1. **Define the following information retrieval terms: (a) probability ranking principle, (b) specificity, (c) exhaustivity, (d) novelty.**

**Probability Ranking Principle** : If an IR system's response to each query is a ranking of the documents in the collection in order of decreasing probability of relevance, then the overall effectiveness of the system to its users will be maximized.

Or in simpler words - The Probability Ranking Principle is a fundamental concept in information retrieval. It asserts that, for optimal retrieval effectiveness, documents should be ranked in descending order of the probability that they will be relevant to the user's information need, given the available evidence. In simpler terms, this principle suggests that the most relevant documents to a user's query should be ranked highest.

**Specificity** : The specificity of the document measures the degree to which its contents are focused on the information needed. For example, the document might satisfy the need but also contain lots of other garbage.

**Exhaustivity** : The exhaustivity of the document reflects the degree to which it covers the information related to the topic.

Or in other words - Exhaustivity is the extent to which an information retrieval system or indexing process captures all possible information or aspects about a particular topic or document. An exhaustive index, for example, would account for every topic or concept contained within a document, making the information more discoverable. However, high exhaustivity might lead to increased noise in the search results as even marginally relevant topics would be indexed.

**Novelty :** The novelty of a document in a document list reflects how much new information the n-th search result gives to the user given the previous results.

Or - In the context of information retrieval and recommendation systems, novelty refers to the introduction or suggestion of new, previously unseen, or unknown items or information to the user. A system that has high novelty will provide users with unique or fresh content, as opposed to repeating the same or similar suggestions. This is particularly important in recommendation systems where users are looking for new content or experiences.

1. **Suppose one had a corpus of Barack Obama speeches from which one developed a language model M. From these speeches it can be determined that he uses a introduces new word with probability 1/(current\_corpus\_length). Suppose the current corpus length 100,000 words. Determine a language model M′ that would include his next speech (which we know in advance is 1000 words).**

[Attached in class thread](https://sjsu.instructure.com/users/4591749/files/74695969/preview?verifier=aJZxZ1NPdm7vrbFSNcg2wfkLYsDPd7wLAGF5Aztc)

1. **Define and give an example posting for the following index types: (a) docid index, (b) frequency index, (c) positional index, (d) schema-independent index.**

Attached in class thread

Docid index: A docid index stores for each term just the documents in which it appears.

Frequency index: A Frequency index stores for each term and for each document it appears in the pair ⟨d,freq(t,d)⟩

Positional index: A positional index consists of postings of the form (d, freq(t,d), <positions>)

Schema-independent index: A schema-independent index does not store any of the document oriented optimizations of a positional index, but otherwise stores the same info.

For the following corpus:

Document 1: "Apples are delicious."

Document 2: "Bananas are a healthy snack"

Document 3: "Monkeys like bananas"

| **term** | **docid list** | **frequency list** | **positional list** | **schema independent** |
| --- | --- | --- | --- | --- |
| **a** | **1: 2** | **1:((2,1))** | **1:((2,1, <3>))** | **1:6** |
| **are** | **2: 1,2** | **2:((1,1), (2,1))** | **2:((1,1,<2>), (2,1,<2>))** | **2:2,5** |
| **apples** | **1:1** | **1:(1,1)** | **1:(1,1, <1>)** | **1:1** |
| **bananas** | **2:2,3** | **2:((2,1), (3,1))** | **2:((2,1, <1>), (3,1, <3>))** | **2:4,11** |

1. **Suppose on a query for "Eloise et Abelard" a search engine returns 3000 results, 500 of which are relevant. There are in the indexed corpus 4000 documents relevant to this query. Calculate the precision and recall given these numbers.**

Given:

Number of retrieved results: 3000  
 Number of relevant results retrieved: 500  
 Total number of relevant documents in the indexed corpus: 4000

We can compute precision and recall using the following formulas:

Precision: This is the fraction of retrieved documents that are relevant. It's calculated as:  
 Precision = Number of relevant documents retrieved / Total number of documents retrieved

Recall: This is the fraction of the total relevant documents that have been retrieved. It's calculated as:  
 Recall= Number of relevant documents retrieved /Total number of relevant documents

\*\*Precision\*\*: 500 / 3000 = 0.1667

\*\*Recall\*\*: 500/4000 = 0.125

1. **Suppose we have three topic areas and two relevant documents for each topic areas. Assume your search engine eventually returns the relevant results for each topic. Given a concrete example showing one possible MAP value might be calculated (i.e., you can say what rank your search engine returns the relevant results, but otherwise the computation is determined).**

Example :

Total number of documents = 4

Among them there are 2 relevant documents for each topic area.

