CSC 583: Assignment 1

January 28, 2017 [redacted]

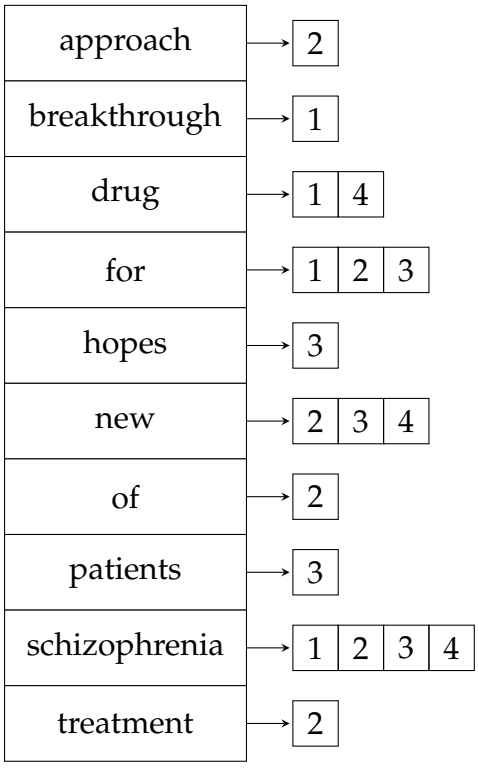
Problem 1

Consider these documents:
Doc 1 breakthrough drug for schizophrenia
Doc 2 new approach for treatment of schizophrenia
Doc 3 new hopes for schizophrenia patients
Doc 4 new schizophrenia drug

1. Draw the term-document incidence matrix for the document collection.

	Doc1	**Doc2**	**Doc3**	**Doc4**
approach	0	1	0	0
breakthrough	1	0	0	0
drug	1	0	0	1
for	1	1	1	0
hopes	0	0	1	0
new	0	1	1	1
of	0	1	0	0
patients	0	0	1	0
schizophrenia	1	1	1	1
treatment	0	1	0	0

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



3. What are the returned results for these queries:

   (a) schizophrenia `AND` drug

   Doc1, **Doc4**

   (b) for `AND NOT` (drug `OR` approach)

   (drug `OR` approach): **Doc1**, **Doc2**, **Doc4**.

   for `AND NOT` (drug `OR` approach):

   Doc3

Problem 2

1. Write out a postings merge algorithm, in the style of Figure 1.6 in IIR, for an x `OR` y query.

```
1:  procedure UNION(p1, p2)
2:      answer ← ⟨⟩
3:      while p1 ≠ NIL and p2 ≠ NIL do
4:          if docID(p1) = docID(p2) then
5:              ADD(answer, docID(p1))
6:              p1 ← next(p1)
7:              p2 ← next(p2)
8:          else if docID(p1) < docID(p2) then
9:              ADD(answer, docID(p1))
10:             p1 ← next(p1)
11:         else
12:             ADD(answer, docID(p2))
13:             p2 ← next(p2)
14:     while p1 ≠ NIL do
15:         ADD(answer, docID(p1))
16:         p1 ← next(p1)
17:     while p2 ≠ NIL do
18:         ADD(answer, docID(p2))
19:         p2 ← next(p2)
```

2. Write out a postings merge algorithm, in the style of Figure 1.6 in IIR, for an x `AND NOT` y query.

```
1:  procedure MINUS(p1, p2)
2:      answer ← ⟨⟩
3:      while p1 ≠ NIL and p2 ≠ NIL do
4:          if docID(p1) = docID(p2) then
5:              p1 ← next(p1)
6:              p2 ← next(p2)
7:          else if docID(p1) < docID(p2) then
8:              ADD(answer, docID(p1))
9:              p1 ← next(p1)
10:         else
11:             p2 ← next(p2)
12:     while p1 ≠ NIL do
13:         ADD(answer, docID(p1))
14:         p1 ← next(p1)
```

Problem 3

Recommend a query processing order for:
(tangerine `OR` trees) `AND` (marmalade `OR` skies) `AND` (kaleidoscope `OR` eyes)
given the following postings list sizes:

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

The estimated (worst-case) lengths for the results of the `OR` operations are:

- (tangerine `OR` trees): $87009 + 316812 = 403821$
- (marmalade `OR` skies): $107913 + 271658 = 379571$
- (kaleidoscope `OR` eyes): $46653 + 213312 = 259965$

These operations must be done first (in any order), and then the `AND` operations should be applied on the results in ascending order of size. Therefore, the recommended order is:

(1) $\leftarrow$ tangerine `OR` trees
(2) $\leftarrow$ marmalade `OR` skies
(3) $\leftarrow$ kaleidoscope `OR` eyes
(4) $\leftarrow$ (3) `AND` (2)
return (4) `AND` (1)

Problem 4

How should the Boolean query x `OR NOT` y be handled? Why is the naive evaluation of this query normally very expensive? Write out a postings merge algorithm that evaluates this query efficiently.

Suppose that the list of all documents is long, with length D.

The naive evaluation would first compute `NOT` y, and then x `OR` (`NOT` y):

1: *result* ← MINUS(*documents*, y)
2: **return** UNION(x, *result*)

This evaluation would require one scan of the document list to compute `NOT` y. Most of the time, the postings list for y is much shorter than the list of documents, so the result of `NOT` y is most of the document list. Computing the union of x with `NOT` y requires a complete scan of the result of `NOT` y, so there is a second scan of length D. In total, the list of all documents is scanned twice. This makes the query expensive.

To be more efficient, first apply DeMorgan's Law to the query:

x `OR NOT` y = `NOT` (`NOT` (x `OR NOT` y)) = `NOT` (`NOT` x `AND` y) = `NOT` (y `AND NOT` x)

Then the algorithm is:

1: *result* ← MINUS(y, x)
2: **return** MINUS(*documents*, *result*)

Computing y `AND NOT` x is linear in the lengths of postings for x and y, which are usually much shorter than D. Then computing `NOT` the result requires just one scan of the documents list. So in total, this algorithm requires only one scan of the list of all documents, which is more efficient.

Problem 5

See program and `README` in `assg1-prog.tar`.