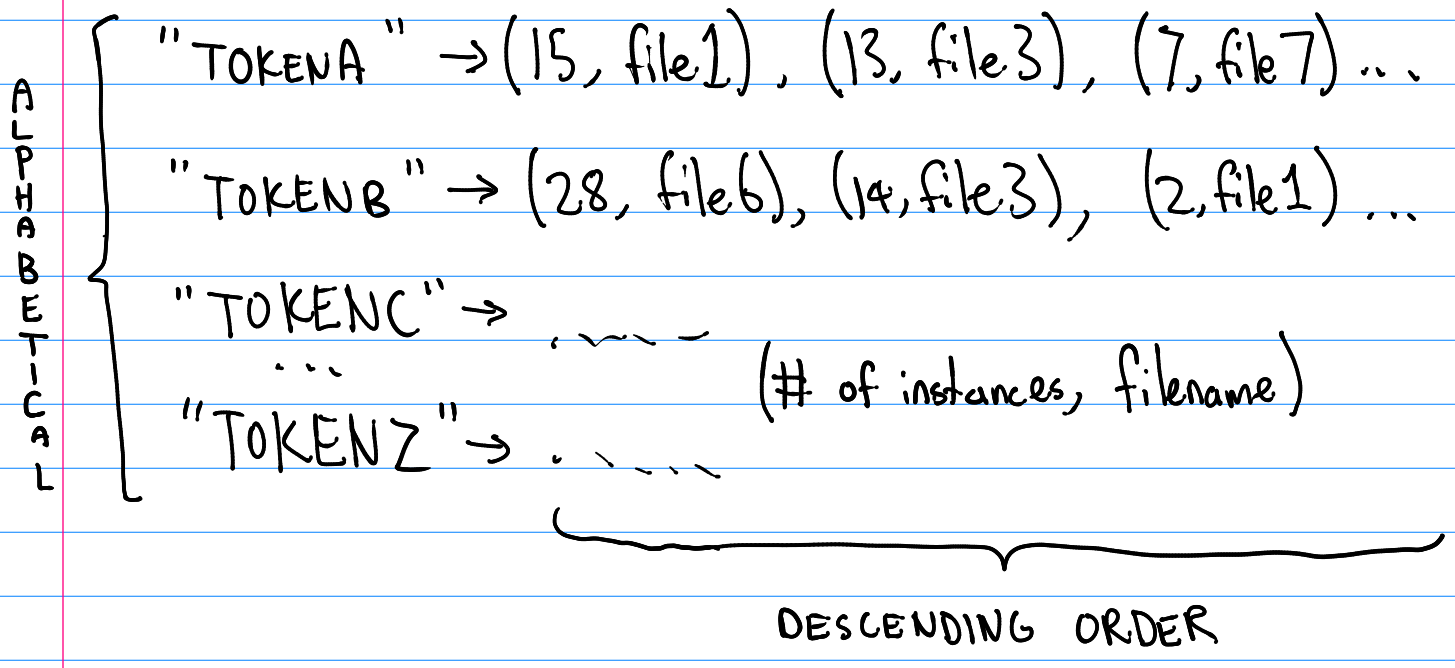
INDEXER NOTES

PROCESS:

