Implementing a Search Engine - Part 2

Part 2 Objective

The objective of this lab is to implement the text processor of the search engine. This processor will extract the terms of given text (HTML document or query), as follows:

1. Parse to extract HTML labels. You should split the title text from the body text.
2. For each text:
    a. Tokenize them (divide it into words)
    b. Normalize the terms (convert to non-capital letters)
    c. Filter stopwords
    d. Reduce to the stem

This processor can be used by the search engine of P1.

Implementation

You should code the class `HtmlProcessor`. So far you have used `SimpleProcessor for processing the text`, so you must change it now to `HtmlProcessor` in the class `SearchEngine` (for retrieval you do not need stopwords, so you can just instantiate with null).

`HtmlProcessor` implements two public methods to process text:

1. The first one is `Tuple<String, String> parse(String html)`, that receives the complete text of the document and extracts the real text from the HTML structure. It returns the title of the page and the body.
2. The second one is `ArrayList<String> processText(String text)`, that receives the text already parsed and returns the list of terms already processed and ready to be indexed. The whole process is implemented using the followind methods:
    - `ArrayList<String> tokenize(String text)`: receives the text already parsed and returns the list of tokens.
    - `String normalize(String text)`: receives one token and returns the normalized term.
    - `boolean isStopWord(String term)`: returns `true` if the term is stopword, and `false` if not.
    - `String stem(String term)`: receives one term and returns its stem.

Use

The `HtmlProcessor` will be used as follows. The classes `Batch` and `Interactive` run the method `processText` with the query, so you do not need to parse. However, in `Indexer` you should `parse` first to remove the HTML tags and split the title from the body. The terms to be included in the index come from the title and the body, so you should run `processText` over both of them.

You can find a text with stopwords in Aula Global (`stop-words.txt`), that you can use for indexing:

```
> java ti.SearchEngine index 2011-myIndex 2011-documentos stop-words.txt
Running first pass...
  Indexing file 2011-00-002.html...done.
  Indexing file 2011-00-003.html...done.
  Indexing file 2011-00-025.html...done.
[...]
  Indexing file 2011-99-111.html...done.
  Indexing file 2011-99-120.html...done.
...done:
```

```
  - Documents: 2088 (96,11 MB).
  - Time: 101,6 seconds.
  - Throughput: 0,95 MB/s.
Running second pass...
  Updating term weights and direct index...done.
  Updating document norms...done.
...done
  - Time: 0,3 seconds.
Saving index...done.
Index statistics:
  - Vocabulary: 209732 terms (5,05 MB).
  - Vocabulary: 92319 terms (1,84 MB).
  - Documents: 2088 documents (43,04 KB).
  - Inverted: 17,95 MB.
  - Inverted: 13,89 MB.
  - Direct: 0 MB.
  - Cache: 0 MB.
```

The size of the vocabulary and the inverted index will be reduced regarding the ones in P1. The exact numbers depend on the implementation of your `HtmlProcessor` class.

Details

Parsing

- As we have HTML documents, we need to parse them to remove HTML labels and comments and return a tuple containing the title and the body.
- We recommend the use of the Jsoup library (`http://jsoup.org`).

Tokenizing

- We should split the sentences in its corresponding words.
- Be careful with symbols and numbers.

Normalization

- Remove Capital letters
- You could also normalize numbers or symbols, but it's not the aim of the lab.

Stopwords

- Meaningless words.
- It's important to remove them to improve the efficiency.
- You can use the list `stop-words.txt` (Aula Global)
- Use an efficient structure for the stopwords: `HashSet<String> stopwords;`

Stemming

- The most usual stemmers are Porter and Krovetz.
- You can use the provided Porter stemmer Stem.java (Aula Global) or choose another, but consider that it should work for English.

Suggestions and Requirements

- For parsing and stemming use external libraries.
- Start with just few documents to test the code.
- You should be able to process all 2011 documents in a couple of minutes.
- You should just fill the code where the comment `// P2` is placed.

Evaluation

Once implemented `HtmlProcessor`, you should run the index using the code of P2, and run P1. Using the evaluation tool *ireval.jar* you should compare the effectiveness of the search engine in P1 (index created using `SimpleProcessor`) and P2 (index created using `HtmlProcessor`).

Delivery

This lab will be submitted on March 20th, where the students will have 15 minutes to demonstrate the performance of the labs.