CS 345: Programming Assignment 2 (Due: Nov 18, 2013 before 23:59)

Rules. Rules. Teams of no more than two students as explained in syllabus (Handout 1). This is to be handed back no later than midnight on the Monday it is due (see information below) in electronic form as a single tar or zip file decompressible on an AFS machine (afsconnect1.njit.edu or afsconnect2.njit.edu), where testing would take place. (At a minimum, the zipfile should contain a text file `HW1_ABCD_EFGH.txt` as explained below.) The code should be compilable/interpretable and executable on one or the other identical AFS machines in any language available there (emphasis C, C++, Java, Python, Perl).

fdsee : A file-based desktop search engine (Index building)

1 The Index of fdsee

1.1 Objectives of the assignment

This assignment is a continuation of PA1. The input to this assignment is the output of

```
% ./fdsee token dfname
```

where `dfname` is a generic name for either a file or a directory name. Alternatively, it can be the output of

```
% ./fdsee stopword dfname
```

if you have/had a successful implementation of Step 3.

In Step 1, the output of the tokenizer will be first stemmed. You only need to implement the elementary (Harman) stemming algorithm, even if it gives something simple and questionable.

The output before or after stemming of previous phases is ordered according to `docID` as documents are tokenized in some sequence and all token of a given document are extracted before moving to the next document. In Step 2, we then order the output stream based on `wordID` first (increasing order), then `docID` (increasing order), then `attr` (increasing order) and finally `wpos` (increasing order). Using this information, you will build an inverted index where you will store information about word occurrences (i.e. inverted lists) in the form of a combined `vocabulary` and `occurrence list (inverted list)` structures as described in class and summarized in this document for completeness. For testing purposes this construction will have an interesting side-effect.

In Step 3, you will be asked to design a query system that implements simple logical (`AND` and `OR` operations.

This is the minimum implementation required of this assignment. You can enrich this implementation by adding additional features.

1.2 Project deliverables

A single executable/interpretable file will be the result of your compilable/interpretable source code into the form of a file named `fdsee`. (If this causes problems with prior implementations, you can name it `fdsee2` instead.) Every source file you submit must include in the form of comments in the first 5 lines the names of the members of the group including the last four digits of their NJIT IDs. In addition a file named `HW1_ABCD_EFGH.txt` (or `HW1_ABCD.txt` if the team has only one member) needs to be included (that also conforms to the first-5-line convention) that includes instructions for compilation/interpretation, bugs, and anything else of interest (eg extensions).

1.3 Step 1: Stemming

The executable file (or class) `fdsee` will read the command line and behave as follows.

```
% ./fdsee stem dfname
```

The first argument in the command line (after the name of the executable file) denotes an action such as `token, tokendebug, searchable, stopword` used in the previous programming assignment. The second argument is a file/directory name.

For action `stem` you need to take the output of `fdsee` and apply to it Harman's stemming algorithm that eliminates very simple suffixes (eg. plural). The output is a stream similar to the input.

Note that it is quite possible that say wordID `25` already corresponds to word `algorithm` and there is another wordID say `35` that corresponds to word `algorithms`. If as a result of the stemming method we have a translation of `algorithms` into `algorithm` you need to decide what to do with wordIDs 25 and 35. The design decision is yours. One way to deal with it is to convert for example `35` into `25`; an issue is what to do with wordID `35` that is left unused. If however, `25, 35` do not exist, a renaming might take place instead.

```
  HarmanStemmingAlgorithm(word)
 1. If word ends in -ies but not -eies or -aies
     then -ies --> -y;
 2. If word ends in -es  but not -aes, -ees or -oes
     then -es  --> -e;
 3. If word ends in -s but not -us or -ss
     then -s   --> -;
```

1.4 Step 2: Inverted Index

Subsequently you take the stemming output (or if you decide to skip that part) the output of `./fdsee token dfname` and generate an inverted index. You call the inverted index creation module of yours with a call to `invert` as follows.

```
% ./fdsee invert  dfname
```

Thus `invert` can take the output of `stem` or `stopword` or `token` or you can just make it behave as if it combines in itself the step of `step, stopword, token` that you have implemented. The choice is yours.

1. Vocabulary and occurrence lists. An inverted index consists of a `Vocabulary` and `occurrence` or `inverted lists`.

2. Vocabulary: technical details. The `Vocabulary` will contain the `wordID`s in sorted order. (The lexicon of PA1 contains wordID and word combinations not necessarily in sorted order.) For every entry in the Vocabulary you need to record information such as `wordID`, `Ndocs`, `Nhits`, and potentially (depending on your implementation and language combination) a pointer `LocP` to the actual entries.

Entry `Ndocs` counts the number of distinct documents that contain `wordID`. Entry `Nhits` counts the number of occurrences of `wordID` in all documents (a given word might occur more than once in a document) and in any possible form (attr).

The (potential) pointer `LocP` points to the area in which you will maintain information about the given `wordID`. Such information will be a list of lists (whether you implement it in an array, linked list or whatever else it is your decision) of occurrences of `wordID`. Every document in which `wordID` occurs will have a separate list. Therefore the number of lists for `wordID` will be `Ndocs`. The elements in each such list can be be ordered by `attr` and within each `attr` by `wpos`. Note that `if you think this is not a good idea,` you can provide a written justification with the deliverables text file. (Can does not mean should be.)

1.4.1 Side Effects

Besides the side-effects of `fdsee` of PA1, additional side-effects will be generated.

Side-Effect 1: Vocabulary file. For a start `invert` will generate a file named `fdsee_vocabulary` that includes a dump of the vocabulary in the form of tuplets (`wordID`, `word`, `Ndocs`, `Nhits`). It suffices to print the four elements of the tuplet, one set of elements per line without the surrounding parentheses or the commas but include one space inbetween.

Side-Effect 2: Directory `fdsee_invindex`. Then `invert` will create a directory named `fdsee_invindex`. Inside that directory you will create a number of files one for each word whose `wordID` is in the vocabulary. In file `word` you will store the occurrence lists of `word` in the form of triplets (`docID`,`wpos`, `attr`). The format of the file will be as follows.

The first entry (or line) in such a file will be two decimal (base 10) integer. If its value is `nhits1`, it will mean that the following `nhits1` tuples/lines belong to the same document say `docid1`. The `nhits1` tuplets are listed in the form `docid, wpos, attr` without the surrounding parentheses. After these `nhits1` tuples another positive value might be listed to indicate another group of occurrences for another document. If a `0` appears in a line, it will indicate the END-OF-FILE. Thus a positive `nhits1` value indicates that `nhits1` lines are to follow and this facilitates the reading of those lines. (The total number of lines of this file will be `Nhits` of the vocabulary plus `Ndocs`, BTW.)

Note that we do not ask you to sort the groups of hits other than the obvious by `docid` order implied by their listing. You might choose to have them sorted by `attr` or `wpos` or not. The intent of all the side-effects is to be able to build an inverted index from all the generated files without requiring the re-reading of the web-pages. ■

Date Posted: 9/16/2013