Overview

Complete the implementation of the `TextSearcher` class, found in `searcher.py`. When finished, please return to us **only** your updated `searcher.py`.

In skeleton, the class looks like this:

```
class TextSearcher:
    def __init__(self, file_name): # Implementation provided
    def initialize(file_contents): pass
    def search(query_word, num_context_words): return list()
```

TextSearcher implements a search interface to an underlying text file, consisting of the single method `search(str, int)`. This method takes in a single query word, and returns all substrings found in the file consisting of the query word and a few words of context, given by `num_context_words`, from either side of the query word.

For example, given an input file containing this text:

> I will here give a brief sketch of the progress of opinion on the Origin of Species. Until recently the great majority of naturalists believed that species were immutable productions, and had been separately created. This view has been ably maintained by many authors. Some few naturalists, on the other hand, have believed that species undergo modification, and that the existing forms of life are the descendants by true generation of pre existing forms.

Consider this code fragment:

```
searcher = TextSearcher("files/short_excerpt.txt")
results = searcher.search("naturalists", 3);
```

The `results` list will contain these two strings:

- “great majority of naturalists believed that species”
- “authors. Some few naturalists, on the other”

Note that each result string has 3 words to each side of the query word, “naturalists”, because `num_context_words` was specified as 3. Also note that the result strings should be returned exactly as they appear in the document, including original punctuation and capitalization.

Programming Environment

You have been provided with a skeletal version of the class `TextSearcher`. Comments in the source code provide additional information.

Unit tests are provided which test all of the requirements on the next page. Get as many of the tests to pass as you can.

A utility class is provided, `TextTokenizer`, which you may find useful for processing the file. See the comments in the source code for details.

You may use either Python 2.7 or 3

Performance notes

- It is safe to assume that the contents of the underlying file can be held in memory, i.e. it is not necessary to handle multi gigabyte files.
- You should assume that many searches will be done over the same document. A single instance of the `TextSearcher` class will be created once, and then the search method will be called many times. You should build an implementation that is efficient for this usage pattern (the search method should be as fast as possible).

Additional Requirements

Look these over before you start coding.

- A “word” consists of any sequence of the characters a-z, A-Z, 0-9, and apostrophe. Therefore, “animal’s” is a single word, as are “1844” and “xxxxx10x”. Any non-word character is regarded as punctuation and will be ignored by the search (but included in the results)
- Search should be case-insensitive. In the above example, the query words “species”, “SPECIES”, and “SpEcIeS” should all return the same three hits.
- If the query word occurs too close to the beginning or end of the file to permit the requested number of context words, start the result string at the beginning or end of the file as appropriate. So, `search("forms", 2)` will include “pre existing forms.” as one of its hits.
- Strings should be returned in document order.
- Overlapping hits require no special treatment; each should be returned as a separate hit.
- The search method should never return null. If no hits are found it should return an empty array.