Total number of topics areas = 3

Let’s suppose for each example the following documents are returned:

Topic A:

1. Document A1
2. Document A2 (Relevant)
3. Document A3
4. Document A4 (Relevant)

Precision at A2 = 1(Number of relevant documents) / 2 (the total number of docs examined till that document) = 0.5

Precision at A4 = 2 (Number of relevant documents) / 4 (the total number of docs examined till that document) = 0.5

Average Precision = (0.5 + 0.5 / 2) = 0.5

Topic B:

1. Document B1 (Relevant)
2. Document B2 (Relevant)
3. Document B3
4. Document B4

Precision at B1 = 1/1 = 1

Precision at B2 = 2/2 = 1

Average Precision = (1+1 / 2) = 1

Topic C:

1. Document C1
2. Document C2 (Relevant)
3. Document C3 (Relevant)
4. Document C4

Precision at C2 = 1/2 = 0.5

Precision at C3 = 2/3 = 0.6

Average Precision = (0.5 + 0.6 / 2) = 0.55

MAP = The Arithmetic Mean of Precision of all topics / total number of topics

MAP = (0.5 + 1 + 0.55) / 3 = 0.683

1. **Suppose a posting list for a term t for a schema-independent index consisted of the numbers 2,3,9,12,77,470,1100,1400,2300. Explain how the galloping search algorithm from class would compute next(t,499).**

[Attached in class thread](https://sjsu.instructure.com/users/4513558/files/74695743/preview?verifier=Z3l2A6RWQ6vD4q5GFvObW5sP4XtmKO4BJqmhKIVk)

1. **Explain how pooling can be used to assign relevance judgements when a corpus of documents is too large.**

Pooling is a widely adopted technique in information retrieval to deal with the challenge of manually judging the relevance of a large number of documents for a set of queries, especially when the corpus is vast. Here's how pooling works and can be used to assign relevance judgments:

1. One or more experimental runs from each group participating in an adhoc task are accepted.

2. Each of these runs consist of top 1000 or 10000 documents for each topic.

3. Top 100 or so documents from each run are pooled into a single set

4. These documents are presented to assessors in a random order of judging. This takes about a week.

5. One group might rank a document 50 and another group 150. If it's in the top 100 for any group it will be judged. This is called pool depth.

6. Unjudged documents are treated as not relevant.

7. Precision@10, recall@k and Mean Average Precision scores are calculated.

1. **Give the Proximity Ranking algorithm discussed in class. Explain how ranking can be done using the vector space model and TF-IDF scores. Give a concrete example being specific about how you are handling TF-IDF scores for queries.**

[Attached in class thread](https://sjsu.instructure.com/courses/1572810/discussion_topics/5061626)

1. **Briefly explain (a) how autoloading in PHP works, (b) how to set up a composer project to use Yioop.**

a)Autoloading in PHP

1.Autoloading in PHP is the process of including or requiring class files when they are needed in your application. Using autoloading, allows us to minimize the number of files we read in before executing code.

2.The require and require\_once commands read in the file given as their argument regardless of whether the classes, functions, etc. are ever used In a web and interpreted environment this can result in slower performance, if there are a lot of files being included in this way.

3.In PHP, we can use the spl\_autoload\_register($your\_autoloader\_function); command to register a PHP function to be called whenever someone tries to instantiate a class or call a function which is not currently in memory.

4.The autoloader function typically makes use of the namespace info from use declarations to search the file system to find the definition of the class or function that is being instantiated or used.

b) how to set up a composer project to use Yioop:

Composer is a package management system for PHP. It lets people combine various PHP projects, making sure that the dependencies between the current project and different versions of the projects it depends are maintained.

To Setup a Composer Project to use Yioop, We will do the following steps:

1.A composer project has at a minimum a composer.json file, and a sub-folder labeled vendor.

2.The composer.json file is used to specify what version of which projects your project depends on.

3.The vendor folder is where Composer downloads and stores the correct versions of these projects.

