**Information Retrieval (CS60092) Test 2**
**Mar 22, 2022 Full Marks: 30**
**Time: 40 minutes (12:00 – 12:40) + 10 minutes for uploading answer-scripts**

** Write your name and roll number clearly on the first page. Write the page number on every sheet that you use.*
** Attempt all questions. All parts of the same question are to be answered together.*
** All workings must be shown, without which no marks will be awarded.*
** If you make any assumption, write the assumption clearly. Any assumption made should be logical.*
** Send* ***a single combined PDF*** *to* assignmentan1998@gmail.com *by 12:50 noon (according to the Moodle clock).* The format of the file would be **<ROLL_NUMBER.pdf>**
*** ***Answer-scripts must be submitted by 12:50 noon*** *(according to the Moodle clock). If you miss this deadline, you need to email your submission to* assignmentan1998@gmail.com *within 12:55 PM; there will be a penalty of 10 marks for such late submissions. No submission will be allowed after 12:50 PM. This time there will be* ***NO EXCEPTIONS****.*
** This is a* ***proctored exam****. You need to join the online examination hall allotted to you and be visible on camera through the entire duration of the class test. Otherwise, you will be considered absent even if you upload the answer-script.*

---

**Question 1 [(2.5 + 2.5) = 5]**
Compute (i) variable byte codes and (ii) gamma codes for the postings list 110, 401, 1169, 16273. Use gaps instead of docIDs for all except the first entry. Give the solution for variable byte codes as a sequence of 8-bit blocks. Give the solution for the gamma codes as a sequence of pairs of bit strings, where the first bit string of each pair corresponds to a length and the second to an offset.

| | VB Code | Gamma Code |
|---|---|---|
| 110 | 11101110 | 1111110,101110 |
| 401 (291) | 00000010<br>10100011 | 111111110,00100011 |
| 1169 (768) | 00000110<br>10000000 | 1111111110,100000000 |
| 16273 (15104) | 01110110<br>10000000 | 11111111111110,1101100000000 |

**Question 2 [2 x 5 = 10]**

(a) What are the worst-case time complexities of hashing and B-tree with blocking for dictionary lookup? Give examples to illustrate. (without examples, no marks will be awarded)

(b) Explain how permuterm index would be used to match the wildcard query ***ntr*** with the words, "intro" and “intra” while avoiding "train" (which can be permuted to “intra”).

(c) What *Boolean query* on a *trigram index* would be generated for the same query “red*”? Can you think of a term that matches the Boolean query, but does not satisfy the Permuterm query for “red*”?

(d) Why is precision used as a metric than accuracy? F1 score or F-measure is the harmonic mean between precision and recall. Why do we not use simple averaging (arithmetic mean)?

(e) Suppose you want to use n-gram overlap method for spelling correction using a Jaccard overlap threshold of k (0<k<1). Imagine an index of n-grams to the terms in vocabulary. Explain how you would modify the standard posting merge method to compute the spell-corrected terms.

---

(a) Hashing will be O(n). B-tree with blocking. Assume block size k, number of elements n. Max path will be log_b n + k. The bound will be O(log_b n + k)

**(b) Should be self-explanatory. We use *ntr*. Rotate *ntr* -> ntr***

intro$ - **ntro$i** -> tro$in -> ro$int -> o$intr -> $intro

Intra$ - **ntra$i** – tra$in – ra$int – a$intr - $intra

train$ -rain$t – ain$tr – in$tra – *n$trai* - $train

ntr* will match the bold the prefix ntr, which will be there in B-tree because of the boldened letters. Ntr* will not match n$trai because of the $ in the middle.

**(c) $red* -> $re AND red    Permuterm: $red***

**recurred**

(d) Accuracy is tp+tn/(tp+fp+tn+fn). Precision is tp/(tp+fp). The term tn and fn will always dominate. Any “clever” IR system will simply retrieve less documents to game the accuracy metric.

P and R both have true positives in their numerators, they have different denominators. So, it makes sense to average their reciprocals.

Also, taking arithmetic mean can be catastrophic for extreme cases (P=0, R=1, or vice versa).

(e) From IR Book, Page 98, solution key should be along the following line:

a. As the scan proceeds, we proceed from one vocabulary term t to the next, computing on the fly the Jaccard coefficient between q and t. If the coefficient exceeds a preset threshold, we add t to the output; if not, we move on to the next term in the postings. To compute the Jaccard coefficient, we need the set of k-grams in q and t.
b. Since we are scanning the postings for all k-grams in q, we immediately have these k-grams on hand. What about the k-grams of t? In principle, we could enumerate these on the fly from t; in practice this is not only slow but potentially infeasible since, in all likelihood, the postings entries themselves do not contain the complete string t but rather some encoding of t. The crucial observation is that

to compute the Jaccard coefficient, we only need the length of the string t. To see this, recall the example of Figure 3.7 and consider the point when the postings scan for query q = bord reaches term t = boardroom. We know that two bigrams match. If the postings stored the (pre-computed) number of bigrams in boardroom (namely, 8), we have all the information we require to compute the Jaccard coefficient to be 2/(8+3−2); the numerator is obtained from the number of postings hits (2, from bo and rd) while the denominator is the sum of the number of bigrams in bord and boardroom, less the number of postings hits.

