CompSci 251

Assignment 8

Due Nov. 26th, 2018

Introduction

This assignment will introduce you to some concepts of information retrieval in computer science. Information retrieval is the process of analyzing documents and answering user queries to find relevant documents related to the user's query.

The terms we will use are Documents, Tokens, Query, Positional Index, and Reversed Index.

Documents

Documents are files that contain information. Typically, documents come in many forms, but for our purposes they will be much simpler. For this assignment, documents will be in the form of a single long String where the document name occurs between the "<" ">" symbols at the start of the String. After the document name will be the file contents.

Example of two documents:

"<TestDoc.txt>This is an example of documents we will handle."

"<AnotherDoc.txt>The name of documents will vary. Their contents will vary too!"

Here, the first document name is "TestDoc.txt", and the second document name is "AnotherDoc.txt". The file contents occur after each document title.

Take note, when we process a document, we assign each document a docID that is unique to each document. To keep things simple, we will assign each document an integer value that increases by one with each newly added document. For example, if we processed the documents above, "TestDoc.txt" would be assigned 1, and "AnotherDoc.txt" would be assigned 2. If we added another document, that documents docID would be 3.

Tokens

Tokens are Strings that are individual words in the document. Sometimes I will refer to them as terms. In the example above:
"this", "is", "an", "example", "of", "documents", "we", "will", "handle" are all tokens in "TestDoc.txt".
"the", "name", "of", "documents", "will", "vary", "their", "contents", "will", "vary", "too" are all tokens in "AnotherDoc.txt".

When we split the string, each token must be converted to lower case and all specified punctuation must be removed before we store it. The punctuation we want to remove are any commas (,), any periods (.), any question marks (?), and any exclamation points (!). This will be done in the removePunctuation method of the Indexer class.

Query
A query is made up of tokens, or words, a user wishes to search for amongst a set of documents. It can consist of many words and many calculations are performed to find matching documents. For this assignment, you won't have to perform any complications, but you will need to find matching terms.

Positional Index
A positional index contains all locations of a token in a document.
For example, Token "documents" occurs at location 6 in "TestDoc.txt", and location 4 in "AnotherDoc.txt". When counting positions, we do not 0 index like arrays. The first word in a document occurs at position 1, the second word occurs at position 2, and so on.

Reversed Index
A reversed index is a very efficient way to answer a user's query and find all related documents to the query. You may think the easiest way to answer a query is to search all documents and look for matching tokens. For a very small set of documents that are very short, this way works fine. But if we have many documents that vary in sizes, answering the query will take very long.

Instead we will use a reversed index. A reversed index works by having each token point to all documents they occur in. This way, we simply find the token in the reversed index and can immediately locate what documents the term occurs in. No extensive searching through documents is required!

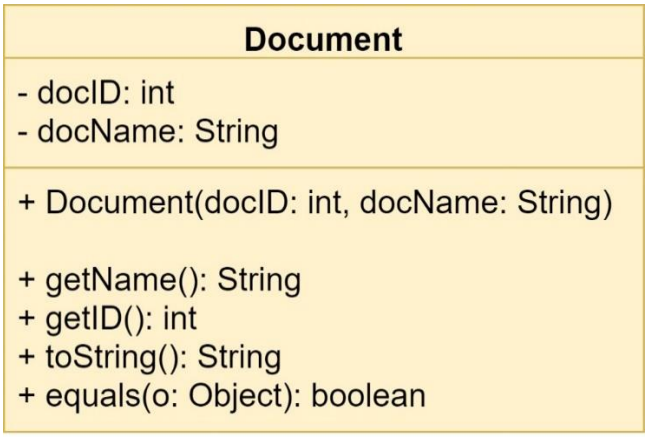
If we refer back to the example above,
Token "documents" would refer to "TestDoc.txt" and "AnotherDoc.txt", of if we used docID's, 1 and 2.
Token "example" would only refer to "TestDoc.txt", or if we used docID's, just 1.

Take note, only one Token is allowed in the reversed index no matter how many times it occurs. If you noticed in the example, "documents" occurs twice, but it will only occur in the reversed index once.