4.We then have an index.php file in the root folder of your project, have a src folder with the src code of your project and have a tests folder with any unit tests.

Composer Install

Composer Update

php index.php

1. **Suppose n=4 what would be the character n-grams for the word *caramel*? What is stopping? What is stemming ? Give an example of the kinds of rules used by a Porter stemmer.**

[Attached in class thread](https://sjsu.instructure.com/courses/1572810/discussion_topics/5061626)

**Past questions**

1. Consider the sentence "two bees or not two fleas". What is M1(bees∣∣two)? Describe how a first order model can be smoothed by a zero'th order model.

M1(bees|two) = freq(two bees)/sum of freq(two x) where x is terms in the sentence that follow two.

M’1(bees|two) = y.M1(bees|two) . (1-y)M0(bees)

M0(bees)= freq(bees)/sum of freq of all terms in seq

1. Give an example Markov Model in which it is possible to generate the string "The good and the bad" from state 1. Show how to calculate the probability of this phrase from state 1 in your model.

Make a finite state wth 4 states makes direction and weight the transition with some value

Phrase calculation using product of going from state x t state y

1. Give the pseudo-code for the nextPhrase(t[1],t[2], .., t[n], position) algorithm from class. Give its run time under different implementations of the next(t,position) function (no need for proofs).

In book

Second part : [discussion thread](https://sjsu.instructure.com/courses/1572810/discussion_topics/5043686)

1. Give the PHP code to read in a file counter.txt which stores an integer as a string. Your program should then output to the default output stream this value, increment the value, and write it back to disk.

<? Php

Function readinFIles(){

$str=file\_get\_content(“counter.txt”);

Echo “value is $str”;

$val=inval($str);

$val++;

file\_put\_contents(“counter.txt”, strval($val))

Echo “new value is $val”

}

1. Assume we have a fully positional index. Give pseudo code for an implementation of next(t, n:m) that uses galloping search where appropriate.

hw2

1. Give pseudo-code for the proximityRank algorithm from class together with a short example showing how it would process a score a query for a illustrative two document corpus.

Assignment in class

1. Define or give the equation of the following concepts: (a) TF-IDF, (b) cosine-similarity, (c) docRight(Q, u), (d) precision@k.

precision@k=∣∣Rel∩Res[1..k]∣∣∣∣Res[1..k]∣∣

TF - IDF : Over the years, several different functions have been proposed for TF and for IDF. For this class, we will define IDF=log(NNt) and we will define TF=log(ft,d)+1 if ft,d>0 and 0 otherwise.

docRight(Q, u): end point of the first candidate solution to Q starting after document u

docLeft(Q, v) : start point of the last candidate solution to Q ending before document v

1. Briefly explain how autoloading works in PHP. What is Composer? Give an example of a couple of its commands.

[Midsem practise thread](https://sjsu.instructure.com/courses/1572810/discussion_topics/5061626)

1. Pick your favorite (family-friendly) limerick. Give the probability of each term in this limerick according to a first order language model. Compute the MLE of the second line.

[Midsem practise thread](https://www.yioop.com/thread/443721)

1. Consider the "As I was going to St Ives ..." riddle, explain how the nextPhrase algorithm would work on inputs: nextPhrase("had", "seven", 10). As there are variants to the riddle, tell me which one you are using as you go through the solution.

[Midsem practise thread](https://www.yioop.com/thread/443721)

1. **Suggest a data structure that could be used to implement an inverted index in PHP, and give a working code snippet to show how to implement first($t), last($t) from our inverted index ADT.**

[Midsem practise thread](https://www.yioop.com/thread/443721)

1. Give pseudo-code for an galloping search implementation of prev(t, pos), where prev is from out inverted index ADT.

hw2

1. Define the following terms (1pt each): (a) docid index, (b) frequency index, (c) positional index, and (d) schema-independent index.

[Midsem practise thread](https://www.yioop.com/thread/443721)

1. In the following two documents fill in the blank with your home town and your favorite holiday respectively. (1) When I was little, I don't recall space aliens visiting \_\_\_\_. (2) Maybe the space aliens were waiting for \_\_\_\_. Assume we are using the vector space model with TF-IDF scores for the components. Compute the TF-IDF vectors for each document. Compute their cosine similarity.

