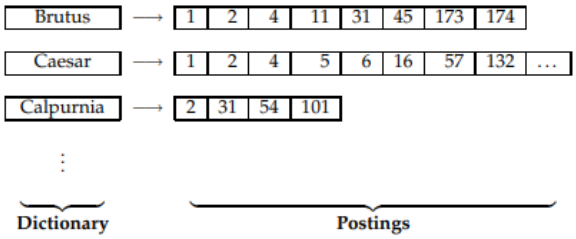
Information Retrieval Assignment 1

Exercise 1.1

Draw the inverted index that would be built for the following document collection.
(See Figure 1.3 for an example.)



► **Figure 1.3** The two parts of an inverted index. The dictionary is commonly kept in memory, with pointers to each postings list, which is stored on disk.

Doc 1 new home sales top forecasts

Doc 2 home sales rise in july

Doc 3 increase in home sales in july

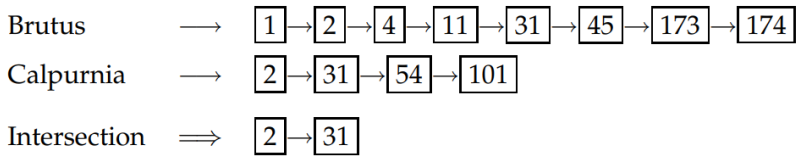
Doc 4 july new home sales rise

Exercise 1.2

Consider these documents:

Doc 1 breakthrough drug for schizophrenia
Doc 2 new schizophrenia drug
Doc 3 new approach for treatment of schizophrenia
Doc 4 new hopes for schizophrenia patients

a. Draw the term-document incidence matrix for this document collection.



► **Figure 1.5** Intersecting the postings lists for Brutus and Calpurnia from Figure 1.3.

b. Draw the inverted index representation for this collection, as in Figure 1.3 .

Exercise 1.3

For the document collection shown in Exercise 1.2, what are the returned results for these queries:

a. schizophrenia AND drug
b. for AND NOT(drug OR approach)

Exercise 1.4

For the queries below, can we still run through the intersection in time $O(x + y)$, where x and y are the lengths of the postings lists for Brutus and Caesar? If not, what can we achieve?

a. Brutus AND NOT Caesar

b. Brutus OR NOT Caesar

Exercise 1.5

Extend the postings merge algorithm to arbitrary Boolean query formulas. What is its time complexity? For instance, consider:

c. (Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

Can we always merge in linear time? Linear in what? Can we do better than this?

Exercise 1.6

We can use distributive laws for AND and OR to rewrite queries.

a. Show how to rewrite the query in Exercise 1.5 into disjunctive normal form using the distributive laws.
b. Would the resulting query be more or less efficiently evaluated than the original form of this query?
c. Is this result true in general or does it depend on the words and the contents of the document collection?

Exercise 1.7

Recommend a query processing order for

d. (tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)

given the following postings list sizes:

Term	**Postings size**
Eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

Exercise 1.8

If the query is:

e. friends AND romans AND (NOT countrymen)

how could we use the frequency of countrymen in evaluating the best query evaluation order? In particular, propose a way of handling negation in determining the order of query processing.

Exercise 1.9

For a conjunctive query, is processing postings lists in order of size guaranteed to be optimal? Explain why it is, or give an example where it isn't.

Exercise 1.10

Write out a postings merge algorithm, in the style of Figure 1.6 (page 11), for an *x* OR *y* query.

Exercise 1.11

How should the Boolean query *x* AND NOT *y* be handled? Why is naive evaluation of this query normally very expensive?

Write out a postings merge algorithm that evaluates this query efficiently.

Exercise 1.12

Write a query using Westlaw syntax which would find any of the words professor, teacher, or lecturer in the same sentence as a form of the verb explain.

