COMP6714 Assignment 1

Name: Wenke Yang
ZID: z5230655

Nov. 2019

Question 1:

(1).

The algorithm basically splits each input list into two halves and recursively find the intersect elements between each half of A and each half of B. There are two base cases: one list is empty or two lists both have length 1.

Algorithm 1 Intersect(A, B)

```
if A.len == 0 or B.len == 0 then
  return []
else if A.len == 1 and B.len == 1 then
  if A == B then
    return A
  else
    return []
  end if
else
  half_len_A = floor(A.len/2)
  half_len_B = floor(B.len/2)

  first_half_A = A[:half_len_A]
  first_half_B = B[:half_len_B]

  second_half_A = A[half_len_A:]
  second_half_B = B[half_len_B:]

  return Intersect(first_half_A, first_half_B) +
    Intersect(first_half_A, second_half_B) +
    Intersect(first_half_B, second_half_A) +
    Intersect(second_half_A, second_half_B)
end if
```

(2).

Assume the new function is called **divide_list(A, B)**. It calls a helper function **divide_list_with_k(A, B, current_k)**, the **initial current_k** is the **hyper parameter K**. This helper function has the similar structure as Intersect(A, B). The differences are the details in base case and recursive case:

- The base case is when current_k is 1, return list of A and list of B directly.
- In the recursive case, the new parameter current_k is reduced by half in each call. Besides, there are only two recursive calls (rather than four). One is the first halves of A and B, the other is the rest halves of A and B, both with half of current_k.
- The return value of recursive call is the concatenation of the two returned A lists and the two returned B lists from above two recursive calls respectively.

The full algorithm of the helper function divide_list_with_k(A, B, current_k) is defined as below:

Algorithm 2 divide_list_with_k(A, B, current_k)

```
if current_k == 1 then
  return [A], [B]
else
  half_len_A = floor(A.len/2)
  half_len_B = floor(B.len/2)

  first_half_A = A[:half_len_A]
  first_half_B = B[:half_len_B]

  second_half_A = A[half_len_A:]
  second_half_B = B[half_len_B:]

  first_splits_A, first_splits_B = divide_list_with_k(first_half_A, first_half_B,
    current_k / 2)
  second_splits_A, second_splits_B = divide_list_with_k(second_half_A,
    second_half_B, k - (current_k / 2))
  return first_splits_A + second_splits_A, first_splits_B + second_splits_B
end if
```

Question 2:

(1).

Assume the highest level of indexes is n. Since the logarithmic strategy is used, there will be at most one index for each level from level 0 to level n. The number of level 0 index required to merge to each level of index is shown below:

index level	count	index	# of level 0 index required for merge
0	1	I_0	$1 = 2^0$
1	1	I_1	$2 = 2^1$
2	1	I_2	$4 = 2^2$
...	...	...	...
n	1	I_n	2^n

$$\begin{aligned} &\text{The max number of } I_0 \text{ we need} \\ &= 2^0 + 2^1 + 2^2 + ... + 2^n \\ &\approx 2^n \end{aligned} \tag{1}$$

The number of indexes for no merge is t and no merge means all indexes are in the same level, in our case is level 0, hence the number of I_0 we have is **t**.

$$\begin{aligned} t &= 2^n \\ n &= log_2 t \end{aligned} \tag{2}$$

Since the number of levels are integers, n should be $\lceil log_2 t \rceil$

(2).

Same as the previous question, we assume that the highest level of indexes is n. The max. total I/O cost should be the reading cost($t \cdot M$) and the sum of the I/O costs for merging I_0, I_1, I_2, ... and I_n. However, since we only consider the big O complexity, we just need to calculate the I/O cost for the largest term: merge cost of I_n. Here is a table to assist the calculation:

index level	index	no. of I_{level} need for generate I_n	index size
0	I_0	2^n	2^0M
1	I_1	2^{n-1}	2^1M
2	I_2	2^{n-2}	2^2M
...	...	...	...
n	I_n	2^0	2^nM

$$\begin{aligned} &\text{The total I/O cost for generating } I_n \\ &= \sum \text{no. of } I_{level} \text{ need for generate } I_n \cdot \text{index size} \\ &= 2^n \cdot 2^0M + 2^{n-1} \cdot 2^1M + 2^{n-2} \cdot 2^2M + ... + 2^0 \cdot 2^nM \\ &= 2^n \cdot M \cdot n \\ &= t \cdot M \cdot log_2 t \end{aligned} \tag{3}$$

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

Question 3:

(1).

$$\begin{aligned} precision_{top20} &= \frac{R\_in\_top\_20}{total\_number\_of\_retrieved} \\ &= \frac{6}{20} \\ &= 0.3 \end{aligned} \tag{4}$$

The precision of the system on the top-20 is 0.3.

(2).

$$\begin{aligned} recall_{top20} &= \frac{R\_in\_top\_20}{total\_number\_of\_relevant} \\ &= \frac{6}{8} \\ &= 0.75 \end{aligned} \tag{5}$$

$$\begin{aligned} F1_{top20} &= \frac{2 \cdot precision \cdot recall}{precision + recall} \\ &= \frac{2 \cdot 0.3 \cdot 0.75}{0.3 + 0.75} \\ &= \frac{3}{7} \end{aligned} \tag{6}$$

The F1 on the top-20 is $\frac{3}{7}$.