Program Specification

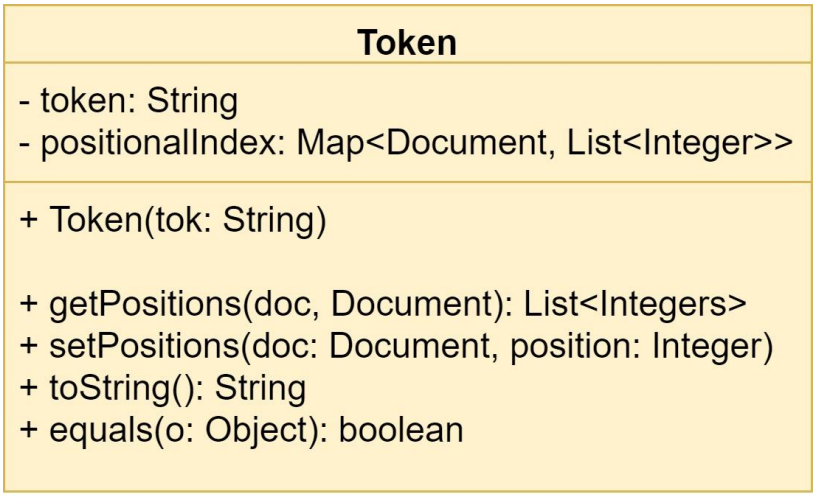
You will write three classes, Document, Token, and Indexer. These three classes will be used to parse many documents, store all the tokens where each token points to the documents they occur in, and then answer simply queries to find all documents related to the query. I have provided a driver. The driver contains all the "String" documents to read, it creates an Indexer object which reads/parses the documents, then it asks for a user to enter a query.

What you need to do



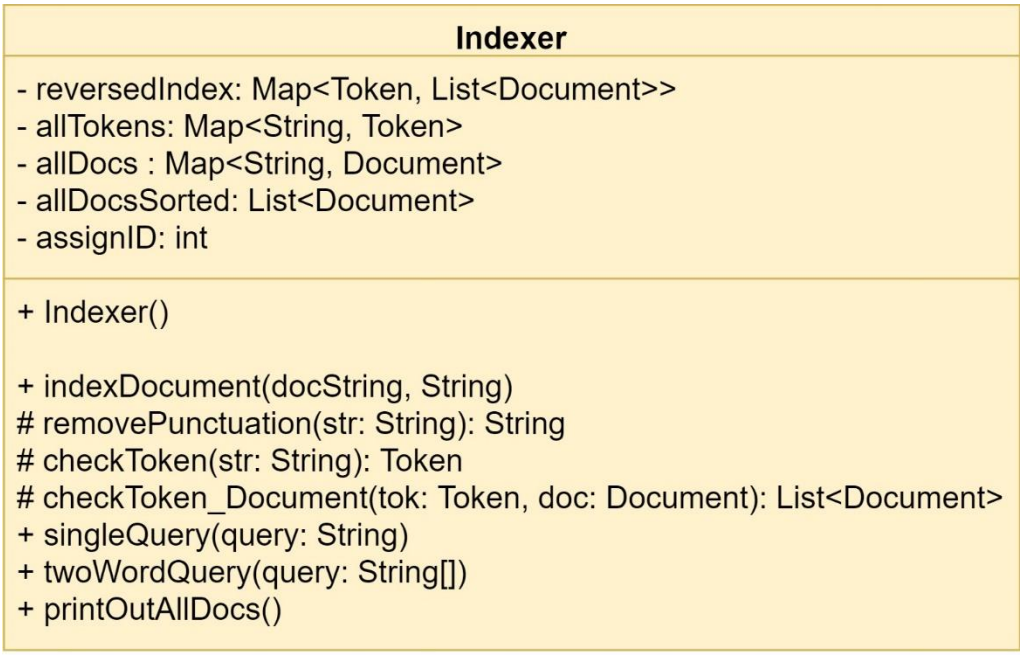
Document Class

- Follow the UML diagram above for the structure.
- When creating a document, you must pass in the actual documents name along with the docID to assign it. All of this is done in the Indexer class.
- A documents toString only returns the docID.
- Two documents are equal if all their instance variables are the same.
- A document object does not store the contents of the document, only the name of the document and doc ID assigned to it.



Token Class

- Follow the UML diagram above for structure.
- A Token object will be a wrapper object for a String.
- A Token will also store a Map that maps a document to a list of integers, the positional index. The integers will be the locations for that Tokens locations in that document.
- See the template I have provided for more explanations.