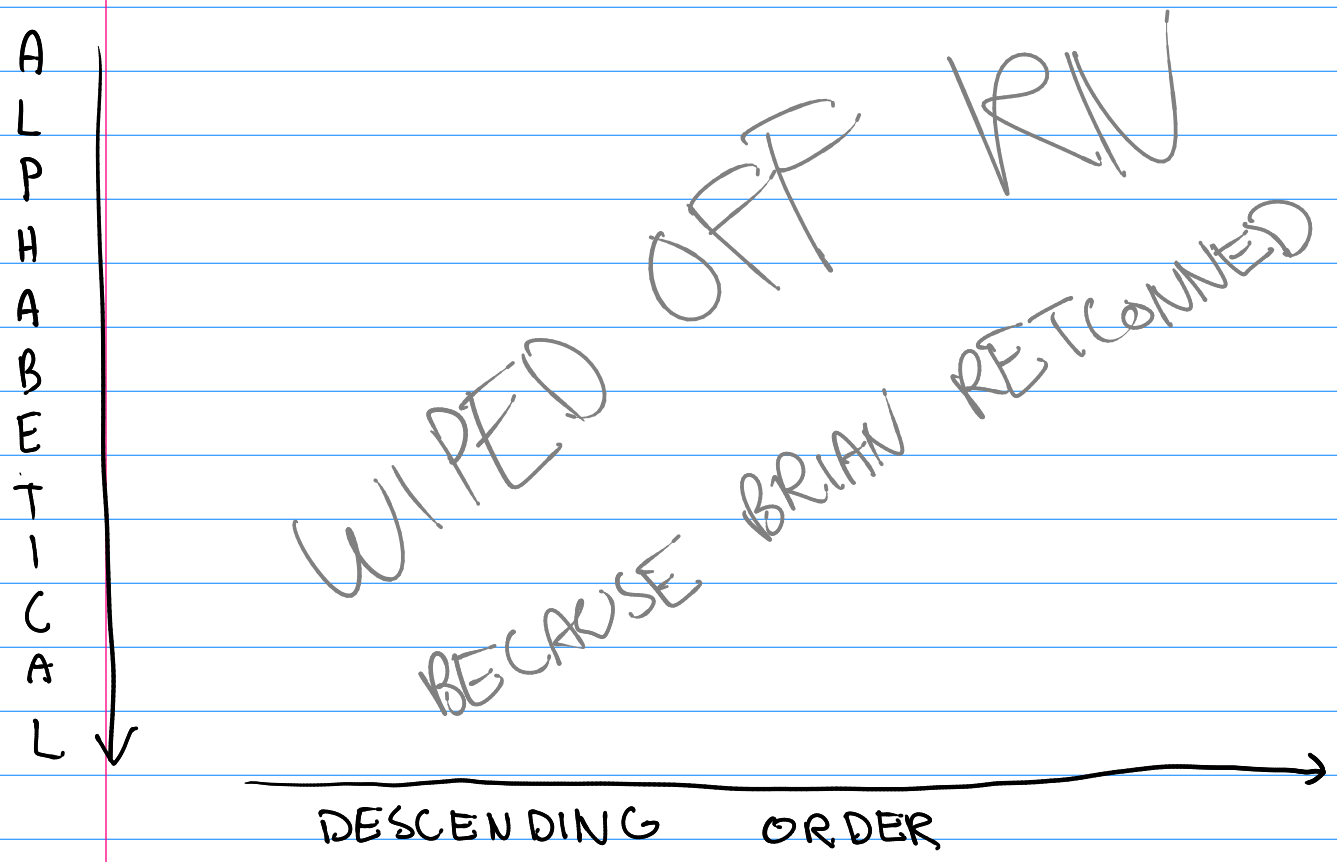
1. TAKE A FILENAME/DIRECTORY AS INPUT
2. TOKENIZE THE CONTENTS OF EACH FILE
3. COLLECT INFO ABOUT EACH TOKEN IN TERMS OF THE TOKEN, WHAT FILES IT APPEARS IN, AND HOW MANY TIMES IT APPEARS IN THAT FILE
4. STORE THIS DATA INTO A FILE

WE NEED
- A TOKENIZER!
- A DRIVER TO CALL THE TOKENIZER ON ALL FILES!
- A WAY TO WRITE MEMORY (THE TOKENIZER OUTPUT) TO FILE!

MEMORY STRUCTURE



FILE STORAGE STRUCTURE



ON MEMORY STRUCTURE:

↳ HASH TABLE OF LINKED LISTS?

→ needs varying number of keys

→ each token must be unique key

↳ what hash function would that even be

28 OCTOBER 2015

NEXT PAGE →

HOW A FILE FLOWS THROUGH OUR COMPONENTS

FILE/DIRECTORY INPUT

↓

TOKENIZER & SORTED LIST & TABLE

↳ each token is converted into a record

↳ each record is stored in a sorted list

↳ each sorted list is stored into a table, hashed by token

token → record → sorted list → table ⇄ FILE

STORING & UPDATING VALUES

→ When a token is first encountered, it is initialized in the hash table with a char* index & a sorted list of 1 record, listing the name of the file and its number of occurences

→ If a token exists in the hash table, only a new record is added to its entry sorted list, in its proper place

→ key idea is that a record is fully completed before being added to the list, NOT added to the list and then being updated

→ After a certain number of sorted lists is generated for same number of tokens, they will be stored to file in JSON format

→ the lists that were stored should then be freed, so that the entire table isn't stored in memory at one time, but the index should remain in the table

→ of course, a list may need to be updated after it's stored in a file

→ Do we need to parse this file?
it sounds messy