CS 345: Programming Assignment 3 (Due: Dec 9, 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 (Query processing)

1 The query engine of fdsee

1.1 Objectives of the assignment

This assignment is a continuation of PA1 and/or PA2. The input to this assignment is the output of the previous one and in particular their side effects including the `fdsee_lexicon, fdsee_doclist , fdsee_vocabulary , fsee_invindex` files or directories. The intent is to use this infrastructure in answering some very simple Boolean queries.

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 Query engine implementation

Having completed the index we ask you to implement a command line-based implementation of a simple up to three-term query language.

```
% ./fdsee and2    term1 term2
% ./fdsee and3    term1 term2  term3
% ./fdsee andnot  term1 term2
% ./fdsee and2not term1 term2  term3
% ./fdsee and3not term1 term2  term3 term4
% ./fdsee or2     term1 term2
% ./fdsee or3     term1 term2  term3
% ./fdsee next    term1 term2
% ./fdsee next5   term1 term2
% ./fdsee next25   term1 term2
```

Each one of the operations above (implicitly) assumes that `./fdsee invert dfname` is first executed. (That is, `invindex` already exists and is well formed.)

The outcome of `and2` is to read the occurrence lists of the wordIDs of `term1` and `term2` and find their common intersection docID-wise, i.e. find those documents that contain both terms. The output is a list of the documents containing both words, one per line. Each line prints not only the `docID`s but also the qualified URLs to each document. The `and3` allows for a 3-term conjunction.

The outcome of `andnot` is to read the occurrence lists of the wordIDs of `term1` and `term2` and find those documents in which `term1` appears but not `term2`. The printout of the result is as before. For `and2not` the documents that contain `term1, term2` but not `term3`, and for `and3not` the documents that contains `term1,term2, term3` but not `term4`.

The outcome of `or` is that of a disjunction and the two variant behave analogously to `and2, and3`.

Note that for this part we only need to use `docid` information and neither `wpos` nor `attr` to generate an answer to the query. A more elaborate processing can occur using `wpos` and `attr` position that will also rank the results.

Alternatively, you may implement the following

```
% ./fdsee boolean "mterm1  op1 mterm2 op2 mterm3 op3 mterm4 op4"
% ./fdsee next    term1 term2
% ./fdsee next5   term1 term2
% ./fdsee next25   term1 term2
```

where `mterm` is either a `term` or `-term` and the $-$ indicates a not. An `op` then can be either an `and` or an `or`. Conjunctions have higher precedence than a disjunction! The operators `next` and `next5` checks whether the terms `term1` or `term2` appear next to each other in a document (i.e. their `wpose` differ by one for the same `docID`) or within distance of 5 and 25 respectively. ■

Date Posted: 9/16/2013