Exercise 1.13

Try using the Boolean search features on a couple of major web search engines. For instance, choose a word, such as burglar, and submit the queries (i) burglar, (ii) burglar AND burglar, and (iii) burglar OR burglar. Look at the estimated number of results and top hits. Do they make sense in terms of Boolean logic? Often they haven't for major search engines. Can you make sense of what is going on? What about if you try different words? For example, query for (i) knight, (ii) conquer, and then (iii) knight OR conquer. What bound should the number of results from the first two queries place on the third query? Is this bound observed?

Exercise 2.1

Are the following statements true or false?

a. In a Boolean retrieval system, stemming never lowers precision.
b. In a Boolean retrieval system, stemming never lowers recall.
c. Stemming increases the size of the vocabulary.
d. Stemming should be invoked at indexing time but not while processing a query.

Exercise 2.2

Suggest what normalized form should be used for these words (including the word itself as a possibility):

a. 'Cos
b. Shi'ite
c. cont'd
d. Hawai'i
e. O'Rourke

Exercise 2.3

The following pairs of words are stemmed to the same form by the Porter stemmer. Which pairs would you argue shouldn't be conflated. Give your reasoning.

a. abandon/abandonment
b. absorbency/absorbent
c. marketing/markets
d. university/universe
e. volume/volumes

Exercise 2.4

For the Porter stemmer rule group shown in (2.1):

a. What is the purpose of including an identity rule such as SS → SS?

b. Applying just this rule group, what will the following words be stemmed to?

circus *canaries* *boss*

c. What rule should be added to correctly stem *pony*?

d. The stemming for *ponies* and *pony* might seem strange. Does it have a deleterious effect on retrieval? Why or why not?

Exercise 2.5

Why are skip pointers not useful for queries of the form *x* OR *y*?

Exercise 2.6

We have a two-word query. For one term the postings list consists of the following 16 entries:

[4,6,10,12,14,16,18,20,22,32,47,81,120,122,157,180]

and for the other it is the one entry postings list:

[47].

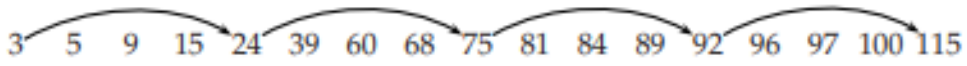
Work out how many comparisons would be done to intersect the two postings lists with the following two strategies. Briefly justify your answers:

a. Using standard postings lists

b. Using postings lists stored with skip pointers, with a skip length of $\sqrt{P}$, as suggested in Section 2.3.

Exercise 2.7

Consider a postings intersection between this postings list, with skip pointers:



and the following intermediate result postings list (which hence has no skip pointers):

3 5 89 95 97 99 100 101

Trace through the postings intersection algorithm in Figure 2.10

```
INTERSECTWITHSKIPS(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then if hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
 9                     then while hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
10                          do p1 ← skip(p1)
11                     else p1 ← next(p1)
12                else if hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
13                     then while hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
14                          do p2 ← skip(p2)
15                     else p2 ← next(p2)
16  return answer
```

► **Figure 2.10** Postings lists intersection with skip pointers.

a. How often is a skip pointer followed (i.e., p_1 is advanced to $skip(p_1)$)?

b. How many postings comparisons will be made by this algorithm while intersecting the two lists?

c. How many postings comparisons would be made if the postings lists are intersected without the use of skip pointers?

Exercise 2.8

Assume a biword index. Give an example of a document which will be returned for a query of New York University but is actually a false positive which should not be returned.

Exercise 2.9

Shown below is a portion of a positional index in the format: term: doc1: (position1,
position2, . . .); doc2: (position1, position2, . . .); etc.

angels: 2: (36,174,252,651); 4: (12,22,102,432); 7: (17);
fools: 2: (1,17,74,222); 4: (8,78,108,458); 7: (3,13,23,193);
fear: 2: (87,704,722,901); 4: (13,43,113,433); 7: (18,328,528);
in: 2: (3,37,76,444,851); 4: (10,20,110,470,500); 7: (5,15,25,195);
rush: 2: (2,66,194,321,702); 4: (9,69,149,429,569); 7: (4,14,404);
to: 2: (47,86,234,999); 4: (14,24,774,944); 7: (199,319,599,709);
tread: 2: (57,94,333); 4: (15,35,155); 7: (20,320);
where: 2: (67,124,393,1001); 4: (11,41,101,421,431); 7: (16,36,736);

Which document(s) if any match each of the following queries, where each expression within quotes is a phrase query?

a. “fools rush in”

b. “fools rush in” AND “angels fear to tread”

Exercise 2.10

Consider the following fragment of a positional index with the format:
word: document: (position, position, . . .); document: (position, . . .)

