SMALL SEARCH ENGINE

Software Assignment:
PART I: Create Index File

Introduction to Python Programming Course
Winter Semester 2011/2012

Submitted by:
Sevtap Fernengel-Sienerth

Submitted to:
Annemarie Friedrich
Stefan Thater

16. January 2012, Saarbrücken

Introduction

A search engine is a piece of software that searches documents for relevant keywords given by the user and returns a list of documents which contain those keywords.

The necessity for using a search engine is motivated by the following reasons:

1. The consumer of the information does not know if and where relevant information exists.
2. There are simply too many information sources for a user to search through to find the information.

A search engine solves the first type of problem by systematically visiting every place possible where information is located and recording the place and the type of information found at that place. This task is called **indexing**.

Once the search engine knows where the relevant information is located, the second task is to present this information in the order of relevance, the most relevant one being on top and the least relevant one being at the bottom of the list. This sorting is called **ranking**.

Assignment

In this software assignment, our aim is not to implement a full-fledged search engine but to focus on implementing two key components of a search engine. Namely:

1. Index a document collection.
2. Accept the search queries given with a special format, using boolean operators (AND, OR, NOT). Find the documents which contain, or do not contain, the keywords in them, depending on the search query. Return the list of document identification numbers of these documents.

Environment

Python version 3.2

Usage

In order to create the index file for the nyt199501 corpus file, simply type the following at the terminal:

python3.2 indexGenerator.py

This will display the following messages:

```
$ python3.2 indexGenerator.py
Processing the nyt199501 file to create the index.txt file...
Finished creating the index file
```

System Diagram

The static view of the system can be captured by the following diagram:

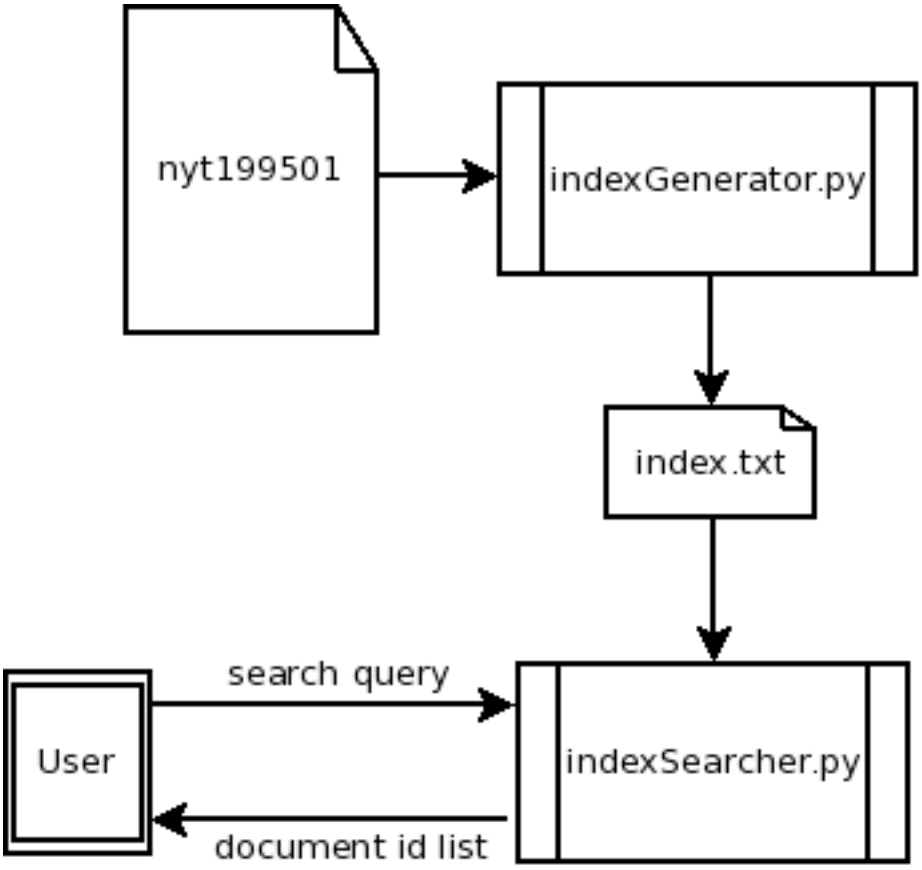

Fig 1: System Overview

In this first part of the project, only the creation of the index file is covered. Once the second part is completed this will also include the interaction with the user as shown in the diagram above.

Class Diagram

For the first part of the project, there are no class diagrams. Since the creation of the index file is a well defined top to bottom procedure, the functionality is implemented using procedural programming techniques, namely the program starts executing from the beginning of the source code and executes line by line until it reaches the end of the source code.

In the second part, there will be class diagrams, because it is easier to capture the dialog between the user and the system, where there are different kind of procedures that need to be accomplished with different constraints to reply to a user query.

Textual Description

The indexGenerator.py code consists of six blocks of code:

1. In the first part, the code reads the nyt1995 file and generates the word counts for each document in a dictionary type of object. All the dictionary objects are stored in a higher level dictionary.
2. In the second part, the code generates the data needed for creating the inverse document frequencies, this data includes which words are used in the document collection and in how many different documents a word occurred in.
3. In the third part, the code generates the actual idf measures for the words that occurred in the document collection using the math formula given.
4. In the fourth part, the code generates the tf-idf scores for each document and stores the results in an another dictionary whose values are accessible by document ids
5. In the fifth part, the code selects only the top ten words with the highest tf-idf scores for a given document and the results are stored in a dictionary again whose values are accessible through document ids
6. In the sixth, the last part, the code writes the document id, tf-idf score for a word, and the word for that document, in a file. This operation is done for all the documents in the document collection.

Improvement Suggestions

The code for the first part implements the basic functionality described in the project assignment document.

However, I can foresee that certain aspects of the code can be further improved given enough time and need for the improved functionality. Some of those improvement suggestions are:

1. A better tokenizer; at the moment, the code uses the string class split and strip functions to get the words out of the document. This is a very naive approach. There could be better ways of implementing a tokenizer which uses linguistic knowledge. Although such a naive approach seems to work with the English language but it is certain that it is going to fail with languages which do not use white space as a word separator.

2. A better organization of the code; as it is now, I saw no need to organize the code into functions or classes, because I can not motivate the idea that I might reuse this code beyond this project.

3. Passing file names as a parameter; again for this simple implementation, I hardcoded the input and the output file names in the code, since the code will be executed only once for a corpus file. However, if this was a batch job, where this code needed to be executed with different files all the time, then it would be wiser to parameterize the file names.

References

Python documentation: http://docs.python.org/py3k/library/