COMP6714 ASSIGNMENT 1 SAMPLE SOLUTION

Q1.

(1) omitted.
(2) Bing obviously expand the query to include accents/umlets.
(3) (a) Neuro-linguistic: "Neuro-linguistic"
(b) otta: "otta",
(c) Neuro-linguistic AND otta: "Neuro-linguistic" "otta"
(d) Neuro-linguistic OR otta: "Neuro-linguistic" OR "otta"
(e) Neuro-linguistic /1 otta: "Neuro-linguistic otta" OR "otta Neuro-linguistic"
(f) bugle: "bugle"
(g) bugle bugle: "bugle" "bugle"
(h) bugle bugle bugle: "bugle" "bugle" "bugle"

No. Let (estimated) query results of A and B be $|A|$ and $|B|$, respectively. Then $|A \cap B| \in [0, \min(|A|, |B|)]$. It is not observed from google's estimates. In fact, if we know the entire collection has $|C|$ documents in total, we can improve the lower bound estimate as $\max(0, |A| + |B| - |C|)$.

Q2. (*20 marks*)

Rearrange the lists as follows

```
fools : 2: <1,17,74,222>;      4: <8,78,108,458>;      7: <3,13,23,193>;
rush  : 2: <2,66,194,321,702>; 4: <9,69,149,429,569>;  7: <4,14,404>;
in    : 2: <3,37,76,444,851>;  4: <10,20,110,470,500>; 7: <5,15,25,195>;

angels: 2: <36,174,252,651>;   4: <12,22,102,432>;     7: <17>;
fear  : 2: <87,704,722,901>;   4: <13,43,113,433>;     7: <18,328,528>;
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>;
```

(1) (a) "fools rush in": doc=2, pos=1–3; doc=4, pos=8–10, doc=7, pos=3–5, 13–15.
(b) "fools rush in" AND "angels fear to tread": the second condition matches docs 4 (pos=12–15) only. So the final results is doc 4, and two match regions: 8–10 and 12–15.
(2) both `where` and `in` appear in the doc=7, pos=15.

Q3.

(1) The algorithm scans both input lists only once, with the help of a "buffer" l, whose max size is k. In the worst case, in each iteration pp_1 moves to the next position,

all items in l are deleted (because they are far from the current pp_1). Hence, the complexity is $O(|p_1| + |p_2| + |p_1| \cdot k)$.[1]

Note that it might be tempting to think about the case when every thing in one list joins with everything in the other list. However, since the positions are all distinct, the maximum positions that can be joined (i.e., within distance of k) is bounded by k. This will give us the same bound as above.

Rather than using a linear search to remove invalid items in l, we can use a binary search as the items in l are in ascending order. However, for binary search to pay off, we need to avoid linearly invalidating those items. Hence, we could use a circular array of size $2k$ and keep two pointers as its head and tail. This brings down the cost to $O(|p_1| + |p_2| + |p_1| \cdot \log k)$.

(2) Assume each document has been divided into a set of paragraph, and each paragraph divided into a set of sentences. We shall record the position of a term in a document in the following scheme:

`paragraphid.sentenceid.position`

where the `paragraphid` is the paragraph id within the document, and the `sentenceid` is the sentence id within the paragraph, and the `position` is the position (of the term) in the *document* (not in the sentence).

- For `/k` queries, we just extract the `position` and process as usual.
- For `/S` queries, we just extract the `sentenceid` and process as usual.
- For `/P` queries, we just extract the `paragraphid` and process as usual.

There is another possible modification. We include two special tokens, one corresponding to each sentence end position and the other to each paragraph end position. In order to answer, e.g., query Q with `/S` operator, we just perform list intersection of keywords in Q and the inverted list for sentence-end. Some changes are needed so that whenever we encounter another sentence end, we output the current "valid" occurrences of keywords in Q (if any), and then force their cursors to move to the first position beyond the current sentence end. This method is also known as the *extent list approach* (c.f., Chap 5.3.4 in [CMS09]).

Q4.

Immediate merge. Always one sub-index.

No merge. Let $n \coloneqq \lceil \frac{|C|}{M} \rceil$. No merge will create n sub-indexes.

Logarithmic merge. Notice the 1-to-1 correspondence between the sub-indexes and the binary format of the number of sub-index created.

Therefore, the number of indexes is $f(n)$, where $f(n)$ gives the number of `1` in the binary representation of n. This is known as the *Hamming weight*[2]. Obviously, $f(n) = O(\log \lceil \frac{|C|}{M} \rceil)$.

---

[1] $|p_i|$ denotes the length of the inverted list p_i.

[2] `https://en.wikipedia.org/wiki/Hamming_weight`