Web Information Retrieval

Term Project

Summer 2009

At the end of the second phase of the project, your program is able to parse and index a set of documents. The end result is that your program has built a term list and a collection of posting lists, one for each term. In this phase of the project, you are to add two more components to your program. One is the term weight which we will use the well-known *tf-idf* in the vector model. The other is the retrieval and ranking module, which should return a ranked list of documents for a given query.

Make sure that both will be added such that a user can actually work through the web interface. That is, queries can be accepted through a browser, and the sorted results should be displayed in the browser as well.

To build the term weight, you will need to use the statistics your program has collected in the indexing phase of the project. There you should have collected the term frequency for each term in every document. Now you will need to compute the *document frequency*. Document frequency of a term is defined as the number of documents that contains the term, which is simply the length of the posting list. With *df* and *tf*, we can compute *idf*, inverted document frequency and the product of *tf* and *idf*. Thus the weight for term i in document j is computed as

$$w_{i,j} = tf_{i,j} * idf_i = tf_{i,j} * log(\frac{doc\ count}{df_j})$$

This value $w_{i,j}$ typically is stored in the *DocNode* on the posting list. As you can see, your program should be able to compute the term weight for all terms when the indexing is finished, before any search query is processed.

Following is a description of a set of algorithms that can accomplish the above task. We discussed these algorithms in class. You have implemented a portion of it in the indexing phase.

```
Algorithm to create an inverted index
-------------------------------------
Create an empty index term list I;
For each document D in the document set V  {
   For each non-empty token T in D   {
      If T is not already in I
        Insert T into I;
      Find the location for T in I;
      If (T, D) is in the posting list for T
        Increase its term frequency for T;
      Else
        Create (T, D);
```

```
        Add it to the posting list for T;
   }
}
end of algorithm to create an inverted index


Algorithm to compute IDF
------------------------
Let N be the total number of documents;
For each token T in I  {
  Determine the total number of documents, M,
     in which T occurs (the length of T's posting list);
  Set the IDF for T to be log(N/M);
}
end of algorithm to compute IDF

Algorithm to compute document vector length
-------------------------------------------
Initialize all document vector length to be 0.0;
For each token T in I  {
   Let idf be the IDF weight of T;
   For each token T in document D (use posting list)   {
     Let C be the count of T in D;
     Increment the vector length of D by (idf * C)^2;
   }
}

For each document D in the document set  {
   Set the vector length of D to be the square-root of the current
        stored length;
}
end of algorithm to compute document vector length
```

As can be seen from the algorithm, it would be better if you maintain a document list as well (similar to term list, except this is for documents) where each node stores the document name, document id, and other statistics such as document vector length. Both the posting list and this document vector should be sorted according to either document name or document id. Using document id throughout the process instead of using its full name would save space if that is a concern.

When a query is received, the query processing algorithm is executed and a list of ranked documents is retrieved. You should decide how many documents should be listed in each return page. Typical choices are listing the top 10 or top 20 as the first page.

Here is a description of the algorithm for retrieval.

```
Inverted-index retrieval algorithm
----------------------------------
Create a vector Q for the query;
Create an empty collection R to store retrieved documents with scores;
For each token T in Q   {
```

```
   Let idf be the IDF of T and K be the count of T in Q;
   Set the weight of T in Q:  W = K * idf;
   Let L be the posting list of T from I (the term list);
   For each node O in L   {
      Let D be the document of O, and C be the term frequency of T in D;
      If D is not already in R      {
         Add D to R;
         Initialize its score to 0.0;
      }
      Increment D's score by W * idf * C;
   }
}
Compute the length L of the vector Q;
For each retrieved document D in R   {
   Let S be the current accumulated score of D;
   // S is the inner-product of D and Q now
   Let Y be the length of D;
   Normalize D's final score S = S/(L*Y); // cosin similarity
}
Sort retrieved documents in R by final score;
Return the top-N results as specified;
end of inverted-index retrieval algorithm
```

What to Hand In

Your team needs to hand in the following in the order given. Please staple all pages.

1. A team report for phase three of the project with a cover sheet. The report shouldn't be too long, maybe two to three pages. The report should include the following as a minimum.
   (a) A description of the roles of each team member and the contributions of each member;
   (b) A summary of the working process, e.g. what the team started with, what the team has accomplished, any problems encountered, how the team solved them, any thoughts on the project, among others.
2. Source code for the programs.
3. Snapshots of sample runs.
4. Email the instructor a copy of the complete source code in `zip` or `tar` format.