DD2476 Search Engines and Information Retrieval Systems

Assignment 2: Ranked Retrieval

Johan Boye and Hedvig Kjellström

The purpose of Assignment 2 is to learn how to do ranked retrieval. You will learn 1) how to include tf-idf scores in the inverted index; 2) how to handle ranked retrieval from multiword queries; 3) how to use PageRank to score documents; and 4) how to combine tf-idf and PageRank scoring.

The recommended reading for Assignment 2 is that of Lectures 4-5.

Assignment 2 is graded, with the requirements for different grades listed below. In the beginning of the oral review session, the assistant will ask you what grade you aim for, and ask questions related to that grade. All the tasks have to be presented at the same review session – you can not complete the assignment with additional tasks after it has been examined and given a grade. ***Come prepared to the review session!*** *The review will take 10 minutes or less, so have all papers in order.*

E: *Completed Task 2.1, 2.2 and 2.3 with some mistakes that could be corrected at the review session.*
D: *Completed Task 2.1, 2.2 and 2.3 without mistakes.*
C: *D + Serious attempt to address Task 2.4.*
B: *C + Completed Task 2.4 with a ranking very similar to the exact one in Task 2.3.*
A: *B + Completed Task 2.5, showing understanding.*

These grades are valid for review February 19, 2013. See the web pages www.csc.kth.se/DD2476/ir13/laborationer for grading of delayed assignments.

Assignment 2 is intended to take around 50h to complete.

Computing Framework

For Task 2.1, 2.2, and 2.5, you will be further developing your code from either Task 1.3 or 1.5.

For Task 2.3 and 2.4, you will be using a source code skeleton found in the course directory `/info/DD2476/ir13/lab/pagerank`. Copy this directory to your home directory.

> If you are using your own computer you will need to copy the sub-directory `svwiki_links` as well.

The `pagerank` directory contains only the file `PageRank.java`, which is compiled simply by

```
javac PageRank.java
```

The program is executed as follows:

`java PageRank` *linkfile*

for instance

```
java PageRank /info/DD2476/ir13/lab/svwiki_links/links1000.txt
```

The link files are found in the folder `svwiki_links` in the course directory. Each line has the following structure:

```
1;365,959,944,
```

meaning that the article in `1.txt` is linking to the articles in `365.txt`, `959.txt` and `944.txt`. There are three such link files, representing the link structure within the first 1000 articles, 10000 articles, and all articles, respectively. The `PageRank` program reads such link files and represents the link structure internally by hash tables. **Note that the program internally uses other ID numbers for files!** For instance, article `365.txt` will be internally represented by some number which is not 365. (This is because we want the program to work for file names that do not happen to be integers.)

A convenient way of seeing what articles are about, is to look in the file `articleTitles.txt`, for instance by:

```
> egrep "^365;"
  /info/DD2476/ir13/lab/svwiki_links/articleTitles.txt
365;Danmark
```

Task 2.1: Ranked Retrieval

Extend the `search` method in the `HashedIndex` (or `MegaIndex`) class to implement ranked retrieval. For a given search query, compute the cosine similarity between the tf-idf vector of the query and the tf-idf-vectors of all matching documents. Then sort documents according to their similarity score.

You will need to add code to the `search` method, so that when this method is called with the `queryType` parameter set to `Index.RANKED_QUERY`, the system should perform ranked retrieval. You will furthermore need to add code to the `PostingsList`, `PostingsEntry`, and `HashedIndex` (or `MegaIndex`) classes, to compute the cosine similarity scores of the matching documents. To sort the matching documents, assign the score of each document to the `score` variable in the corresponding `PostingsEntry` object in the postings list returned from the `search` method. If you do this, you can then use the `sort` method in the built-in `java.util.Collections` class.

