CPS 842  
Info Retrieval and Web Search

Assignment 1

October 2, 2019

Shah Moiz Alam - 500578704

Josh Vandenbor - 500695841

**What the program does?**

The assignment was coded in Python. The data structure used in the assignment to create a dictionary was the *Python Dictionary*. Python has a dictionary which is an unordered collection of values similar to a map. It has a key and value association where the keys are unique, and the values correspond to one key. Similarly, the postingsList was created using the same technique and data structure. The keys will be unique terms and the values being the document ID, term frequency and position of all occurrences for every document.

When you run **invert.py** it will create a dictionary file and postingsList file for the words in CACM.all. The dictionary file(dictionary.txt) will consist of terms and their respective document frequency and all the terms will be sorted in alphabetical order. Similarly, the postingsList(postingsList.txt) will include the postings for all the documents that the term occurs in. It will display the document ID, term frequency and the position of all the occurrences, sorted in alphabetical order.

The **test.py** program works by taking data from **invert.py**  then excepting a users input to search for a term, and printing the corresponding information to that term. The time is also recorded both for each query and also for the average of all the queries. Typing “ZZEND” will stop the program and display the average time. Notworthy: if stemming is activated, the users input will also be stemmed to allow convenient searches for say “computer” if the user doesn’t know the stemmed version of the word “comput”.

**Our Algorithm Explained:**

The program starts off by reading in the CACM.all file and storing it in a variable. Each document within the CACM file is then split by the document ID, “.I”. The document is then parsed through to isolate and store the **title**, **abstract** and document **ID** In a custom object defined as a “**DocumentStruct**”. After this, we remove punctuation marks from the document and use Porters Algorithm to provide the stemming functionality if chosen. We also remove the stopwords if the option was chosen, (Stopwords are read from the file, *Stopwords.txt* provided by instructor). Once we have stored the important fields from the document, we append the title and abstract to create a new variable and then create the dictionary and postings list.

The **postingList** is created by looping through the documents and keeping track of each term in that document and the amount of times it appears in all documents. This variable is stored in each documents DocumentStruct. We also keep track of the positions of each word as it appears in the combined Title and Abstract and add it to the value associated to that key of a dictionary called positions in the document object. This successfully creates functioning dictionaries for both term frequency, and positions. We initially populate postinglist with a key for each unique word in all the documents. Then we assign document objects corresponding to the key in each posting, to the posts. When we print the **postingList.txt** file, we go through each posting, and print out the doc ID, frequency, and positions from the objects assigned to the posting.

Ex. [ ID:3176 | Freq:1 | Pos: 44 ]

For the **dictionary** we go through all the text from the combined title and abstracts of each document, and count the number of occurrences for each unique term. This is stored in a dictionary with a key for each term, and its associated frequency in all the text. This is then printed to the **dictionary.txt** file.

**How to run the program:**

To do the index construction run **invert.py**, by typing “python invert.py”. The program will as the user if they would like to omit StopWords from the postingsList and Dictionary and the user can choose to type “y” or “n” to omit or keep the stopwords, respectively. The user will then be asked if they would prefer the output to be stemmed or net. The user can choose to type “y” or “n” to stem or not stem, respectively, the output. After this the program will begin to create the dictionary and postingsList file. The two files once created, will be found in the same folder as the rest of the files.

To test the inverting program, run **test.py,** by typing “python test.py”. The program will ask the user to input a term. If this term exists in the documents from the collection, it will output the document frequency, all the documents that contain the term and for each term they will include; documentID, the title, term frequency, all the positions the term occurs in, a summary of the document highlighting the first occurrence of the term with 10 terms in the context and the time taken to retrieve this information. The program will repeatedly ask the user for a new term until “**ZZEND”** is typed in as input. Once this is done, the program will terminate and display the average amount of time taken to retrieve the information.

