Problem 1 - Boolean Retrieval

Consider the following fragment of a positional index with format DocId: <Position, ...>.

\[
\begin{align*}
\text{abc: } & 1: \langle 2, 5, 100 \rangle; 2: \langle 34, 82 \rangle; 5: \langle 13, 14 \rangle; 10: \langle 1, 216 \rangle; 214: \langle 325, 1221 \rangle \\
\text{xyz: } & 1: \langle 25, 56 \rangle; 5: \langle 12, 15 \rangle; 191: \langle 8, 25, 31, 52 \rangle; 214: \langle 322, 1222 \rangle
\end{align*}
\]

A phrase query “word1 word2” retrieves the occurrences where word1 is immediately followed by word2. A /\(k\) query “word1 /\(k\) word2” (\(k\) is a positive integer) retrieves the occurrences of word1 within \(k\) words of word2 on either side. For example, \(k = 1\) demands that word1 be adjacent to word2, but word1 may come either before or after word2.

1. List the set of documents and the matching positions satisfying the following queries using the format DocId: \(<\text{Position}_{\text{abc}}, \text{Position}_{\text{xyz}}>, \ldots\>:
   
   (a) “abc xyz”
   (b) “abc /2 xyz”

2. A stream \(s\) is a data sequence that supports the following operations:
   
   - \(s\).peek(): returns the element at the head of the sequence, or null if the sequence does not have any element left.
   - \(s\).next(): advances to the next element, which becomes the new head of the stream.
Consider the query “word1 /k word2”. In each document \(d\), we want to find all positions of the two terms in the document that satisfy the query. Let the streams \(p\) and \(q\) contain the positions of word1 and word2 in \(d\). (Example: For the query “abc /2 xyz” and \(d = 1\), the stream \(p\) contains 2, 5, and 100, while the stream \(q\) contains 25 and 56.) To support streaming, we need to use buffers to temporarily store some positions. How big must the total size of the buffers be if the maximum number of positions per term is \(n\)? Give a brief justification.

3. We can use the \(/k\) operator (and filtering) to answer phrase queries, but there is a more efficient algorithm that does not require additional buffer space. Consider the phrase query “word1 word2”. Given two streams \(p\) and \(q\) of positions of the two query terms in the same document, describe a streaming algorithm (e.g., by writing pseudocode) that prints all possible positions that satisfy the phrase query for that document without using additional buffer space.

4. How would you modify the positional index to support queries that demand the terms to be in the same sentence? You can assume that the parsing step is able to identify the sentences in a document. Write down an example of the modified postings list.

**Problem 2 - Index Compression**

1. Suppose the vocabulary for your inverted index consists of the following 6 terms:
   
   elite
elope
ellipse
eloquent
eligible
elongate

Assume that the dictionary data structure used for this index stores the sorted list of terms using dictionary-as-a-string storage with front coding and a block size of 3. Show the resulting storage of the above vocabulary of 6 terms. Use the special symbols * and ♦ as used in the discussion on front coding in Chapter 5.

2. Consider a postings list:

   \(<4, 15, 62, 63, 265, 268, 270, 500>\)

   with the corresponding list of document gaps:

   \(<4, 11, 47, 1, 202, 3, 2, 230>\)

   Assume that the length of the postings list is stored separately, so the system knows when a postings list is complete. We are going to contrast the size and speed tradeoffs of using variable byte encoding with gamma encoding.

   (a) Encode the document gaps using variable byte encoding.
(b) Encode the document gaps using gamma encoding.

(c) Assuming a slow ethernet connection (1 million bytes / second) to network attached storage, how long would it take to transfer just the given postings list stored in variable byte encoding (of document gaps) to memory? How about the gamma encoded list? (Answer in $\mu s = 10^{-6}$ seconds.)

(d) Now, assume that to decode a gamma encoded postings list into usable numbers takes a constant time of $n/2 \mu s$ where $n$ is the number of elements in the postings list. This adds a processing time to storing a postings list in gamma encoded format. For the given postings list, what is the upper limit on how much processing time (in $\mu s$ per list element) variable byte encoding can take, in order to be faster than (or equally fast as) gamma encoding?

**Problem 3 - Tolerant Retrieval**

Consider the following text:

```
peter piper picks papers
```

1. How many character bigram dictionary entries and character bigram posting entries are generated by indexing the bigrams in the terms in the text above?

2. How would the wild-card query $p^*er$ be best expressed as an AND query using the bigram index over the text above?

3. For the most efficient query in part 2, how many postings must be traversed?

4. How many permuterm dictionary entries are generated by indexing the words in the text above?

5. How would the wild-card query $p^*er$ be expressed for lookup in the permuterm index?

**Problem 4 - Vector Space Model**

Consider the following documents:

```
A: to be not to be not
B: it was not to be
```

1. Write down the term frequency of each term in each document.

2. Write down the number of occurrences of each word bigram in each document (ignoring the begin-of-document and end-of-document tokens).

3. Consider a vector space where each dimension is a word bigram. For each document, write down the normalized vector of raw bigram counts (no logarithm or idf) using the format \{bigram: value, bigram: value, \ldots\}. Use Euclidean normalization.
4. Now consider the tf-idf weighting:

\[ w_{t,d} = \text{tf}_{t,d} \times \log_{10}(N/df_t) \]

For each document, write down the normalized vector of tf-idf weights where each dimension is a word bigram.

5. For each of the weighting schemes in parts 3 and 4, compute the cosine similarity between documents A and B.