When your implementation is ready, load the `svwiki/files/1000` data set, select the "Ranked retrieval" option in the "Search Options" menu, and try the query "december". It should work like this:

december

Found 117 matching document(s)

0. svwiki/files/1000/726.txt 0,04564
1. svwiki/files/1000/929.txt 0,02402
2. svwiki/files/1000/955.txt 0,02298
3. svwiki/files/1000/318.txt 0,01544
4. svwiki/files/1000/183.txt 0,01485
5. svwiki/files/1000/768.txt 0,01361
6. svwiki/files/1000/184.txt 0,01039
7. svwiki/files/1000/952.txt 0,00995
8. svwiki/files/1000/401.txt 0,00956
9. svwiki/files/1000/977.txt 0,00599
10. svwiki/files/1000/973.txt 0,00597
etc.

Our list above was computed with ordinary tf-idf scores and document length = [*# words in the document*]. Depending on exactly how you compute the similarity scores, the numerical values and the ordering of the documents above might differ slightly from those produced by your program – this is fine. However, your list should not be fundamentally different from the one above.

At the review

To pass Task 2.1, you should show that the search engine indeed returns 117 documents in response to the query **december** on the `svwiki/files/1000` data set, with a ranking *similar* to the one above *(exact match not required)*. You should also be able to explain all parts of the code that you edited, draw the data structure on paper, and explain from that figure how a ranked one-word query is executed.

Task 2.2: Ranked Multiword Retrieval

Modify your program so that it can search for multiword phrases like "november eller december", and present a list of ranked matching documents. All documents that include at least one of the search terms should appear in the list of search results.

When your implementation is ready, load the `svwiki/files/1000` data set, select the "Ranked retrieval" option in the "Search Options" menu, and try the phrase:

november eller december

Found 474 matching document(s)

0. svwiki/files/1000/726.txt 0,08984
1. svwiki/files/1000/922.txt 0,03355
2. svwiki/files/1000/768.txt 0,02722
3. svwiki/files/1000/955.txt 0,02626
4. svwiki/files/1000/514.txt 0,02560
5. svwiki/files/1000/929.txt 0,02402
6. svwiki/files/1000/646.txt 0,02310
7. svwiki/files/1000/293.txt 0,02295

8. svwiki/files/1000/954.txt 0,02256
9. svwiki/files/1000/563.txt 0,01925
10. svwiki/files/1000/953.txt 0,01908
etc.

Again, the list and similarity scores above might differ slightly from your solution.

Why do we use a union query here, but an intersection query in Assignment 1?

Look up the document titles of the highest ranked documents for a couple of queries, using the instructions in Computing Environment above. Do the the hits seem reasonable? Why?

At the review

To pass Task 2.2, you should show that the search engine indeed returns 474 documents in response to the query **november eller december** on the `svwiki/files/1000` data set, with a ranking *similar* to the one above. You should also be able to explain all parts of the code that you edited, draw the data structure on paper, and explain from that figure how a ranked multi-word query is executed. In addition, you should be able to discuss the questions in italics above.

Task 2.3: Computing PageRank with Power Iteration

Your task is to **extend the class `PageRank.java` so that it computes the pagerank of a number of Wikipedia articles** given their link structure. Use the standard power iteration method, as described in Lecture 5 and in the textbook (Section 21.2.2), and run your program both on `links1000.txt` (containing the link structure of the 1000 first documents) and `links10000.txt` (containing the link structure of the 10000 first documents). (Running the program on `links.txt`, which contains the entire Wikipedia link structure, is likely to take a very long time.)

Make sure your program prints the pagerank of the 50 highest ranked pages. Use the array `docName` to translate from internal ID numbers to file names.