**Sample Images (Figure 1 – Figure 3)**



Figure 1: Dictionary with no Stemming and with StopWords

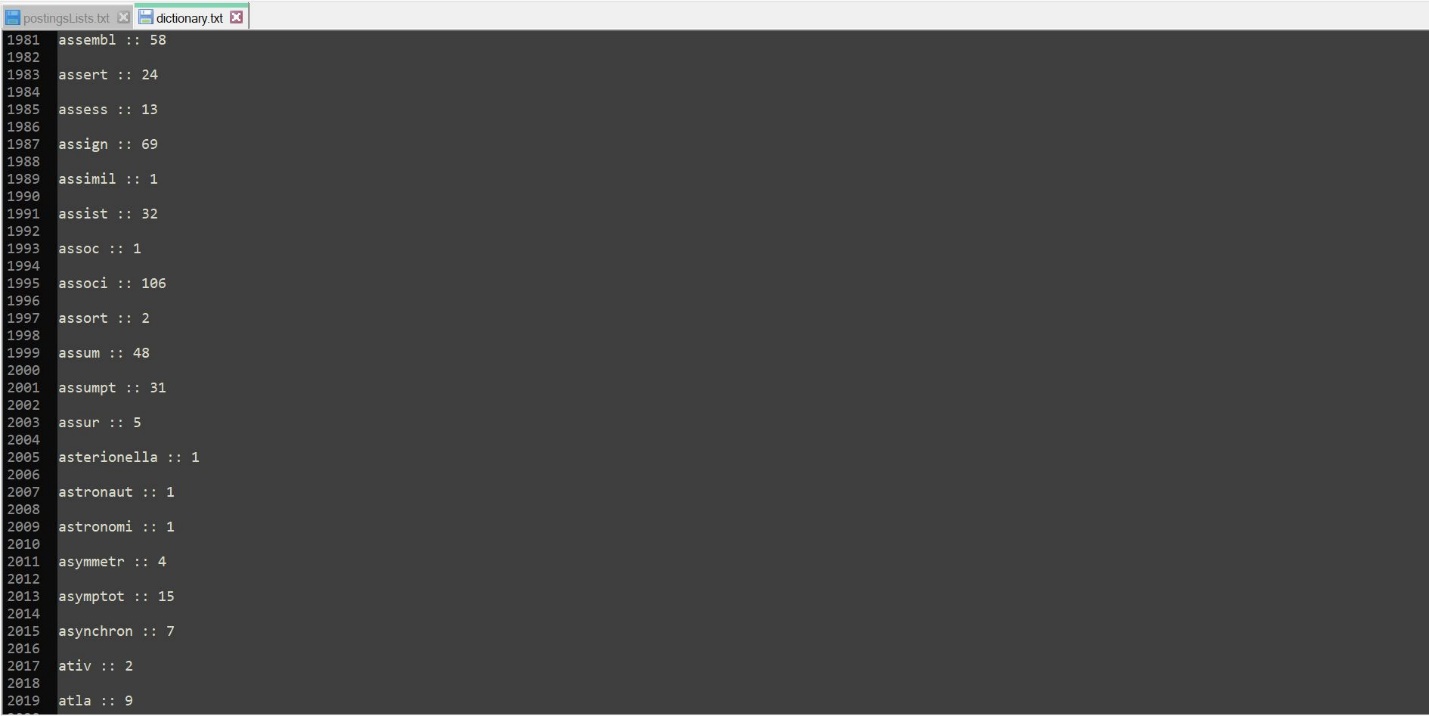


Figure 2: Dictionary with Stemming and no StopWords

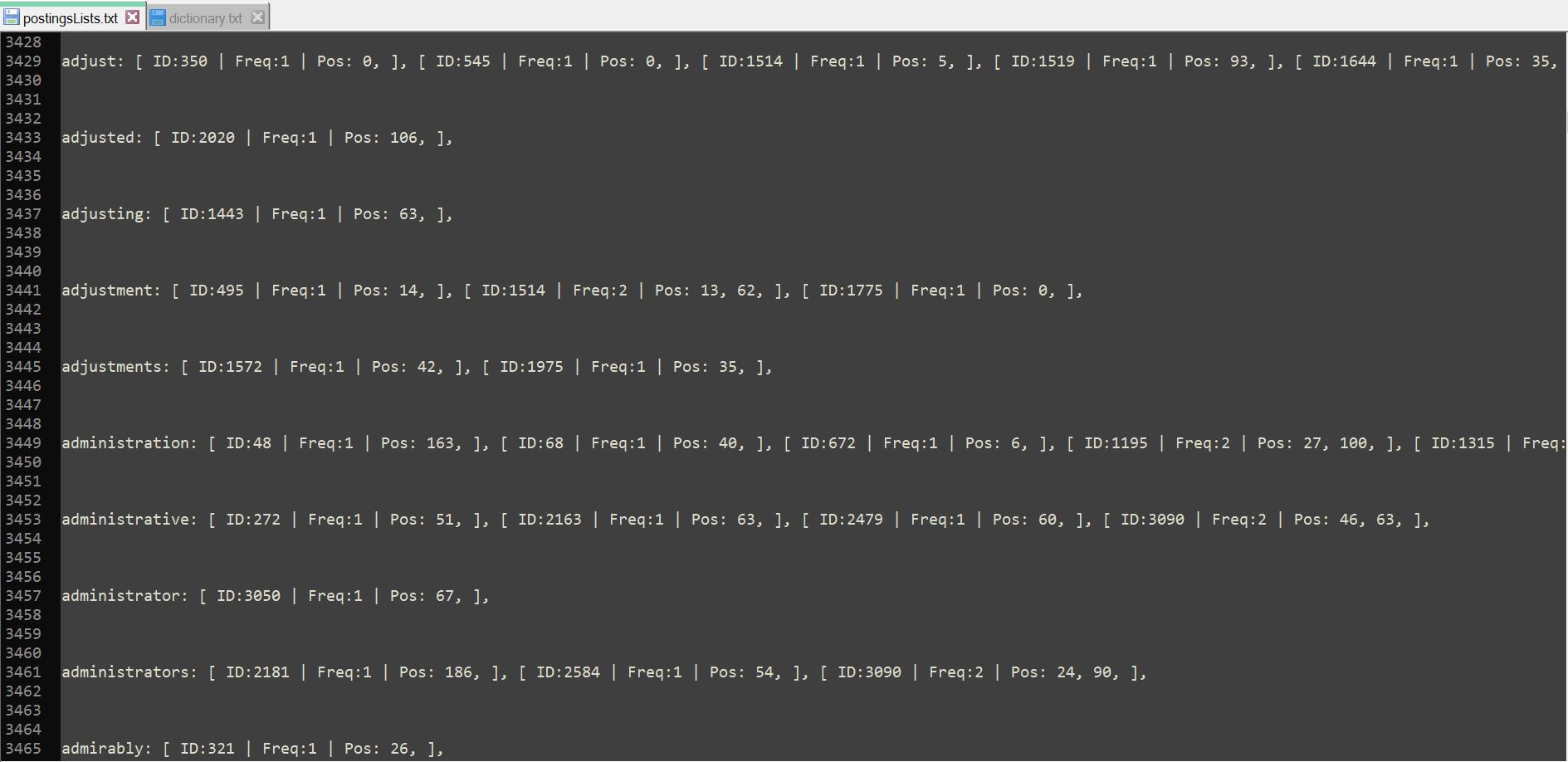


Figure 3: PostingsList with no Stemming and with StopWords