FACOM- UFU
Teacher: Wendel Melo

First work on Information Organization and Retrieval 2023-1

Description

Objective:

1. Implementation of an inverted index generator ;
2. Implementation of the Boolean IR model that should use the inverted index generated to answer a query.

Only **one** program developed in Python that performs the described task should be submitted. The work must be done individually and the code generated must be submitted using an appropriate form on the Teams platform.

Important note: if copying or any kind of cheating between different students is detected, all students will be penalized with a mark of zero. So think carefully before asking to copy your classmate's work, as they could be punished too!

It is mandatory to use the nltk package to extract radicals from vocabulary terms and obtain a valid list of *stopwords*. The details of inverted index generation and query processing are described below. **It is important to read carefully and follow all the details of the specification, otherwise you will lose points in your grade!**

The document base

The document database consists of an arbitrary set of plain text files. Assume that in these text files, words are separated by one or more of the following characters: blank space ( ), period (.), ellipsis (...) comma (,), exclamation mark (!), question mark (?) or enter (\n). Your program should treat uppercase and lowercase characters as equivalent.

The *stopwords*

Stopwords are terms that, taken in isolation, do not contribute to understanding the meaning of a document. Therefore, *stopwords* should **not** be taken into account when generating the inverted index! Your program must take into account the list of stopwords for the Portuguese language available in the nltk package, as seen in class (see the examples from the practical nltk class).

Consultations

The queries to be answered by the system are made up of terms connected by the operators & (*AND)*, | (*OR*) and ! (*NOT*). Thus, the system must be able to answer queries such as the following:

- dog & catRead : dog *AND* cat
- oven | stove & kitchenRead : oven *OR* stove *AND* kitchen
- airplane | car & !carriage | shipRead : airplane *OR* car *AND NOT* carriage *OR* ship

To make your life a little less difficult, assume beforehand that queries cannot contain parentheses and that the ! (*NOT*) operator takes precedence over & (*AND*), which in turn takes precedence over | (*OR)*. In other words, you can assume that the query is always received already in disjunctive normal form. Therefore, the query:

airplane | car & !cart | ship can be

answered through three types of documents:

1. Documents containing the term "airplane";
2. Documents that contain the term "car" but do not contain the term "cart";
3. Documents containing the term "ship".

Entering the program

Your program should receive two arguments as input **from the command line**. The first argument specifies the path of a text file that contains the paths of all the files that make up the base, each on one line. The second argument specifies the path of a text file that contains a query to be answered.

Example: Let's assume that our base is made up of files *a.txt*, *b.txt* and *c.txt*. Let's also assume that our program is called *model_boolean.py*. We would then call our program from the command line by doing:

> python model_boolean.py base.txt query.txt

where the *base.txt* file contains the paths to the files that make up the document database (note that the *base.txt file* can contain an arbitrary number of paths to the files that make up the document database, not necessarily 3), as follows:

a.txt
b.txt
c.txt

base.txt

The *query.txt* file contains a query to be answered by the IR system, written on a single line in the format specified above.

```
home & love | home & !mora
```
query.txt

Let's assume that our list of *stopwords*, obtained from the nltk package, contains only the following terms:

```
not
in
a
```
list of stopwords

Leaving the program

The program should generate two output files, with names and contents exactly as follows:

- *indice.txt* : file containing the inverted index generated from the documents in the database
- *answer.txt* : file with the names of the documents that answer the user's query

The file *indice.txt*:

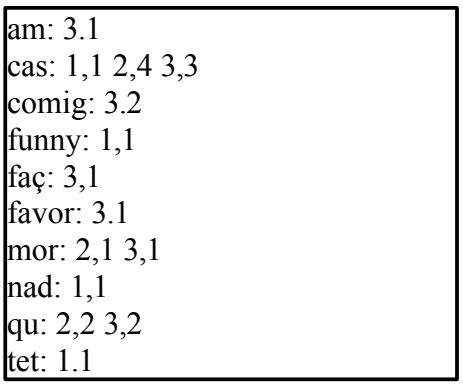
The program must generate a file called *indice.txt*, which contains the inverted index generated from the documents in the database, in the same way as in Job 1.

For each of these radicals in the index, it is necessary to indicate the number of the file in which it appears, and the number of times it appears in the file. The files are numbered according to the order in which they appear in the file indicating the documents in the base, which for our example has been named *base.txt*. Thus, file *a.txt* is file 1, file *b.txt* is file 2 and, finally, file *c.txt is file* 3. Suppose these files are filled in as follows:

```
It was a very funny
HOUSE. It had no roof,
nothing.
```
a.txt

```
those who marry want a
home. Those who don't
live at home also want a
home!
```
b.txt

```
will you marry me, love?
will you marry me,
please!
lives in my house!
```
c.txt



indice.txt (with radical extraction)

The *answer.txt* file

The answer.*txt* file contains the answer to the query contained in the query file, in our example, *query.txt*. The first line of this file should contain the number of documents that satisfy the query. The other lines contain the database files that satisfy the query, one per line, as in the following example, where we answer the query specified in our query.*txt* file. Note that all the disjunctions in the query must be taken into account, as well as disregarding *stopwords in* the query and extracting the radicals of its terms:

answer.txt

To build the answer to the query, your program must rely exclusively on the content of the inverted index generated! This means that once the inverted index has been built, the content of the documents in the database must no longer be used directly! To impress your teacher and show that you have understood the usefulness of the inverted index, try to use the structure of the index generated in memory to obtain the set of documents that answer the query (You don't need to read the inverted index from the output file indice.txt).

Be sure to test your code. You can use the testing tool provided by the teacher in the course files on Teams:

To run the corrector, download and unzip the file corrector_boolean_model.zip . Move the *.pyc files to the folder where your code is saved. Open an operating system terminal in that same folder (yes, the operating system one, not the python one), and run the command:

python3 waxm_broker_boolean_model.pyc <BASE FILE> <QUERY FILE> < F I L E WITH YOUR CODE>

If your system is Windows, perhaps this is the command:

py waxm_broker_boolean_model.pyc <BASE FILE> <QUEST FILE> < F I L E WITH YOUR CODE>

For example, assuming that the file specifying the base is called base.txt, the query file is query.txt and your code is in a file called model_boolean.py, do the following:

python3 waxm_broker_boolean_model.pyc base.txt query.txt boolean_model.py

or

py waxm_broker_boolean_model.py c base.txt query.txt boolean_model.py

You can also download the sample course bases