COMP6714

Information Retrieval and Web Search

Assignment

Jingshi Yang

z5110579

Q1

1)

```
Intersection(A, B)
if A.Len = 1 or B.Len = 1 then
    Intersect ← [];
    if A.Len = 1 then
        if A[0] in B:
            Intersect.add(A[0]);
    else
        If B[0] in A:
            Intersect.add(B[0]);
    return Intersect;
else
    mid_A  ←⌈(A.Len + 1) / 2⌉;
    mid_B  ←⌈(B.Len + 1) / 2⌉;
    arr_1 = Intersection(A[:mid_A], B[:mid_B]);
    arr_2 = Intersection(A[:mid_A], B[mid_B:]);
    arr_3 = Intersection(A[mid_A:], B[:mid_B]);
    arr_4 = Intersection(A[mid_A:], B[mid_B:]);
    return arr_1 + arr_2 + arr_3 + arr_4;
```

2)

The logic of my first part is evenly dividing each list into 2 sub-lists and find the intersection. Same logic for my second part, evenly dividing each list into k sub-lists, but some changes on code:

1. In the outer if, change the condition to "**if** A.len / k <= 1 **or** B.len <= 1 **then** "
2. In the outer else, evenly dividing each list and add a loop to find the intersection.

Below is the new pseudocode, but the old and new pseudocode have same logic:

```
Intersection(A, B)
if A.Len / k <= 1 or B.Len / k <= 1 then
    Intersect ← [];
    if A.Len / k <= 1 then
        For each element in A:
            if  element in B:
                Intersect.add(element);
    else
        For each element in B:
            If element in A:
                Intersect.add(element);
    return Intersect;
```

```
else
        slice_A  ← A.Len / k;
        slice_B  ← B.Len / k;
        i ← 0;
        res ← [];

        while i < k:
                j ← 0;
                while j < k:
                        partial_res ← Intersection(A[int(i * slice_A): int((i + 1) * slice_A],
B[int(j * slice_B): int(i + 1) * slice_B)]);
                        res += partial_res;
                        j ← j + 1;
                i ← i + 1;

        return res;
```

Q2

1)
I use contradiction to prove at most $\lceil \log_2 t \rceil$ sub-indexes created by using logarithmic merge.

Now I **assume** there are $\lceil \log_2 t \rceil + 1 = y$ sub-indexes, I will calculate the smallest number which the $\lceil \log_2 t \rceil + 1$ sub-indexes can reach. Obviously the smallest number is $2^0 + 2^1 + 2^2 + \ldots + 2^{y-1}$ because no same generation number when use logarithmic merge, and here the power is continuous from 0 to y - 1, no gap between each two numbers. The reason why this is the smallest number is because the number of items are fixed(**y** sub-indexes), that means say for example, if we drop 2^1, then we must add 2^y at the end, if we want to drop an item, we must add one more bigger item to keep the number of sub-indexes equal to **y**.

Now I calculate the result of $2^0 + 2^1 + 2^2 + \ldots + 2^{y-1}$, the result is $2^y - 1$, then substitute y with $\lceil \log_2 t \rceil + 1$, I get $\mathbf{2^y - 1 = t + 1}$, that means if we have $\lceil \log_2 t \rceil + 1 = y$ sub-indexes, the smallest number when there are $\lceil \log_2 t \rceil + 1$ sub-indexes is **t + 1**. Obviously this violates the initial assumption the number **t**.

Hence if the logarithmic merge strategy is used, it will result in at most $\lceil \log_2 t \rceil$ sub-indexes.

2)
For example if we have t = 12 (when use no merge strategy), we need 12/2 = 6 times merges to merge two I_0 to one I_1, each cost is **1 read + 2 write = 3**, and 6/2 = 3 merges to merge two I_1 to one I_2, each cost is **2 read + 4 write = 6**, and $\lfloor 3/2 \rfloor = 1$ (I use floor here) times merge to merge two I_2 to one I_3, each cost is **4 read + 8 write = 12**.

