Boolean Retrieval - IR weekly report

hello underworld

29th April 2021

1 Some Initial Concepts

1.1 Token, Type, Term

A token is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing. A type is the class of all tokens containing the same character sequence. A term is a (perhaps nor- malized) type that is included in the IR system's dictionary. The set of index terms could be entirely distinct from the tokens, for instance, they could be semantic identifiers in a taxonomy, but in practice in modern IR systems they are strongly related to the tokens in the document.

1.2 What's the Index granularity

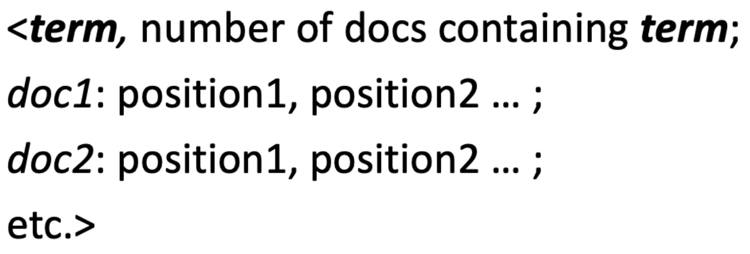
In the process of getting the vocabulary term, we have to decide the scale of each document unit, also named the index granularity.It becomes clear that there is a precision/recall tradeoff here. If the units get too small, we are likely to miss important passages because terms were distributed over several mini-documents, while if units are too large we tend to get spurious matches and the relevant information is hard for the user to find.

1.3 What's the Phrase Query

On my perspective, phrase query requires targeted terms appear closely, within k words of place(k = 0, 1,2...).A little example for Phrase Query in the book:We would like to be able to pose a query such as Stanford University by treating it as a phrase so that a sentence in a document like 'The inventor Stanford Ovshinsky never went to university' is not a match. Here are two solutions for this problem, including positional index and biword index.

1.4 What's the Positional Index

Actually, I think this is a kind of inverted index. Here, for each term in the vocabulary, we store postings of the form docID: (position1, position2, . . .), as shown in following pattern, where each position is a token index in the document. Each posting will also usually record the term frequency



<***term,*** number of docs containing ***term***;

doc1: position1, position2 … ;

doc2: position1, position2 … ;

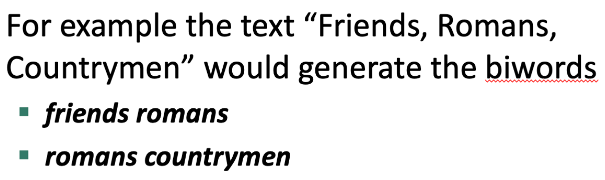
etc.>

Figure 1: pattern of positional index

The proximity search could satisfy the positional index to do the phrase query. In the next part, I will introduce the the proximity search algorithm by the positional index

1.5 What's the Biword Index

In this model, we treat each of these biwords as a vocabulary term. Being able to process two-word phrase queries is immediate. Longer phrases can be processed by breaking them down. A simple example in book:


For example the text "Friends, Romans, Countrymen" would generate the biwords

- ***friends romans***
- ***romans countrymen***

Figure 2: pattern of biword index

1.6 What's the Skip List

In the last weekly report, we walk through the two postings lists simultaneously, in time linear in the total number of postings entries. If the list lengths are m and n, the intersec- tion takes O(m + n) operations. Can we do better than this? One way to do this is to use the skip list. By using skip list, we could reduce the times of comparing two posting lists. Here is an example of the usage of skip list

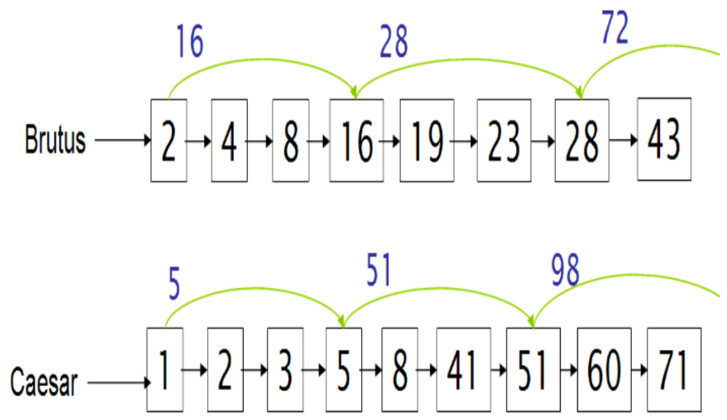


Figure 3: example of skip list

Setting the skip pointers, postings lists with skip pointers. The postings intersection can use a skip pointer when the end point is still less than the item on the other list.

2 Some processes

2.1 How to get the term vocabulary?

2.1.1 Obtainning the character sequence

Actually, this process is a little difficult, because I don't really understand the systems of other languages except the English and Chinese.

