* Design issues, what are the problems and how you solve them
* Data structures you use, the decision behind selecting them
* Algorithms you employ, again with a justification of your decision
* Particular emphasis should be placed on the running time of your algorithm
* Optimization issues: what could you do to further optimize your algorithm
* You need to specifically address the problem of scalability: would you implementation is efficient in the case of very large text collections?
* The report will also include results you obtained for the given sample queries, and for at least five
* additional queries of your choice
* Any other remarks about your design and implementation

Problems:

P01: Searching the correct query.

Problem:

When coding the first instance of the search engine, I included the case of lowercase and uppercase letters in the database, so with every node, it needs to have at least 62 child pointers (‘A’->’Z’, ‘a’->’z’, ‘0’->’9’), and in the trie, “indeed” and “Indeed” are two different branches. Which also means “indeed” will only returns “indeed” in search results and not “Indeed”. Indeed this is still somewhat correct, but not very practical since people rarely query searches grammatically correct.

Solution:

I solved this by normalizing all input data to lowercase with a toLower() function to efficiently store them in the trie. I also normalize the input query, so the trie search function will return the intended results every time. To write out the data’s original text, I first check if the trie search function returns true, then read in the whole paragraph again and compare each temporary normalized word with the keywords to know when to print it out highlighted.

P02: Highlighting the correct keywords.

Problem:

Now that both “indeed” and “Indeed” are printed out and highlighted. What about “Indeed,”? The punctuation now poses a minor problem in which “Indeed,” is highlighted instead of “Indeed”. This is because when inputting the trie, I used a function called getValidTxt(), which comprises of a getValidWord() function that shaves off unregistered characters and a toLower() function, to eliminate every character that is not in the lowercase alphabet or numbers from 0 to 9. And when I temporary normalized the words in a paragraph to compare with the keywords, I also use the same function. Therefore, to the computer, “Indeed” and “Indeed,” are the same words.

Solution:

To solve this, I make two more functions, getPrefix() and getSuffix() to store the unregistered characters in a word before temporary normalizing it for comparing. When printing the word out, I print the prefix first, then the word (itself highlighted), and finally the suffix.