. . .
Gates: 1: (3); 2: (6); 3: (2,17); 4: (1);
IBM: 4: (3); 7: (14);
Microsoft: 1: (1); 2: (1,21); 3: (3); 5: (16,22,51);

The /k operator, word1 /k word2 finds occurrences of word1 within k words of word2 (on either side), where k is a positive integer argument. Thus k = 1 demands that word1 be adjacent to word2.

a. Describe the set of documents that satisfy the query Gates /2 Microsoft.

b. Describe each set of values for k for which the query Gates /k Microsoft returns a different set of documents as the answer

Exercise 3.1

In the permuterm index, each permuterm vocabulary term points to the original vocabulary term(s) from which it was derived. How many original vocabulary terms can there be in the postings list of a permuterm vocabular term?

Exercise 3.2

Write down the entries in the permuterm index dictionary that are generated by the term mama.

Exercise 3.3

If you wanted to search for s*ng in a permuterm wildcard index, what key(s) would one do the lookup on?

Exercise 3.4

Refer to Figure 3.4; it is pointed out in the caption that the vocabulary terms in the postings are lexicographically ordered. Why is this ordering useful?

Exercise 3.5

Consider again the query fi*mo*er from Section 3.2.1. What Boolean query on a bigram index would be generated for this query? Can you think of a term that matches the permuterm query in Section 3.2.1, but does not satisfy this Boolean query?

Exercise 3.6

Give an example of a sentence that falsely matches the wildcard query mon*h if the search were to simply use a conjunction of bigrams.

Exercise 3.7

If $|s_i|$ denotes the length of string s^i, show that the edit distance between s^1 and s^2 is never more than $\max\{|s_1|, |s_2|\}$.

Exercise 3.8

Compute the edit distance between paris and alice. Write down the 5 × 5 array of distances between all prefixes as computed by the algorithm in Figure 3.5.

```
EDITDISTANCE(s1, s2)
 1  int m[i, j] = 0
 2  for i ← 1 to |s1|
 3  do m[i, 0] = i
 4  for j ← 1 to |s2|
 5  do m[0, j] = j
 6  for i ← 1 to |s1|
 7  do for j ← 1 to |s2|
 8     do m[i, j] = min{m[i − 1, j − 1] + if (s1[i] = s2[j]) then 0 else 1fi,
 9                      m[i − 1, j] + 1,
10                      m[i, j − 1] + 1}
11  return m[|s1|, |s2|]
```

► **Figure 3.5** Dynamic programming algorithm for computing the edit distance between strings s_1 and s_2.

Exercise 3.10

Compute the Jaccard coefficients between the query bord and each of the terms in Figure 3.7 that contain the bigram or.

Exercise 3.11

Consider the four-term query catched in the rye and suppose that each of the query terms has five alternative terms suggested by isolated-term correction. How many possible corrected phrases must we consider if we do not trim the space of corrected phrases, but instead try all six variants for each of the terms?

Exercise 3.14

Find two differently spelled proper nouns whose soundex codes are the same.

Exercise 3.15

Find two phonetically similar proper nouns whose soundex codes are different.

Exercise 4.3

For n = 15 splits, r = 10 segments, and j = 3 term partitions, how long would distributed index creation take for Reuters-RCV1 in a MapReduce architecture? Base your assumptions about cluster machines on Table 4.1.

Q. Explain map reduce paradigm

Q. Explain the process of dynamic indexing along with its pseudo code of logarithmic merging