2.1.2 Choosing the document unit

According to the preference of the users, different sizes of document unit should be set.

2.1.3 Tokenization

Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called tokens, perhaps at the same time throwing away certain characters, such as punctuation.

2.1.4 Dropping common terms: stop words

Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called stop words, like 'of' and 'the'. Using a stop list significantly reduces the number of postings that a system has to store.

2.1.5 Normalization (equivalence classing of terms)

Token normalization is the process of canonicalizing tokens so that matches occur despite superficial differences in the character sequences of the to- kens.4 The most standard way to normalize is to implicitly create equivalence classes, which are normally named after one member of the set.

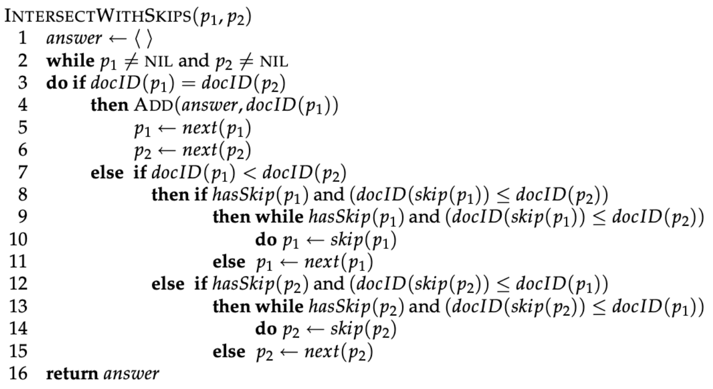
2.1.6 Stemming and Lemmatization

Actually, I think this process is relevant to the process of normalization, aimed at to reduce the number of the terms in vocabulary. Stemming and lemmatization is a process of reducing inflectional forms and sometimes derivationally related forms of a word to a common base form. But actually, these two processes also reduce the precison of the query to some extent.

3 Some algorithms

3.1 Intersection with skip lists

In section 1.6, we get a better way to intersect two posting lists, with skip list. Here, we gave the concrete algorithm of this process:



```
IntersectWithSkips($p_1, p_2$)
answer ← ⟨ ⟩
while $p_1 \neq$ NIL and $p_2 \neq$ NIL
do if $docID(p_1) = docID(p_2)$
     then Add($answer, docID(p_1)$)
          $p_1 \leftarrow next(p_1)$
          $p_2 \leftarrow next(p_2)$
     else if $docID(p_1) < docID(p_2)$
            then if $hasSkip(p_1)$ and $(docID(skip(p_1)) \leq docID(p_2))$
                   then while $hasSkip(p_1)$ and $(docID(skip(p_1)) \leq docID(p_2))$
                          do $p_1 \leftarrow skip(p_1)$
                   else $p_1 \leftarrow next(p_1)$
            else if $hasSkip(p_2)$ and $(docID(skip(p_2)) \leq docID(p_1))$
                   then while $hasSkip(p_2)$ and $(docID(skip(p_2)) \leq docID(p_1))$
                          do $p_2 \leftarrow skip(p_2)$
                   else $p_2 \leftarrow next(p_2)$
return answer
```

Figure 4: algorithm of intersection with skiplist

Actually, the skip span is an important parameter in this algorithm. A simple heuristic for placing skips, which has been found to work well in practice, is that for a postings list of length P, use √P evenly-spaced skip pointers.

3.2 Proximity Intersection

With the positional index, we could use the proximity search to do the phrase query. Here is this algorithm for this process.

```
PositionalIntersect($p_1, p_2, k$)
answer $\leftarrow \langle\, \rangle$
while $p_1 \neq$ NIL and $p_2 \neq$ NIL
do if $docID(p_1) = docID(p_2)$
      then $l \leftarrow \langle\, \rangle$
           $pp_1 \leftarrow positions(p_1)$
           $pp_2 \leftarrow positions(p_2)$
           while $pp_1 \neq$ NIL
           do while $pp_2 \neq$ NIL
                 do if $|pos(pp_1) - pos(pp_2)| \leq k$
                       then Add($l, pos(pp_2)$)
                       else if $pos(pp_2) > pos(pp_1)$
                               then break
                    $pp_2 \leftarrow next(pp_2)$
                 while $l \neq \langle\, \rangle$ and $|l[0] - pos(pp_1)| > k$
                 do Delete($l[0]$)
                 for each $ps \in l$
                 do Add($answer, \langle docID(p_1), pos(pp_1), ps \rangle$)
                 $pp_1 \leftarrow next(pp_1)$
           $p_1 \leftarrow next(p_1)$
           $p_2 \leftarrow next(p_2)$
      else if $docID(p_1) < docID(p_2)$
              then $p_1 \leftarrow next(p_1)$
              else $p_2 \leftarrow next(p_2)$
return answer
```

Figure 5: algorithm of intersection with positional index