# Lab 1 – A Simple Inverted Index

CSE 490h – Winter 2007

**Assigned –** 4/4/07 **Due –** 11:59pm on Tuesday, 4/10/07 (turn in coming)

**Review** - An inverted index is a mapping of words to their location in a set of documents. Most modern search engines utilize some form of an inverted index to process user-submitted queries. In its most basic form, an inverted index is a simple hash table which maps words in the documents to some sort of document identifier. For example, if given the following 2 documents:

Doc1 (if you don't know the story, Wikipedia it):

```
 Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.
```

Doc2:

```
 Buffalo are mammals.
```

We could construct the following inverted file index:

```
      Buffalo -> Doc1, Doc2
      buffalo -> Doc1
      buffalo. -> Doc1
      are -> Doc2
      mammals. -> Doc2
```

**Part 1** - Some words are so common that their presence in an inverted index is "noise" -- they can obfuscate the more interesting properties of a document. For the first part of this project, run a word count over an interesting corpus (e.g. Shakespeare or the web crawl data) to identify any such words.

**Part 2** - For this portion of the assignment, you will design a MapReduce-based algorithm to **calculate the inverted index over the web crawl data**. Your final inverted index should *not* contain the words identified in part 1 of this project. How you choose to remove those words is up to you; one possibility is to create multiple MapReduce passes, but there are many possible ways to do this. Also, the format of your MapReduce output (i.e. the inverted index) must be simple enough to be machine-parseable; it is not impossible to imagine your index being one of many data structures used in a search engine's indexing pipeline. Lastly, your submitted indexer should be able to run successfully on the corpus specified in your write-up. "Successfully" means it should run to completion without errors or exceptions, and generate the correct word->DocID mapping. You are required to turn in all relevant `Mapper` and `Reducer` Java files, in addition to any supporting code or utilities.

**For web crawl data info:** http://www.cs.washington.edu/education/courses/cse490h/07wi/whmr.html

**Part 3 -** In addition to the requirements detailed above, implement at least two extensions of your choice. The extensions may be as simple or as complex as you'd like; the primarily goal is to give you a chance to play with the MapReduce system. We have provided some sample extensions:

- A naive parser will group words by attributes which are not relevant to their meaning. Modify your parser to "scrub" words. You can define "scrub" however you wish; some suggestions include case-insensitivity, punctuation-insensitivity, etc. You will get extra karma for creating a language-independent scrubbing algorithm, but this is not required.
- Write a query program on top of your inverted file index, which will accept a user-specified word (or phrase!) and return the IDs of the documents which contain that word.
- Instead of creating an inverted file index (which maps words to their document ID), create a full inverted index (which maps words to their document ID + position in the document). How do you specify a word's position in the document?
- If you created a full inverted index, write a query program on top of the index which returns not only the document IDs but also a text "snippet" from each document showing where the query term appears in the document.
- Modify your parser to strip HTML tags (if your corpus includes HTML). Eg, "`<B>Bolded text</B>`" should parse as "`Bolded`" and "`text`", not "`<B>Bolded`" and "`text</B>`"
- Run your indexer on a non-English corpus. How do the results differ for western European languages (e.g., German, Spanish, Portuguese)? How about non-western-European languages (e.g., Russian, Korean, Vietnamese)?

**Write-up:**

1. How long do you think this project would take you to finish?
2. How much time did you actually spend on this project?
3. Acknowledge any assistance you received from anyone except assigned course readings and the course staff.
4. What test corpus did you load into HDFS? Where is it located (i.e. HDFS path)?
5. What, if any, "noisy words" did you find as a result of your WordCount? By what criteria did you determine that they were "noisy"? Were you surprised by any of these words?
6. Which extensions did you do? Please also detail any interesting results you found.