**Question 3 [2 x 5 = 10]**

Consider a document collection having

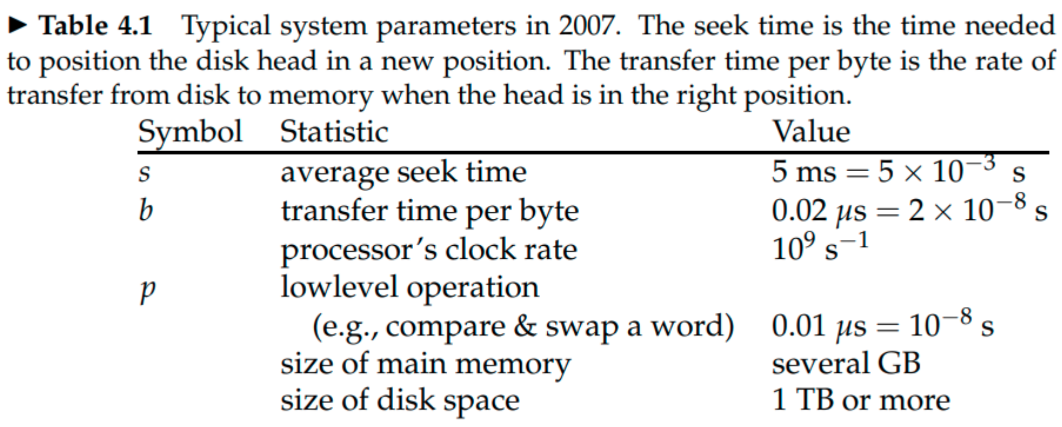
Number of terms: 1,50,000

Number of tokens: 30,000,000

Bytes per token: 8

Bytes per term id / doc id: 4

Consider a system with the typical system parameters shown below.



► **Table 4.1** Typical system parameters in 2007. The seek time is the time needed to position the disk head in a new position. The transfer time per byte is the rate of transfer from disk to memory when the head is in the right position.

| Symbol | Statistic | Value |
|---|---|---|
| $s$ | average seek time | 5 ms $= 5 \times 10^{-3}$ s |
| $b$ | transfer time per byte | 0.02 $\mu$s $= 2 \times 10^{-8}$ s |
| | processor's clock rate | $10^9$ s$^{-1}$ |
| $p$ | lowlevel operation (e.g., compare & swap a word) | 0.01 $\mu$s $= 10^{-8}$ s |
| | size of main memory | several GB |
| | size of disk space | 1 TB or more |

We want to construct an index for the document collection using external storage. We apply BSBI indexing for this purpose. Compute the total time required for BSBI indexing for each of the following steps. Assume we have 15 blocks to read data from or write data to.

(a) Reading collection of data from disk

(b) Total sorting time of blocks

(c) Total time for writing sorted blocks to disk

(d) Apply MapReduce to the problem of counting how often each term occurs in a set of files. Specify map and reduce operations for this task, following the schema in below figure.

**Schema of map and reduce functions**

| | |
|---|---|
| map: input | $\rightarrow$ list$(k, v)$ |
| reduce: $(k,$list$(v))$ | $\rightarrow$ output |

**Instantiation of the schema for index construction**

| | |
|---|---|
| map: web collection | $\rightarrow$ list(termID, docID) |
| reduce: $(\langle$termID$_1$, list(docID)$\rangle$, $\langle$termID$_2$, list(docID)$\rangle$, ...) | $\rightarrow$ (postings_list$_1$, postings_list$_2$, ...) |

**Example for index construction**

| | |
|---|---|
| map: $d_2$ : C died. $d_1$ : C came, C c'ed. | $\rightarrow$ $(\langle$C, $d_2\rangle$, $\langle$died,$d_2\rangle$, $\langle$C,$d_1\rangle$, $\langle$came,$d_1\rangle$, $\langle$C,$d_1\rangle$, $\langle$c'ed,$d_1\rangle)$ |
| reduce: $(\langle$C,$(d_2,d_1,d_1)\rangle$,$\langle$died,$(d_2)\rangle$,$\langle$came,$(d_1)\rangle$,$\langle$c'ed,$(d_1)\rangle)$ | $\rightarrow$ $(\langle$C,$(d_1$:2,$d_2$:1$)\rangle$,$\langle$died,$(d_2$:1$)\rangle$,$\langle$came,$(d_1$:1$)\rangle$,$\langle$c'ed,$(d_1$:1$)\rangle)$ |

