COMP6714 Assignment 1

Hongxiao Jin (z5241154)

Question 1

Algorithm 1: Q1($p_1, p_2, p_\$$)

```
answer ← ∅;
while p1 ≠ nil ∧ p2 ≠ nil do
    if doc(p1) = doc(p2) then
        l ← [ ];
        pp1 ←positions(p1); pp2 ←positions(p2);
        while pp1 ≠ nil do
            end ← skipTo(p$,docID(pp1), pos(pp1))
            while pp2 ≠ nil do
                if pos(pp2) < end then
                    if pos(pp2) > pos(pp1) then
                        add(l,pos(pp2));
                else
                    break;
                pp2 ← next(pp2);
            while l ≠ [ ] ∧ l[1] < pos(pp1) do
                delete(l[1]);
            for each ps ∈ l do
                answer ← answer ∪ [docID(p1),pos(pp1), ps];
            pp1 ← next(pp1);
        p1 ← next(p1); p2 ← next(p2);
    else
        if docID(p1) < docID(p2) then
            p1 ← next(p1);
        else
            p2 ← next(p2);
return answer
```

Question 2

(1)

At first, the smallest($\mathbf{Z_0}$) is in memory. If $\mathbf{Z_0}$ gets to the upper bound of the memory, since the memory is full, it will be written into disk as $\mathbf{I_0}$. We can treat this $\mathbf{I_0}$ as **Generation 0**. However, if $\mathbf{I_0}$ already exists, $\mathbf{Z_0}$ and $\mathbf{I_0}$ will be merged into a new sub-index $\mathbf{Z_1}$. At this time, if there is no $\mathbf{I_1}$, $\mathbf{Z_1}$ will be written to disk as $\mathbf{I_1}$(**Generation 1**). While if $\mathbf{I_1}$ is in disk, $\mathbf{Z_1}$ will merge with $\mathbf{I_1}$ to form $\mathbf{Z_2}$ and so on.
From the algorithm of **logarithmic merge**, we can also know that this strategy maintain a series of indexes and each twice as large as the previous one.
Let **n** represent the number of sub-indexes.
- The 1^{st} layer can record **M** pages;
- The 2^{nd} layer can record **2M** pages;
- ......
- The n^{th} layer can record $\mathbf{2^{n-1} \cdot M}$ pages;
Thus, $M \cdot t = M \cdot (1 + 2 + 2^2 + 2^3 + \cdots + 2^{n-1})$
We can figure out $t = 2^n - 1$ and $n = log_2(t + 1)$.
Therefore, *logarithmic merge* will result in at most $\lceil \mathbf{log_2 t} \rceil$ sub-indexes.

(2)

First of all, writing all pages into disk will cost $\mathbf{M \cdot t}$.
Then, we should consider **merging cost**.
According to (**1**), the number of sub-indexes $\mathbf{n} = \lceil \mathbf{log_2 t} \rceil$, so sub-indexes can be record as $\mathbf{I_0, I_1, I_2, \ldots, I_{n-1}}$.
Since each index is twice as large as the previous one, we can conclude that
$2^1 \cdot I_0 = 2^0 \cdot I_1$
$2^2 \cdot I_0 = 2^1 \cdot I_1 = 2^0 \cdot I_2$
$2^3 \cdot I_0 = 2^2 \cdot I_1 = 2^1 \cdot I_2 = 2^0 \cdot I_3$
......
$2^{n-1} \cdot I_0 = 2^{n-2} \cdot I_1 = 2^{n-3} \cdot I_2 = \ldots = 2^1 \cdot I_{n-2} = 2^0 \cdot I_{n-1}$
Every time when one I_{n-1} Generating, two I_{n-2} need to be read. Before reading two I_{n-2}, they have to be written into the disk.
Therefore, except every I_0 is read once and every I_{n-1} is written once, the parameters generated in this process are read once and written once respectively.
The calculation process is as follows:

$$
\begin{aligned}
merge\_cost &= (2^{n-1} \cdot 2^0 + 2 * (2^{n-2} \cdot 2^1 + 2^{n-3} \cdot 2^2 M + \ldots + 2^1 \cdot 2^{n-2}) + 2^0 \cdot 2^{n-1}) \cdot M \\
&= M \cdot (2^n + 2^n \cdot (n-2)) \\
&= M \cdot 2^n \cdot (n-1) \\
&= M \cdot t \cdot (log_2 t - 1) \\
total\_cost &= write\_cost + merge\_cost \\
&= M \cdot t + M \cdot t \cdot (log_2 t - 1) \\
&= O(M \cdot t \cdot log_2 t)
\end{aligned}
$$

Therefore, the total I/O cost of the logarithmic merge is $O(t \cdot M \cdot log_2 t)$.

