New Jersey Institute of Technology

A. V. Gerbessiotis

February 27, 2006

(120 points) Due: April 3, 2006

CIS 750

Spring 2006

Programming Assignment 3

1 Introduction

This assignment is a continuation of the previous two. In this assignment you will make use of the files that were created as side effects in the previous two: (a) `doclist`, (b) `lexicon`, (c) `vocabulary`, and (d) the indexing-specific files in directory `invindex`.

You will be given a query in some restricted form that say contains up to five terms and the logical operators `OR`, `AND`, operator `-` (to denote NOT in front of a term) and `NEXT-n`, for $n = 1, \ldots, 9$.

You will then be asked to implement an evaluation function that retrieves all documents that satisfy the query and also prints the list of documents that satisfy the query in the output. In addition you will rank these documents based on some ranking measures an outline of which will be given to you but whose further details you are responsible to determine.

This is the minimum implementation required to gain you the `120` points of this assignment. You can enrich this implementation by adding additional features. Or you can think ahead of adding even more features and components in the context of the independent work component of the course.

2 Deliverables

The relevant files will become available in `homework/pa3` as described in Handout 2. A single executable file will be the result of your compilable source code into the form of a file named `pa3`.

3 Part 1 : Query parsing and evaluations (60 points)

The executable file `pa3` will read the command line and behave as follows.

```
% ./pa3 resolve "mterm1 op1 mterm2 op2 mterm3 op3 mterm4 op4 mterm5"
```

The first argument in the command line (after the name of the executable file) denotes the action that will be performed i.e. `resolve` and print the documents that satisfy the query that is described by the second command-line argument. Printing the documents means giving the docID and the qualified name of the document (as printed under option `debug-token` in PA1).

The second argument is a query string enclosed in double quotes " ". An `op` is an operator and `mterm` is a modified-term. The query is limited to at MOST 4 operators and at most 5 terms.

Rule 1. The logical operators are `OR` and `AND` whith the normal meaning and semantics. The operator `-` standing for `NOT` immediately precedes the term it is associated with i.e. `-algorithm` means `NOT algorithm`. In addition the operator `NEXT-n` is defined for $n = 1, \ldots, 9$. `a NEXT-5 b` if terms `a,b` are in the same docID and `|wordpos(a) - wordpos(b) | <= 5` , where `wordpos` is the wordposition in a document for a term such as `a` or `b`. A `NEXT-n` operator will have priority right below `-` and above `AND`. So priority-wise from highest to lowest priority we have the operators: `NOT-n, - , AND, OR`.

Rule 2. A modified term such as `mterm` is a `term` or `-term`.

Rule 3. A `term` (as in Rule 2) can be a `indexable-term` or `stemmable-term`.

Discussion of Rules 2 and 3. To explain Rules 2 and 3 let us give an example: `-algorithms`

The modified term follows the pattern `-term` of Rule 2, i.e. `algorithms` is a term.

`algorithms` is also a stemmable-term because it is not an indexable-term. The word `algorithms` has been stemmed in PA2 into `algorithm` and only the latter is indexed in *invindex*. A `stemmable-term` is first-stemmed (using the algorithm you implemented in PA2) and then and only then can be searched i.e. it has become an `indexable-term`.

Therefore it is imperative that you stem the query terms the way you stemmed the index term in PA2. That is, some cleaning of data needs to be performed at query time as part of the requirements of Part 1 of this assignment.

4 Part 2 : Ranking and Evaluation (40 points)

If the operation to be performed is not just a `resolve` (i.e. Boolean model-based resolution of the query) but a `rank` the output of Part 1 needs to be ranked, i.e. you need to assign ranks to each document printed in Part 1 and list those documents in non-increasing order (i.e. highest to lowest rank).

```
% ./pa3 rank "mterm1 op1 mterm2 op2 mterm3 op3 mterm4 op4 mterm5"
```

How to rank a term.

How do we evaluate `s(term, dj)` i.e. how do we find the similarity between term `term` and document `dj` ?

For every `dj` and `wordid` we collect information $(c_0, \ldots c_4)$ that counts the number of occurrences of `wordid` as type $0, \ldots, 4$ term in the document. We also define weights $(w_0, \ldots, w_4)$.

Their default values are $(0.1, 0.3, 0.25, 0.20, 0.15)$ for the five types defined in PA1. Alternatively, a user-defined file `pa3-setup` might contain 6 values one per line with the last one being `-1.0`, an END-OF-LIST separator. If such a file exists, the values there overwrite the default one. Note that sum of the 5 values must all add up to 1.

Then, for a simple term (`indexable-term`) we have the term weight

$$w_{ij} = \sum_{i=0}^{4} c_i w_i.$$

Term weights weigh in favor of long documents. A better measure is the normalized version $W_{ij} = \dfrac{w_{ij}\mathbf{idf_i}}{\sqrt{\sum_i w_{ij}^2 \mathbf{idf_i^2}}}$

However requires the determination of idf_i from incomplete information that might need to be retrieved (eg. n_i).

You are allowed to use either w_{ij} or W_{ij} in the remainder. This can only affect part 3 but not part 2. Either contribution is generically denoted by $s(term_i, d_j)$ in the remainder to outline the similarity between the query term and the document in question.

What if we want to find the similarity between a complex query and document d_j?

What if we have `T = term1 AND term2` ? Then, a choice of the similarity function can be, $s(T, d_j) = s(term1, d_j) + s(term2, d_j)$.

What if we have `T = term1 OR term2` ? Then, $s(T, d_j) = \max\{s(term1, d_j), s(term2, d_j)\}$.

What if we have `T = term1 NEXT-n term2` ? Then, $s(T, d_j) = (s(term1, d_j) + s(term2, d_j)) * 1.1 * (n+10)/10$.

Therefore you can determine the similarity between the query q and the document in question q_j as $s(q, d_j)$. This could determine the rank of document d_j for the query.

Repeat this for all the documents output during the `resolve` execution and print the document in nonincreasing rank to complete the requirements of this part.

5 Part 3 : Quality of results (20 points)

The quality of the output will be judged also. Up to 20 points will be determined based on it.

Bonus points (up to 50) might be given based on the overall quality and efficiency of the implementation.

6 Comments

For this part we assume all side effects files are located in the directory from which the executable `pa3` is called.