COMP6714 ASSIGNMENT1

HUIYAO ZUO

Z5196480

Q1

answer ← ∅;

while p_1≠**nil** ∧ p_2≠**nil** do

if docID(p_1) = docID(p_2) **then**

l ← [ ];

pp_1 ← positions(p_1); pp_2 ← positions(p_2); $pp_\$$ ← positions($p_\$$) //get the p$ position

while pp_1≠ **nil do**

while pp_2 < pp_1 **or** $pp_\$$ < pp_2 **do** // when not p1 < p2 <p$

while pp_2 < pp_1 **do** //get next p2 until p2 > p1

p_2 ← next(p_2);

if pp_1 < $pp_\$$ < pp_2 **then** //not in the same sentence

pp_1 ← next(pp_1); //change p1 to next

while $pp_\$$< pp_2 **do** //after change p1 ,need to change p$ to next

$pp_\$$← next($pp_\$$); //jump to the p2 sentence

while pp_2≠ **nil do**

if pos(pp_2) − pos(pp_1) ≤ k **then** //do not need abs

add(l, pos(pp_2));

else //direct break

break;

pp_2 ← next(pp_2);

if pos(pp_2) > pos($pp_\$$) **then** //when not in same sentence ,break

break;

```
            while l/= [ ] ∧ l[1] < pos(pp1) do          //when p2 < p1,delete
                delete(l[1]);
            for each ps ∈ l do
                answer ← answer ∪ [docID(p1), pos(pp1), ps];
            pp1 ← next(pp1);
        if  pp$ < pp2 :     //when p2 already jump to next sentence ,only need continue ,the first new
                              while will deal with it
            continue
        else:               //normal jump
            p1 ← next(p1); p2 ← next(p2);
    else
        if docID(p1) < docID(p2) then
            p1 ← next(p1);
        else
            p2 ← next(p2);
return answer ;
```

Q2

(1) When the sub-indexes is t ,and each pages is M , when use logarithmic merge strategy, the merge indexes change is :

M ,2 M , 4 M , 8 M … t/4 , t/2 , t

And each part have a sub-index , and because t is not specific equal to 2^k , so need to count t + x = 2^k ,and there is k sub-indexes(if not count extra x part ,the result would be some part of t is not used)

So the result is $\lceil \log_2 t \rceil$

(2) for each M pages , there would be most merge $\log_2 t$ times ,and each I/O

cost is t*M , so the total cost is $O(t \cdot M \cdot \log_2 t)$

Q3

01000101 11110001 01110000 00110000 11110110 11011
The decode number is
0: d = 0 r = 0 dd = 0 dr = 0 equal 1
1000: d = 1 r = 0 dd = 1 dr = 0 equal 2
10111: d = 1 r = 1 dd = 2 dr = 3 equal 7
11000101: d = 2 r = 0 dd = 3 dr = 5 equal 13
11000000: d = 2 r = 0 dd = 3 dr = 0 equal 8
110000 11: d = 2 r = 0 dd = 3 dr = 3 equal 11
11011011011 : d = 2 r = 3 dd = 6 dr = 27 equal 91
So the document IDs in the list is : 1,2,7,8,11,13,91

Q4

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

Background: pickTerm() selects the term with the maximal idf .
The bug is the blue part code: when the maximal idf terms A(which is not terms[0]) also in the pivot. So the next A position is pivot. And if the value of A is not change ,so the new pTerm would not change , and A is not posting[0].so the posting[0] is still smaller than pivot , after the judge, the currentDOC is not change ,so the pivot still larger than the currentDOC , then the code need to pick one from terms to change to pivot ,and it would still pick A because it has the maximal idf score. Then A still in the positon pivot. And posting[0] not change , currentDOC and pTerm would not change, Then this code stuck in a infinite loop.
In order to fix it ,need to pick the terms in the [0, pTerm -1] which not in the pivot to change .
EXAMPLE

	A	B	C
UB	4	5	8
LIST	<1,3>	<1,4>	<1,4>
	<2,4>	<5,4>	<3,3>
			<5,4>

1. CurDOC = 0 , PSOTING = ABC , pivot = C => CurDOC = 1 value = 11
2. pickTerm() => C =><3,3> , smaller than value => new sort
3. A => <1,3> , B => <1,4> , C =><5,4> , value = 11 , PSOTING = ABC, pivot = C, CurDOC = 1 (pivot > CurDOC , PSOTING[0] != pivot)
4. Into bug code: Pick [A,B] to change ,pickTerm would pick B , B =><5,4>
5. A => <1,3> , B => <5,4> , C =><5,4> , value = 11 , PSOTING = ABC, pivot = C, CurDOC = 1(pivot > CurDOC , PSOTING[0] != pivot)
6. Into bug code: Pick [A, B] to change ,pickTerm would pick B , B =><5,4>
7. then repeat(5)(6).