(3).

In order to assist answering the question(3) and (4), a table of moving precision and recall of the top-20 is attached below:

k-th input	1	2	3	4	5		6	7	8	9	10
$prec_k$	1.0	1.0	0.67	0.5	0.4		0.33	0.29	0.25	0.33	0.3
$recall_k$	0.13	0.25	0.25	0.25	0.25		0.25	0.25	0.25	0.38	0.38

k-th input	11	12	13	14	15		16	17	18	19	20
$prec_k$	0.36	0.33	0.31	0.29	0.33		0.31	0.29	0.28	0.26	0.3
$recall_k$	0.5	0.5	0.5	0.5	0.63		0.63	0.63	0.63	0.63	0.75

According to the tables above, the uninterpolated precisions of the system at 25% recall are: 1.0, 0.67, 0.5, 0.4, 0.33, 0.29, 0.25.

(4).

According to the tables in question (3), the interpolated precision at 33% recall which is the maximum precision after 8th input is 0.36 (at 11th input).

(5).

$$\begin{aligned} MAP &= \frac{\sum \text{precisions if the current input is relevant}}{\text{no. of relevant docs}} \\ &= \frac{1+1+0.33+0.36+0.33+0.3}{8} \\ &\approx 0.42 \end{aligned} \tag{7}$$

The MAP for the query is 0.42.

(6).

The largest possible MAP can be reached if the 21th and 22th results are both relevant.

$$\begin{aligned} MAP_{max} &= \frac{\sum \text{precisions if the current input is relevant}}{\text{no. of relevant docs}} \\ &= \frac{1+1+0.33+0.36+0.33+0.3+\frac{7}{21}+\frac{8}{22}}{8} \\ &\approx 0.50 \end{aligned} \tag{8}$$

The largest possible MAP that this system could have is 0.50.

(7).

The smallest possible MAP can be reached if the 9999th and 10000th results are relevant.

$$\begin{aligned} MAP_{min} &= \frac{\sum \text{precisions if the current input is relevant}}{\text{no. of relevant docs}} \\ &= \frac{1+1+0.33+0.36+0.33+0.3+\frac{7}{9999}+\frac{8}{10000}}{8} \\ &\approx 0.42 \end{aligned} \tag{9}$$

The smallest possible MAP that this system could have is 0.42.

(8).

$$\begin{aligned} max\_MAP\_error &= MAP_{max} - MAP_{min} \\ &= 0.50 - 0.42 \\ &= 0.08 \end{aligned} \tag{10}$$

The largest error for the MAP by calculating (5) instead of (6) and (7) could be 0.08.

Question 4:

(1).

$$\begin{aligned} no\_of\_words_{d1} &= 2+3+1+2+2+0 = 10 \\ no\_of\_words_{d2} &= 7+1+1+1+0+0 = 10 \end{aligned} \tag{11}$$

$$\begin{aligned} p(Q|d_1) &= \prod_{i=1}^{6} p(w_i|d_1) \\ &= \frac{2}{10} \times \frac{3}{10} \times \frac{1}{10} \times \frac{2}{10} \times \frac{2}{10} \times \frac{0}{10} \\ &= 0 \\ \\ p(Q|d_2) &= \prod_{i=1}^{6} p(w_i|d_2) \\ &= \frac{7}{10} \times \frac{1}{10} \times \frac{1}{10} \times \frac{1}{10} \times (\frac{0}{10})^2 \\ &= 0 \end{aligned} \tag{12}$$

According to the likelihood calculated, $p(Q|d_2)$ and $p(Q|d_1)$ are both zero, they should have the same rank.

(2).

$$
\begin{aligned}
p(Q|d_1) &= \prod_{i=1}^{6} p(w_i|d_1) \\
&= \prod_{i=1}^{6} 0.8 \cdot p_{mle}(w_i|M_{d1}) + (1-0.8) \cdot p_{mle}(w_i|M_c) \\
&= (0.8 \times \frac{2}{10} + 0.2 \times 0.8) \times (0.8 \times \frac{3}{10} + 0.2 \times 0.1) \times \\
&\quad (0.8 \times \frac{1}{10} + 0.2 \times 0.025) \times (0.8 \times \frac{2}{10} + 0.2 \times 0.025)^2 \times \\
&\quad (0.2 \times 0.025) \\
&= 9.63 \times 10^{-7}
\end{aligned} \tag{13}
$$

$$
\begin{aligned}
p(Q|d_2) &= \prod_{i=1}^{6} p(w_i|d_2) \\
&= \prod_{i=1}^{6} 0.8 \cdot p_{mle}(w_i|M_{d2}) + (1-0.8) \cdot p_{mle}(w_i|M_c) \\
&= (0.8 \times \frac{7}{10} + 0.2 \times 0.8) \times (0.8 \times \frac{1}{10} + 0.2 \times 0.1) \times \\
&\quad (0.8 \times \frac{1}{10} + 0.2 \times 0.025)^2 \times (0.2 \times 0.025)^2 \\
&= 1.30 \times 10^{-8}
\end{aligned}
$$

After recompute the likelihood with smoothing, document 1 has higher ranking.