Indexer Class

- Follow the UML diagram above for structure.
- This class is responsible for reading String documents, parsing the documents into individual tokens, removing any punctuation, mapping all tokens to documents that contain them, and updating all positional indexes for each token.
- In the code I have placed many explanations and hints on what to do for each method.
- Of the two query methods, undergraduates only have to finish the singleQuery method.
- Graduate students, you guessed it, complete both singleQuery and twoWordQuery methods.

As stated above, I have provided a driver for you to perform queries on your classes. Below is some sample output.

```
The documents have been indexed.  If done correctly, all 10 documents should appear below,
with DocID and DocName shown correctly.
DocID: 1 , DocName: Milwaukee.txt
DocID: 2 , DocName: DeerHunting.txt
DocID: 3 , DocName: Madison.txt
DocID: 4 , DocName: Brewers.txt
DocID: 5 , DocName: CitizenKane.txt
DocID: 6 , DocName: Brats.txt
DocID: 7 , DocName: HappyDays.txt
DocID: 8 , DocName: Exports.txt
DocID: 9 , DocName: Lakes.txt
DocID: 10 , DocName: GreenBayPackers.txt

Please enter a query to search for or type 'quit': milwaukee

Single Query: milwaukee
Documents containing "milwaukee": [1, 4]
DocID: 1 , DocPositions =  [1]
DocID: 4 , DocPositions =  [2]

Please enter a query to search for or type 'quit': fonz

Single Query: fonz
Documents containing "fonz": [1, 7]
DocID: 1 , DocPositions =  [20]
DocID: 7 , DocPositions =  [23]

Please enter a query to search for or type 'quit': 1970's

Single Query: 1970's
Documents containing "1970's": [7]
DocID: 7 , DocPositions =  [10]

Please enter a query to search for or type 'quit': 595000

Single Query: 595000
Documents containing "595000": [1]
DocID: 1 , DocPositions =  [10]

Please enter a query to search for or type 'quit': 595,000

The query "595,000" was not found.

Please enter a query to search for or type 'quit': nate

The query "nate" was not found.
```

```
Please enter a query to search for or type 'quit': green

Single Query: green
Documents containing "green": [10]
DocID: 10 , DocPositions =  [2, 11]

Please enter a query to search for or type 'quit': bay

Single Query: bay
Documents containing "bay": [10]
DocID: 10 , DocPositions =  [3, 12]

Please enter a query to search for or type 'quit': green bay

Two word query "green bay" found at location [2] in Document [10]
Two word query "green bay" found at location [11] in Document [10]

Please enter a query to search for or type 'quit': miller park

Two word query "miller park" found at location [14] in Document [4]

Please enter a query to search for or type 'quit': is the

Two word query "is the" found at location [2] in Document [1]
Two word query "is the" found at location [2] in Document [3]
Two word query "is the" found at location [17] in Document [10]

Please enter a query to search for or type 'quit': no nate

The query "no nate" was not found.

Please enter a query to search for or type 'quit': more than two words won't work

Error in query entry.

Please enter a query to search for or type 'quit': quit

Goodbye
```

A short example

“<cats.txt>Cats eat mice. I like cats.”
“<dogs.txt>Dogs eat dog food. Dogs chase cats.”
“<fish.txt>Fish eat fish food. Fish do not like cats.”

Document Name	Document ID if processed in order above
cats.txt	1
dogs.txt	2
fish.txt	3

Each Token in the Reversed Index should be mapped to the documents they occur in.
Each Token should also keep track of their positional index for each Document.
Below is an example of all tokens referring to their documents and positions.
Note: Hash Map’s do not maintain alphabetical order like I did below.

Token	Documents Token occurs in (docID used)	Position in Document
cats	1	1,6
	2	7
	3	9
chase	2	6
do	3	6
dog	2	3
dogs	2	1, 5
eat	1	2
	2	2
fish	3	1, 3, 5
food	2	4
	3	4
i	1	4
like	1	5
	3	8
mice	1	3
not	3	7

As shown by the table above, if we wish to find all documents associated with a term, we simply look up the term to find all documents the term occurs in. With such a small example this doesn't truly show the power of this method. Imagine if we had over 20,000 documents, and each document contained over 10,000 words. With out a reversed index if we wanted to find a term, we would have to search every word of every document. This means worst case scenario the computer compares your term to over +200,000,000 different words. With the reversed index, we can avoid this problem. While searching becomes much more efficient, this does require us to perform a lot of pre-processing and memory storage before any user can start searching for information.