So I can generate a math function to show the total I/O cost **tc**:
tc= $\lfloor t/2^1 \rfloor * (1 + 2)M + \lfloor t/2^2 \rfloor * (2 + 4)M + \lfloor t/2^3 \rfloor * (4 + 8)M + \dots + \lfloor t/2^h \rfloor * (2^{h-1} + 2^h)$
(Note: Here **h** is the largest number which can make 2^h smaller or equal to t)

Then simplify the equation, I get

$$tc = \sum_{i=0}^{h-1} \lfloor \frac{t}{2^{i+1}} \rfloor (2^i \times (1+2)) \times M$$

Then continue to simplify, get

$$tc = M \times (\frac{3}{2} t \times h)$$

And because h = $\lfloor \log_2 t \rfloor$, then substitute h with $\lfloor \log_2 t \rfloor$:

$$tc = t \times M \times \frac{3}{2} \lfloor \log_2 t \rfloor$$

Because we are finding big O and $\lfloor \log_2 t \rfloor$ is a variable, Hence the total I/O cost is
$O(t \times M \times log_2 t)$

Q3

1)
There are 6 correct documents out of 20 documents, hence

$$precision = \frac{3}{10} = 0.3$$

2)
There are 8 relevant documents in total, that means

$$Recall = \frac{3}{4} = 0.75$$

By using the f1 formula, hence

$$F1 = \frac{2 \times \frac{3}{10} \times \frac{3}{4}}{\frac{3}{10} + \frac{3}{4}} = \frac{3}{7} = 0.429$$

3)
When recall = 0.25, that means the number of relevant documents retrieved is
$8 \times 0.25 = 2$
Now we have 2 relevant documents, and then we can retrieve 2, 3, 4 … 8 relevant documents
Hence the uninterpolated precision of the system at 25% level may be

$$1, \frac{2}{3} \frac{1}{2}, \frac{2}{5}, \frac{1}{3}, \frac{2}{7}, \frac{1}{4}$$

4)
When at 33% recall, there are at least 3 documents retrieved which are relevant, so there are 4/11, 5/15, 6/20, clearly the highest accuracy is 4/11

Hence the interpolated precision at 33% recall is
$$\frac{4}{11} = 0.364$$

5)
According to the definition of MAP
Sum of precision when recall increases divided by The size of ground truth.
$$MAP = \frac{1}{6} \times (1 + 1 + \frac{3}{9} + \frac{4}{11} + \frac{5}{15} + \frac{6}{20}) = 0.555$$

6)
The largest MAP is
$$MAP = \frac{1}{8} \times (1 + 1 + \frac{3}{9} + \frac{4}{11} + \frac{5}{15} + \frac{6}{20} + \frac{7}{21} + \frac{8}{22}) = 0.503$$
7)
The smallest MAP is
$$MAP = \frac{1}{8} \times (1 + 1 + \frac{3}{9} + \frac{4}{11} + \frac{5}{15} + \frac{6}{20} + \frac{7}{9999} + \frac{8}{10000}) = 0.416$$

8)
The max error is $0.55 - 0.417 = 0.138$ and the min error is $0.55 - 0.503 = 0.052$
Hence the error is between **0.052** and **0.138**

Q4

1)
$$P(Q|d_1) = \frac{2}{10} \times \frac{3}{10} \times \frac{1}{10} \times \frac{2}{10} \times \frac{2}{10} \times \frac{0}{10} = 0$$
$$P(Q|d_2) = \frac{7}{10} \times \frac{1}{10} \times \frac{1}{10} \times \frac{1}{10} \times \frac{0}{10} \times \frac{0}{10} = 0$$
Because $P(Q|d_2) = P(Q|d_1)$

Hence they rank same

2)

$$P(Q|d_1) = (0.2\times0.8+0.8\times\frac{2}{10})\times(0.2\times0.1+0.8\times\frac{3}{10})\times(0.2\times0.025+0.8\times\frac{1}{10})\times(0.2\times0.025+0.8\times\frac{2}{10})\times(0.2\times0.025+0.8\times\frac{2}{10})\times(0.2\times0.025+0) = 9.62676\times10^{-7}$$

$$P(Q|d_2) = (0.2\times0.8+0.8\times\frac{7}{10})\times(0.2\times0.1+0.8\times\frac{1}{10})\times(0.2\times0.025+0.8\times\frac{1}{10})\times(0.2\times0.025+0.8\times\frac{1}{10})\times(0.2\times0.025+0)\times(0.2\times0.025+0) = 1.3005\times10^{-8}$$

Because $P(Q|d_2) < P(Q|d_1)$
Hence document 1 will be ranked higher