[Midsem practise thread](https://www.yioop.com/thread/443721)

1. Give the nextSolution algorithm from class for positive boolean queries. Show how it would work on a concrete corpus and query.

Slides

nextSolution(Q, position)

{

v := docRight(Q, position);

if v = infty then

return infty;

u := docLeft(Q, v+1);

if(u == v) then

return u;

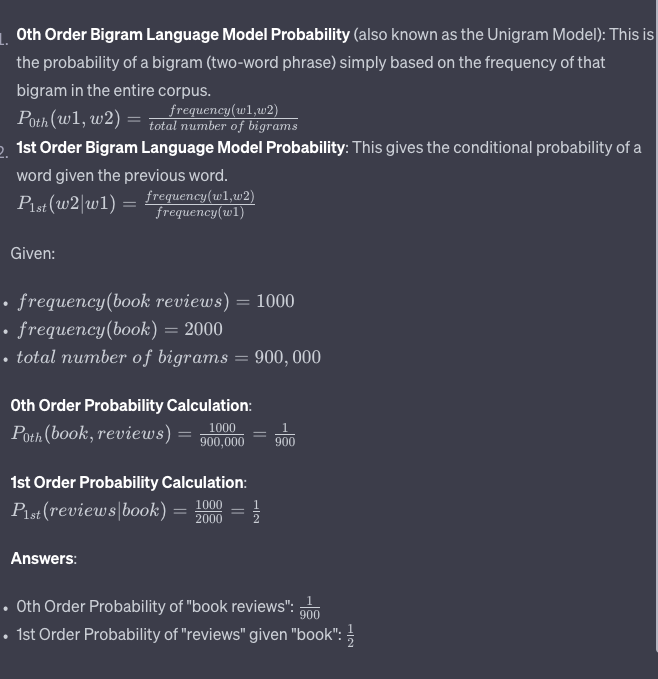
else

return nextSolution(Q, v);

}

1. Suppose `n=3` what would be the character `n`-grams for the word **toast**? (1pt) What is stopping? (1pt) What is stemming (1pt)? Give an example of the kinds of rules used by a Porter stemmer.

Same like Midsem practise thread

1. Suppose the phrase **book reviews** occurs 1000 times in a corpus of 900,000 bigrams, whereas, the term **book** appears 2000 times in our corpus. What is the 0th order bigram language model probability of **book reviews**? (1pt answer, 1pt work). What is the first model probability? (1pt answer, 1pt work).
2. Give the algorithm from class for phrase search of the phrase given by terms t[1], ..., t[n] after the location **position**, using our inverted index ADT. (4pts).
3. Suggest a data structure that could be used to implement an inverted index in PHP (2pts)), and give a working code snippet to show how to implement first($t) (1pt), last($t) from our inverted index ADT.

"All PHP arrays can be viewed as associative arrays".

We will thus use an associative array for our inverted index and indexed arrays for the posting list within the index.

// assume our associated array inverted index is globally accessible and referred to as $inverted\_index

function first($t) {

if (array\_key\_exists($t, $inverted\_index))

return reset($inverted\_index[$t]);

return false;

}

function last($t) {

if (array\_key\_exists($t, $inverted\_index))

return end($inverted\_index[$t]);

return false;

}

1. **Suppose the schema independent posting list for bob looks like 1, 3, 120, 1000, 1200, 1201, 1202, 1300. Explain how next(bob, 2) would be computed using galloping search as the implementation for next. (4pts).**

Midsem practise thread

1. Suppose our whole corpus consists of the two sentences: (a) fox news was quick to report the story. (b) the quick brown fox jumped over the lazy dog. Assume we are using the vector space model with TF-IDF scores for the components. Compute the TF-IDF vectors for each document (1pt each). Compute their cosine similarity (2pts).

Same like Midsem practise thread

1. Give the Proximity Ranking algorithm discussed in class.

slides

1. Construct a simple language model based on the following word frequencies for a corpus: bob 3, knows 10, sally 2. Using this model compute the MLE for the phrase: bob knows sally knows.

