

CHAPTER 31



Information Retrieval

Practice Exercises

- 31.1** Compute the relevance (using appropriate definitions of term frequency and inverse document frequency) of each of the Practice Exercises in this chapter to the query “SQL relation”.

Answer:

We do not consider the questions that contain neither of the keywords because their relevance to the keywords is zero. The number of words in a question includes stop words. We use the equations given in Section 31.2 to compute relevance; the log term in the equation is assumed to be to the base 2.

Q#	#wo- -rds	# “SQL”	#“rela- -tion”	“SQL” term freq.	“relation” term freq.	“SQL” relv.	“relation” relv.	Tota relv.
1	84	1	1	0.0170	0.0170	0.0002	0.0002	0.0004
4	22	0	1	0.0000	0.0641	0.0000	0.0029	0.0029
5	46	1	1	0.0310	0.0310	0.0006	0.0006	0.0013
6	22	1	0	0.0641	0.0000	0.0029	0.0000	0.0029
7	33	1	1	0.0430	0.0430	0.0013	0.0013	0.0026
8	32	1	3	0.0443	0.1292	0.0013	0.0040	0.0054
9	77	0	1	0.0000	0.0186	0.0000	0.0002	0.0002
14	30	1	0	0.0473	0.0000	0.0015	0.0000	0.0015
15	26	1	1	0.0544	0.0544	0.0020	0.0020	0.0041

- 31.2** Suppose you want to find documents that contain at least k of a given set of n keywords. Suppose also you have a keyword index that gives you a (sorted) list of identifiers of documents that contain a specified keyword. Give an efficient algorithm to find the desired set of documents.

Answer:

Let S be a set of n keywords. An algorithm to find all documents that contain at least k of these keywords is given below.

This algorithm calculates a reference count for each document identifier. A reference count of i for a document identifier d means that at least i of the keywords in S occur in the document identified by d . The algorithm maintains a list of records, each having two fields – a document identifier, and the reference count for this identifier. This list is maintained sorted on the document identifier field.

```

initialize the list  $L$  to the empty list;
for (each keyword  $c$  in  $S$ ) do
  begin
     $D :=$  the list of documents identifiers corresponding to  $c$ ;
    for (each document identifier  $d$  in  $D$ ) do
      if (a record  $R$  with document identifier as  $d$  is on list  $L$ ) then
         $R.reference\_count := R.reference\_count + 1$ ;
      else begin
        make a new record  $R$ ;
         $R.document\_id := d$ ;
         $R.reference\_count := 1$ ;
        add  $R$  to  $L$ ;
      end;
    end;
  for (each record  $R$  in  $L$ ) do
    if ( $R.reference\_count \geq k$ ) then
      output  $R$ ;

```

Note that execution of the second *for* statement causes the list D to “merge” with the list L . Since the lists L and D are sorted, the time taken for this merge is proportional to the sum of the lengths of the two lists. Thus the algorithm runs in time (at most) proportional to n times the sum total of the number of document identifiers corresponding to each keyword in S .

- 31.3** Suggest how to implement the iterative technique for computing PageRank given that the T matrix (even in adjacency list representation) does not fit in memory.

Answer:
FILL

- 31.4** Suggest how a document containing a word (such as *leopard*) can be indexed such that it is efficiently retrieved by queries using a more general concept (such as “carnivore” or “mammal”). You can assume that the concept hierarchy is not very deep, so each concept has only a few generalizations (a concept can, however, have a large number of specializations). You can also assume that

you are provided with a function that returns the concept for each word in a document. Also suggest how a query using a specialized concept can retrieve documents using a more general concept.

Answer:

Add doc to index lists for more general concepts also.

- 31.5** Suppose inverted lists are maintained in blocks, with each block noting the largest popularity rank and TF-IDF scores of documents in the remaining blocks in the list. Suggest how merging of inverted lists can stop early if the user wants only the top K answers.

Answer:

For all documents whose scores are not complete, use upper bounds to compute the best possible score. If the K th largest completed score is greater than the largest upper bound among incomplete scores, output the top K answers.

