Homework Assignment № 1

Prof. Joao Palotti (jpalotti@andrew.cmu.edu), CMU-Q Due to **22/Mar/2017**

1 GitHub Page

Take your time to visit this GitHub page: `https://github.com/joaopalotti/cmu_67300`. There you will find the document collection that you will need to download/clone. Apart from that, you will find other useful material, such as the slides used during the lectures and plenty of code examples, both in *notebooks* and *project_code* directories. I highly recommend that you clone this repository in your system. For this, you can use the following command:

```
 $ git clone https://github.com/joaopalotti/cmu_67300
```

Alternatively, you can use a management tool such as GitHub Desktop and clone the repository from there.

2 Preprocessing and Indexing a Collection

2.1 Goal

This homework is intent for you to get used to many of useful tools/libraries from the Python world. Also, you will give your first steps towards building your search engine.

2.2 SimpleWikipedia Collection

In this course, we will use a collection of Simple Wikipedia documents. This collection consists of the most recent mirror of Simple Wikipedia documents, download from Wikipedia on the 1st of March 2017. It consists of all 174,925 articles from Simple Wikipedia and occupies 830 MB after decompressed. Documents were slightly preprocessed with a parser tool called *mwparserfromhell*, which, for example, cleaned the Wiki markup language. As a convention, the documents were named from *simple000001.txt* to *simple174925.txt* Every single document has its title in the first line and the whole content concatenated in the following lines. Therefore, the only fields that you will be able to access are title and content. Note that indexing different fields in this course is not required, but encouraged.

Below you find the output of the Linux command *head*, which shows the first 10 lines of a file belonging to this collection:

```
 $ head swcollection/simple012345.txt
Aracaju
```

```

Aracaju  is the modern capital city of the state of  Sergipe  in the northeastern region of  Brazil . It has about 470.000 inhabitants (estimate 2003) and lies between the cities of  Salvador, Bahia Salvador  and  Maceio , also in the northeast of Brazil. It was planned and built to be the state capital in  1855 .

 Category:Cities in Brazil
 Category:Sergipe
```

2.3 Example Code

In the *project_code* folder, you will find three Python scripts. Let's start by analysing the main script (*main_index.py*), this is a good example of script that will be run to evaluate your program.

Listing 1: Main Index Code

```python
from build_index import IIndex

"""
Some examples of stemmers
"""
from nltk import stem
porter = stem.PorterStemmer()

# Another one:
snowball = stem.SnowballStemmer("english")

# Or you can use another library if you want to:
# from stemming import lovins
# lovins.stem("arabic")

"""
An example of stopword
"""
stopwords = []
with open("stopwords.txt") as fstop:
    stopwords = [sw.strip() for sw in fstop.readlines()]

# you will need to adjust this parameter to map the location of this collection in your computer
myindex = IIndex("../project_collection/swcollection")
myindex.create(stopword_list=stopwords, stemming_func=porter.stem)
```

You can follow the API defined or not, but in case you do not follow it, please add to your documentation how your program should be executed. To help your start, the basic code structure is already created in the other two scripts: *build_index.py* and *utilities.py*.

An important command from the code above is its last line: *myindex.create(...)*. It is expected that this command will create the inverted index files in the disk. Later, in the following homework, you will simply load the inverted index in memory and operate on the top of it.

The two files you are expected to create are named *docs.map* and *iindex.map*. *docs.map* keeps the internal mapping from a document name to a document id. Every single line of this file follows the pattern **<external_docname,internal_doc_id>**, where *external_docname* is the name of an arbitrary file in the collection and *internal_doc_id* is an arbitrary **integer** id defined by you. As illustrated bellow:

```
$  head docs.map
simple139803.txt,163664
simple122996.txt,141941
simple075908.txt,10414
simple129555.txt,72243
simple048041.txt,150608
simple147333.txt,126675
simple059225.txt,74194
simple015744.txt,17930
simple002243.txt,7955
simple124131.txt,82248
```

The *iindex.map* will keep the inverted index. It has a format like:
<token>,<doc1>:<freq1>,<doc2>:<freq2>,...,<docN>:<freqN>, where <*doc1*> is the internal document id for a document in the collection in which token <*token*> appears <*freq1*> times. The next Linux command will find all the occurences of "joao" in the file *iindex.map*.

```
$ grep "joao"  iindex.map
  joao,163664:12,142851:2,43015:1,8715:6,16399:2
  joaozinho,28590:1
  bandeira_de_joao_pessoa,16006:1
  joaopessoa,16006:4
```

The first entry for the token *joao* shows that it appeared 12 times in document with internal id 163664. Looking up this number in file *docs.map*, we see that this is the first line of the file and is a map to file *simple139803.txt*.

3 Deliverable

It is expected as the deliverable for this homework a zip file containing:

1. The whole code that you created to generated the inverted index

2. A simple documentation with your name, AndrewID and a brief explanation of which pre-processing you plan to use in your next homework (using stopwords in your index? If yes, which stopword list? Using any stemming method? Why?).

3. Also, report the number of lines that your *iindex.map* and *docs.map* files have. What is the meaning of these two numbers?

4. Bonus part (+2 points): How large is the sum of the postings size in your inverted index? In the example shown above, this number is 8 (postings for joao has size 5, plus postings for joaozinho with size 1, and so on). This number will possible not be the same for every student. A string with the 16 characters, like *simple000000.txt* takes 53 bytes of memory, while an integer takes only 32 bytes. How many bytes are you saving when you use integer document IDs instead of the document name?