Web Search Engines – Problem Set 1

Narasimman Sairam (N13296703)

1.A. Processing the posting list in the order of their size ***is not always optimal***. This can occur when in a set of posting list A,B,C,D,E in the order of their size and R being the intermediate result list have R ^ E = {} // Empty list.

Eg: Say there are three lists with posting list size as follows

A = 100, B = 200, C = 300 and A^B = 90, A^C = 0

The algorithm given in the book will have to walk through all the three lists. So, total number of comparison operations would be *100 + 200 + 300 = 600;*

But, if A and C were to be compared in the first iteration and then with B, the number of steps would be:
100 + 300 + 0 = 400; (because the intermediate result list with A^C will return an empty set and there is no need to process any list further)

B. Assuming that the hash table is pre-populated at the time of indexing and we also record the length of the posting list for each word, we can modify the algorithm to use the hash table as follows:

1. Given two postings p1 and p2, find the shorter list. (as it is in the order of size of posting list, it should generally be p1)
2. Now, for every docID in the shorter list (or intermediate result list) and word of the second(or larger) list, find if there exists a key in the hashtable of the for <word2,docID>. word2 - word of the second list and docID is for every docID of the shorter list. If there exists a key, then add that to the result list.

```
INTERSECT(p1, p2, word2)
  answer <- ()
  shorterlist = len(p1) < len(p2)? p1: p2
  while (shorterlist != null) {
    if(hashMap.containsKey(<word2,docID(shorterlist)>)) {
        add(answer, docID(shorterlist))
    }
    shorterlist <- next(shorterlist)
  }
return answer
```

So, the complexity of the algorithm would be the length of the shorterlist as looking up the hashtable is constant operation. if p1 = x and p2 = y then it's x.

2. A pre-processor should be able to recognize phone numbers and perform normalization. So, the content (phone number) that is being indexed will be in a single standard format making the job of the tokenizer easier.

There are several challenges in recognizing and formatting of phone numbers.
Phone numbers can be of different formats:
(212)998-3123
(212)998---3123
212.998.3123
l-925-225-3000
l 925 225 3000
etc

1. Identifying that the input is a phone number by checking the following demarcations and verify it against the standard telephone number patterns.
   a. hyphens, dots and brackets
   b. Phone numbers variously use spaces, periods, hyphens, brackets, and even slashes to group digits in various ways, often not consistently even within one country. Additionally, phone numbers may include international or

national long distance codes, or just show a local number, and there may or may not be explicit indication of this via other marks such as brackets and plus signs.

c. Missing information. Phone numbers might not have country code/area code, etc. in which case the the format analyzer can try to predict the context/location to find the missing information. When identifying each token, several characteristics of the token can be stored.

2. Identifying the phone number in a document using some additional information or emphasis that is present in the document structure.
   a. For eg: In case of HTML documents, Search Engine should not ignore the difference between content and 'markup'.

   b. Format analysis should be performed on the document and identify any HTML tags, tags that contain the keywords in the attributes/tagnames. eg: 123-456-3434 or Call me

3. Variant coding of information of a certain semantic type like local syntax for phone numbers should be handled. Phone numbers can appear in many formats and can vary from one country to the other. Any search Engine dealing with documents that contain text from different countries or written according to different stylistic conventions has to be prepared to deal with typographical differences.

4. If the search engine supports multiple document formats, preparation of the document for normalization is challenging. Identifying the semantic information about the content of the document will vary from one document to the other. Some document's semantic structure may not be publicly available.

3. Crawler and Refresh Strategy:

3.1 The probability of a web page getting changed follows a Poisson distribution. So, the formula for age is:

$$Age(\lambda, t) = lim(0 \dashrightarrow t)\ \lambda e^{-\lambda x}(t - x)\ dx$$

The second derivative of this expression gives the correct value for the cost of not crawling a web page. Similarly, we can get an expression of the age of a web page with respect to the amount of change $\Delta\sigma$. The random variable in the above case is the expression (t-x) which gives an approximate value of age. Similarly we can represent by $\Delta\sigma$ the random variable of amount of change of a website w.r.t time.

Then the amount of change can be given as

$$Change(\lambda, \sigma, t) = lim(0 \rightarrow t)\ \lambda e^{-\lambda x}\Delta\sigma dx$$

The first order derivative of the above equation is:

$$d/dx\ lim(0 \rightarrow t)\ \lambda e^{-\lambda x}\Delta\sigma dx = \lambda e^{-\lambda x}\Delta\sigma$$

Minimizing the equation(3) w.r.t time gives us the optimal frequency with which we have to crawl a particular web site.

3.2 ***Information for better crawl performance***:
The crawler should collect the following information while crawling a webpage

1. The difference between the sizes of the web page between the previous crawl and current crawl.
2. The average refresh frequency of the web page
3. Statistics such as whether the webpage is a portal with multiple links to other websites.

This helps us figure out how frequently the index of the webpage should be updated and also the optimal crawling strategy. Using these data, we will be able to identify the factors that help in minimizing the cost function in deciding the refresh rate of a web page.

The crawler should penalize the web pages that change too often. The optimal method for keeping average freshness high includes ignoring the pages that change too often and considering the average age of the page and keeping it low.

3.3 When we refresh page A and there is a new link to page B, then B will be crawled. But, B will not be crawled until A is being refreshed. To handle this, we can look at A's header and if it has been modified since last crawl, we refresh it immediately. So, apart from looking at the last modified time in the header of the page, we can consider the number of outbound links that are present on a web page and if it is above a considerable threshold, we can tweak our refresh strategy to refresh more often than average. So, we would have to differentiate between the ephemeral content and persistent content where ephemeral content is given more weightage and hence refreshed more often.