(e) For distributed indexing, we use the MapReduce algorithm. Assume that machines in MapReduce have 100 GB of disk space each. Assume further that the postings list of the term the has a size of 200 GB. Do you need to modify the MapReduce algorithm to handle such cases? If yes, provide the sketch of the algorithm? If not, why?

a) Reading the collection from disk = Number of token * Average bytes per token * Time to transfer each byte.+ disk seek time

Reading collection from disk= 30,000,000*8*2*10^(-8)+2*10^(15)*5*10^(-3)=4.95s

b) Total sorting time = Number of blocks * (2*N log N) * time for low level operation, N = Number of records in each block
   N=number of tokens/no.of blocks= 3*10^(7)/15=2*10^(6)
   Log N should be base 2, as its sorting, not e or 10.
   Total sorting time= 15*2*2*10^(6)*log(2*10^(6))*10^(-8)=60*20.93*10^(-2)=12.54s

c) Total writing time for writing sorted blocks back to disk = Number of blocks * time for writing single block

Total writing time for writing sorted blocks back to disk=15*24120=361800s

   a. Time for writing single block = (Number of terms per block + Number of posting in each block ) * (size of term id / doc id) * time for transferring 1 byte data to disk
   Time for writing single block=(150000/15+ 2*10^(6))*(150,000*4)*2*10^(-8)=24120s

(d)
Instantiation of the schema for index construction
map: web collection → list(termID, docID)
reduce: ( termID1 , list(docID) , termID 2 , list(docID) , . . . ) → (<termID1, count(list(docID))>

If position is required

map: web collection → list(termID, docID, posID)
reduce: ( termID1 , list(docID-posID) , termID 2 , list(docID) , . . . ) → (<termID1, count(list(docID-posID))>…)

(e) Master keeps track of which machines have their disks full.

1. Master only assigns parsing jobs if the disk is not full.
2. Whenever parser sees disk space is full, sends a signal to Master. Master updates its array. Master assigns another parser to finish the job, only if that parser has disk-full bit set.

Otherwise, Master can do an approximate estimate from a small sample of docs. How big is the total postings lists are becoming. Then, assigns in a pessimistic way, so that it never runs out Parser memory. Both of these have some or the other tradeoff.

**Question 4 [5 marks = 0.5+1+0.5+3)**

Imagine you are designing an IR system for an e-commerce website with relevance feedback. For relevance feedback, suppose you decide to ask for feedback on K documents in each iteration, and iterate T times.

(a) Quantify the man-hour effort in terms of K and T (assume variables what you need to assume).

(b) Recall the Rocchio's (original) optimization objective. How do you make a change to the optimization equation to factor in the fact that you want to minimize the total man-hour effort, as well as maximizing the optimal query's similarity to $D_r$ (relevant docs) and minimize the similarity to $D_{nr}$ (non-relevant docs). Do you need to introduce any other parameter?

(c) Show that pseudo-Relevance feedback minimizes the man-hour effort based on your equation in (b).

(d) Design a hybrid algorithm that makes use of both relevance and pseudo-relevance feedback ideas. Show how in terms of man-hour effort it is a tradeoff between the two (RF and pseudo-RF). You can make the assumption that recall for RF is higher than recall for PsRF in each iteration. What is the effect of your hybrid algorithm on recall?

(a) (D_r+D_nr)*T*h hours or KTh

(b) $q_{opt}, K, D_r, T = argmax_{q,K,D_r,T}\ (sim(q,\mu(D_r)) – sim(q,\mu(D_{nr})) - \lambda * \mathbb{I}_{RF} * \log_{10}\{(D_r + D_{nr}) * T * h\}$

$\mathbb{I}_{RF} = 1$, if RF is used, 0 if RF is not used. This is a trick to make sure the argument to log does not become zero. Of course, $K, T, \lambda$ all these are hyperparameters. Can be found using grid search.
NOTE, 0.5 will be subtracted if students do not include some sort of smoothening or normalization.

(c ) simply T and h will be zero. So. Minimizes the man-hour effort
(d )
A simple algo would be randomization. In each iteration, randomly choose with probability 0.5 to go to humans vs. use previous iterations top k as relevant. There could be other intelligent alternatives.

For random instance, the man-hour effort will be upper bounded by a factor of 2 in comparison to the RF. Pseudo-RF of course its zero.

Bounding recall requires probabilistic analysis. But if you assume some relation between recall in every iteration of PsRF and RF, suppose $recall_i(RF) > recall_i(PsRF)$ for all i. Then, it can be shown that $recall(hybrid) \leq \max(recall(RF), recall(PsRF))$. This is simplistic assumption.