A correctly implemented power iteration with c = 0.85 gives the following top 50 ranking for `links10000.txt` (not necessarily with the same pagerank values):

```
1.  1081 0,00393   11. 7031 0,00235   21. 5115 0,00170   31. 3105 0,00148   41. 837  0,00136
2.  522  0,00382   12. 3094 0,00228   22. 5621 0,00169   32. 723  0,00143   42. 6039 0,00135
3.  454  0,00357   13. 2381 0,00221   23. 425  0,00160   33. 6074 0,00142   43. 3743 0,00134
4.  2634 0,00354   14. 1306 0,00214   24. 6070 0,00156   34. 2635 0,00142   44. 2    0,00132
5.  365  0,00286   15. 9765 0,00192   25. 838  0,00155   35. 8071 0,00141   45. 4919 0,00132
6.  36   0,00277   16. 6287 0,00192   26. 6722 0,00154   36. 8098 0,00140   46. 664  0,00131
7.  526  0,00269   17. 1432 0,00188   27. 8184 0,00152   37. 2343 0,00138   47. 6451 0,00131
8.  3930 0,00264   18. 4762 0,00186   28. 1584 0,00150   38. 2136 0,00137   48. 8813 0,00129
9.  1324 0,00246   19. 2353 0,00186   29. 3931 0,00149   39. 21   0,00137   49. 5559 0,00127
10. 483  0,00239   20. 5608 0,00176   30. 6907 0,00149   40. 1524 0,00137   50. 2134 0,00127
```

The highest ranked document, 1081, has the title Latin, while the lowest ranked document, 669, has the title Gunnebo IP (a sports field in a tiny town in Sweden). Does this pagerank ordering seem reasonable? Why?

Look up the titles of some other documents with high rank. What is the trend with decreasing pagerank?

At the review

To pass Task 2.3, you should show that the method returns a very similar top 50 ranking for `links10000.txt` to the one shown above. You should also be able to explain all parts of the code that you edited, and be able to discuss the questions in italics above.

Task 2.4: Approximations of PageRank (C or higher)

The power iteration method is very time-consuming, but it is possible to compute approximate pagerank values in far less time. Implement at least two of the methods for approximate pagerank computation mentioned in Lecture 5. Assess how good the algorithms are (i.e., how fast they converge and how similar the solution is to the exact solution) by running them on `links1000.txt` and `links10000.txt`. If they have converged, you should get a similar ordering as with the power iteration algorithm.

If you have working memory enough, you can now calculate the approximate pagerank of all Wikipedia articles, listed in `links.txt`. What is the highest ranked document in the whole structure?

At the review

To pass Task 2.4, you should show that both approximative methods return the a top 50 ranking for `links10000.txt` very similar to the power iteration solution shown in Task 2.3. You should also be able to explain all parts of the code that you edited.

Task 2.5: Combine tf-idf and PageRank (A)

Your final task is to integrate your results from Task 2.3 and 2.4 into the search engine we have been developing in Assignment 1 and Task 2.1 and 2.2. When doing a ranked query, make sure that the **score is computed as a function of the tf-idf similarity score and the pagerank** of each article in the result set. Design the combined score function so that you can vary the relative effect of tf-idf and pagerank in the scoring.

Use the pageranks you computed from `links.txt` in Task 2.4, or from `links10000.txt` in Task 2.3. You should pre-compute the pageranks.

You will need to add code to the `search` method, so that when this method is called with the `rankingType` parameter set to `Index.TF_IDF`, the system should perform ranked retrieval based on tf-idf score only, with the `rankingType` parameter set to `Index.PAGERANK`., only pagerank should be regarded, and with the `rankingType` parameter set to `Index.COMBINATION`, your combined score function is used to rank the documents.

When you are ready, load the `svwiki/files/1000` data set, select the "Combination" option in the "Ranking Score" menu, and try the query "november eller december". It should work like this:

november eller december

Found 474 matching document(s)

1. …

What is the effect of letting the tf-idf score dominate this ranking? What is the effect of letting the pagerank dominate?

At the review

To pass Task 2.5, you should present a plausible function for combining tf-idf and pagerank scores, and be able to motivate your choice of function. Moreover, you should show that the search engine indeed returns 474 documents in response to the query **november eller december** on the `svwiki/files/1000` data set, and be able to discuss the effect of tf-idf and pagerank on the subsequent ranking.