Question 3

Decode: 01000101 11110001 01110000 00110000 11110110 11011

0 : **1**

Let 10 0 0 represents number $\mathbf{k_1}$

$\because k_{1dd} = \lfloor log_2(k_{1d}+1) \rfloor = 1$

$k_{1dr} = (k_{1d}+1) - 2^{k_{1dd}} = 0$

$\therefore k_{1d} = 1$

$\because k_{1r} = k_1 - 2^{k_{1d}} = 0$

$\therefore k_1 = 2$

Thus, 10 0 0 represents **2**.

Let 10 1 11 represents number $\mathbf{k_2}$

$\because k_{2dd} = \lfloor log_2(k_{2d}+1) \rfloor = 1$

$k_{2dr} = (k_{2d}+1) - 2^{k_{2dd}} = 1$

$\therefore k_{2d} = 2$

$\because k_{2r} = k_2 - 2^{k_d} = 3$

$\therefore k_2 = 7$

Thus, 10 1 11 represents **7**.

Similarly,we can calculate:

110 00 101: **13**

110 00 000: **8**

110 00 011: **11**

110 11 011011: **91**

Therefore, the result of decoding is 1, 2, 7, 13, 8, 11, 91.

Since these numbers The document IDs are 1, 3, 10, 23, 31, 42, 133.

Question 4

(1)

Line 21 causes the bug in the algorithm in Figure 2.

The main problem is the **pickTerm**() method. According to the original paper, we can know that **pickTerm**() selects the term with the maximal idf. So if the idf of a term is maximum among all terms and is the **pTerm** in one cycle, this algorithm will end up in an infinite loop.

(2)

The table below is from lecture 5b. Infinite loop would appear in the process of this algorithm.

	A	B	C
UB	4	5	8
List	$\langle 1,3\rangle$ $\langle 2,4\rangle$ $\langle 7,1\rangle$	$\langle 1,4\rangle$ $\langle 2,1\rangle$ $\langle 7,2\rangle$ $\langle 8,5\rangle$ $\langle 9,2\rangle$ $\langle 11,5\rangle$	$\langle 1,4\rangle$ $\langle 2,2\rangle$ $\langle 5,1\rangle$ $\langle 7,7\rangle$ $\langle 10,1\rangle$ $\langle 11,8\rangle$

In the 1st cycle

$\theta = 0$, curDoc $= 0$

Sorted Term	A	B	C
Doc	1	1	1
Cumulative Upper Bound	4	9	17

- pTerm = A
- pivot = posting[0].DID = 1
- pivot > curDoc (**Line14**)
  - posting[0].DID = pivot (**Line15**)

    curDoc = 1, $\theta = 11$

    return (curDoc, posting)

In the 2nd cycle

$\theta = 11$, curDoc $= 1$

Sorted Term	A	B	C
Doc	1	1	1
Cumulative Upper Bound	4	9	17

- pTerm = C
- pivot = posting[2].DID = 1
- pivot = curDoc (**Line10**)

  aterm = pickTerm(terms[0..pTerm]) (**the idf of A is biggest, choose A**) posting[A].DID = 2

In the 3$^{\text{rd}}$ cycle

$\theta = 11$, curDoc = 1

Sorted Term	B	C	A
Doc	1	1	2
Cumulative Upper Bound	5	13	17

- pTerm = C
- pivot = posting[1].DID = 1
- pivot = curDoc (**Line10**)

  aterm = pickTerm(terms[0..pTerm]) (**assume choose C**)

  posting[C].DID = 2

In the 4$^{\text{th}}$ cycle

$\theta = 11$, curDoc = 1

Sorted Term	B	A	C
Doc	1	2	2
Cumulative Upper Bound	5	9	17

- pTerm = C
- pivot = posting[2].DID = 2
- pivot > curDoc (**Line14**)
  - curDoc $\neq$ posting[0].DID (**Line19**)

    aterm = pickTerm(terms[0..pTerm]) (**the idf of A is biggest**, **choose A**)

    posting[A].DID = 7

In the 5$^{\text{th}}$ cycle

$\theta = 11$, curDoc = 1

Sorted Term	B	C	A
Doc	1	2	7
Cumulative Upper Bound	5	13	17

- pTerm = C
- pivot = posting[C].DID = 2
- pivot > curDoc (**Line14**)
  - posting[0].DID $\neq$ pivot (**Line19**)

    aterm = pickTerm(terms[0..pTerm]) (**the idf of A is biggest**, **choose A**)

    posting[A].DID = 7

In this case, **pickTerm**() will always choose A because of the maximum idf. θ, curDoc and other parameters would not change as well.

Therefore, this algorithm will end up in an infinite loop.