

COMP6714 ASSIGNMENT 1

MINGLANG XIE z5228006

Q1

Algorithm 1: $Q1(p_1, p_2, p_s)$

```
1 answer  $\leftarrow \emptyset$ 
2 while  $p_1 \neq \text{nil} \wedge p_2 \neq \text{nil}$  do
3   if  $\text{docID}(p_1) = \text{docID}(p_2)$  then
4      $l \leftarrow []$ 
5      $\text{DID} \leftarrow \text{docID}(p_1)$ 
6      $pp_1 \leftarrow \text{positions}(p_1)$ ;  $pp_2 \leftarrow \text{positions}(p_2)$ ;  $pp_s \leftarrow p_s.\text{DID}$ 
7     while  $pp_1 \neq \text{nil}$  do
8        $\text{skipTo}(pp_s, \text{docid}, \text{pos}(pp_1))$ 
9       while  $pp_2 \neq \text{nil}$  do
10        if  $\text{pos}(pp_1) < \text{pos}(pp_2) \wedge \text{pos}(pp_2) < \text{pos}(pp_s)$  then
11           $\text{add}(l, \text{pos}(pp_2))$ 
12        else
13          if  $\text{pos}(pp_2) > \text{pos}(pp_1)$  then
14             $\text{break}$ ;
15           $pp_2 \leftarrow \text{next}(pp_2)$ 
16        while  $l \neq [] \wedge l[1] < \text{pos}(pp_s) \wedge l[1] > \text{pos}(pp_1)$  do
17           $\text{delete}(l[1])$ 
18        foreach  $ps \in l$  do
19           $\text{answer} \leftarrow \text{answer} \cup [\text{docID}(p_1), \text{pos}(pp_1), ps]$ 
20         $pp_1 \leftarrow \text{next}(pp_1)$ 
21       $p_1 \leftarrow \text{next}(p_1)$ ;  $p_2 \leftarrow \text{next}(p_2)$ 
22   else
23     if  $\text{docID}(p_1) < \text{docID}(p_2)$  then
24        $p_1 \leftarrow \text{next}(p_1)$ 
25     else
26        $p_2 \leftarrow \text{next}(p_2)$ 
27 return answer
```

Q2

(1)

Assume that t sub-indexes (each of M pages) will be created if one chooses the no-merge strategy. Thus, there are tM pages in total, if the logarithmic merge strategy is used, we are following two rules:

1. only create I_k and I_k has size $2^k M$
2. whenever we have $2^k I_k$, we need to merge it to form a I_{k+1}

Therefore, when the generation grow to g , we will get $[M, 2M, 4M, \dots, 2^k M]$

$$M, 2M, 4M, \dots, 2^k M = tM$$

$$M(1 + 2 + 4 + \dots + 2^k) = tM$$

$$\frac{(2^{k+1}) - 1}{2 - 1} = t$$

$$(2^{k+1}) = t + 1$$

$$k = \log_2(t + 1) - 1$$

when t is sufficient large, $k = \lceil \log_2(t) \rceil$. Therefore, if the logarithmic merge strategy is used, it will result in at most $\lceil \log_2(t) \rceil$ sub-indexes.

(2)

First, there are tM pages in total.

Second, we know the logarithmic merge strategy will result in at most $k = \lceil \log_2(t) \rceil$, which means after k generations, only the last generation in disk. Simply progress is:

Merge I_0 2^{k-1} times

Merge I_1 2^{k-2} times

...

Merge I_k one time

Hence, the total cost of the logarithmic merge is: $(2^0 * 2^{k-1} + 2^1 * 2^{k-2} + \dots + 2^{k-1} * 2^0)M = k * 2^{k-1} * M \approx \log_2 t * t * M$

The total I/O cost of the logarithmic merge is $O(\log_2 t * t * M)$

Q3

After the δ encoding, the compressed non-positional inverted list is

01000101 11110001 01110000 00110000 11110110 11011

Since this compressed list started with a 0, and 1 is the only number to get a 0 after the δ encoding. So, we can divide the compressed list as:

0 1000101 11110001 01110000 00110000 11110110 11011

Then, we get the first document ID which is 1, and remain:

1000101 11110001 01110000 00110000 11110110 11011

The next document ID will be decode(1000), which is 2, because the remain compressed list started with 10, and so $k_{dd} = 1$, and 0 after 10, which give $k_{dr} = 0$, use $k_{dr} = (k_d + 1) - 2^{\lceil \log_2(k_d + 1) \rceil}$, we have $0 = (k_d + 1) - 2^1$, given $k_d = 1$, next, 0 after 100 given $k_r = 0$. Therefore, $k_r = k - 2^{k_d} \Rightarrow 0 = k - 2 \Rightarrow k = 2$. Then remain:

101 11110001 01110000 00110000 11110110 11011

Continue this progress, we can divide the compressed list as below:

(10 1 11) (110 00 101) (110 00 000) (110 00 011) (110 11 011011)

$$\text{decode}(10\ 1\ 11) = 7$$

$$\text{decode}(110\ 00\ 101) = 13$$

$\text{decode}(110\ 00\ 000) = 8$
 $\text{decode}(110\ 00\ 011) = 11$
 $\text{decode}(110\ 11\ 011011) = 91$

Therefore, the document IDs in this list is $[1, 2, 7, 13, 8, 11, 91]$.

Q4

The line 21 in Figure 2 that causes the bug, loop happens when pivot is unchanged, because *pickTerm* selects the term with the maximal idf, if we have terms in order A, B, C. C is pTerm return by *findPivotTerm*(*terms*, θ), B have the same DID as pivot with the largest idf, and A have the smallest DID. Thus, *pickTerm*(*terms*[0..*pTerm*-1]) will always return term B, which will not change the posting. Therefore, the posting remains the same, resulted in the pivot is unchanged.

For example:

Threshold = 8

	A	B	C
UB	3	4	4
List	$\langle 1, 3 \rangle$	$\langle 1, 4 \rangle$	$\langle 1, 4 \rangle$
	$\langle 2, 3 \rangle$	$\langle 4, 2 \rangle$	$\langle 4, 3 \rangle$