Pay attention to the TrieNode Implementation and its methods, maybe I can use it in the future.

Week 6

1st part of C2 w6 is implementing the spelling suggestion module. It is relatively easy by just doing distanceOne remove/substitution/delete one character from the given string, if not enough string found, then doing a second DistanceOne on all the words derived from the original string.

2nd part of C2 w6 is implementing the Word Path Game, The core of this assignment is the creation of the findPath method. findPath will function very similarly to the suggestions method you authored in the prior NearbyWords assignment. The fundamental difference between these two methods is that suggestions simply searched for nearby words, whereas findPath must find, and return, the path to a specific word. To be able to return the path, we need to have a way of reconstructing how we got to the target word - and we'll do this by creating a tree as we search.

Input: word1 which is the start word
Input: word2 which is the target word
Output: list of a path from word1 to word2 (or null)

Create a queue of WPTreeNodes to hold words to explore
Create a visited set to avoid looking at the same word repeatedly

Set the root to be a WPTreeNode containing word1
Add the initial word to visited
Add root to the queue

while the queue has elements and we have not yet found word2
  remove the node from the start of the queue and assign to curr
  get a list of real word neighbors (one mutation from curr's word)
  for each n in the list of neighbors
    if n is not visited
      add n as a child of curr
      add n to the visited set
      add the node for n to the back of the queue