The Maximum Likelihood Estimate (MLE) for a word in a language model is simply the probability of that word given the corpus. It's computed using the following formula:

\[ P(word) = \frac{\text{Frequency of word in the corpus}}{\text{Total number of words in the corpus}} \]

From the given data, let's first calculate the total number of words in the corpus:

Total words = 3 (bob) + 10 (knows) + 2 (sally) = 15

Now, compute the MLE for each word:

1. \( P(bob) \) = \( \frac{3}{15} \) = 0.2

2. \( P(knows) \) = \( \frac{10}{15} \) = 0.6667 (rounded off to four decimal places)

3. \( P(sally) \) = \( \frac{2}{15} \) = 0.1333 (rounded off to four decimal places)

Given the phrase "bob knows sally knows", the probability (or MLE) for this phrase is the product of the probabilities of each word:

\[ P(bob\ knows\ sally\ knows) \]

\[ = P(bob) \times P(knows) \times P(sally) \times P(knows) \]

\[ = 0.2 \times 0.6667 \times 0.1333 \times 0.6667 \]

\[ \approx 0.0178 \] (rounded off to four decimal places)

So, the MLE for the phrase "bob knows sally knows" is approximately 0.0178.

1. Explain and give a concrete example of smoothing a first order language model with a zeroth order language model.

Done above

1. Write a PHP version of the nextPhrase algorithm from class. You can assume functions: next($t, $v, $d), prev($t, $v, $d), first($t, $d), last($t,$d) have already been implemented where the additional argument $d is an inverted index object of some kind. Use the strings "infty" and "-infty" if you go off posting lists.

<?php

Function nextPhrase($t, $pos){

$v=$pos

for($i=1;$i<=n;$i++){

$v=next($t[$i],$v)

}

if($v==”infit”){

Return [“infint”,”infty”]

}

$u=$v

for($i=$n-1;$i>=1;$i–){

$u=prev($t[$i],$u)

}

if($v-$u==$n-1){

Return [$u,$v]

}

Else{

Return nextPhrase($t, $u)

}

}

1. **Briefly describe the following kinds of inverted indexes: (a) docid index, (b) frequency index, (c) positional index.**

Midsem practise thread

1. Define and give an example of calculating each of the following effectiveness measures: (a) Recall, (b) Precision@5, (c) MAP.

slide

1. Step through how the algorithm from class would work to compute nextCover("what", "you", 2) if the corpus consisted of one document containing the phrase: "What you do speaks so loudly that I cannot hear what you say".

In class discussion

1. Explain how the vector space model works. Give an example showing how to compute the cosine similarity between a query vector and a short document of your choice.

Midsem practise thread

1. **Give the pseudo-code for the nextPhrase(t[1],t[2], .., t[n], position) algorithm from class. Give its run time under different implementations of the next(t,position) function (no need for proofs).**

Midsem practise thread

1. **Give the PHP code to read in a file my\_integer.txt which stores an integer as a string. Your program should then output to the default output stream this value, increment the value, and write it back to disk.**

DONE ABOVE

1. Briefly describe the galloping search implementation of the next(t, position) function.

Midsem practise thread

1. Define or give the equation of the following concepts: (a) a cover of a term vector, (b) the proximity scoring function, (c) docRight(Q, u), (d) precision@k.

slide

1. **Explain and give an example of how to compute the maximum likelihood estimate of a phrase based on a language model for terms in a vocabulary.**

Midsem practise thread

1. Suppose the word zwaggered appears four times in our corpus and appears twice in document 7. Suppose our corpus has 16 documents. What would be the TF-IDF score for zwaggered in document 7.

Midsem practise thread

1. **What is MAP? Give an example of how to calculate a MAP score.**

Midsem practise thread

***FINALS***

1. Briefly give the API operations expected of an inverted index dictionary that supports indexing, queries, including prefix queries. Briefly describe the insert-at-back and move-to-front heuristic heuristics for dictionary term collisions.

Sol: The API operations could consist of indexing: Inserting term (Insert a term t given) , searching for terms (Search for a term t and return if found), and searching by prefix (Return all terms t that start with given prefix p).

If there is a collision, for the insert-at-back heuristic: the new term is inserted in the back of the list of terms at that location. For the move to front heuristic: When a term is accessed, it gets moved to the front of the list, because it is expected that it will most likely be accessed soon.

1. Describe the merge-based index construction algorithm (disk-based). How is it modified if logarithmic merging is done? Give an example.

Sol: Part 1:

* we loop over eah dicument ansd check tokens , we check if each token is in the dictionary, if not, we add the term to the dictionary, if it’s there, we search for the term and add it to the posting list.
* we have a fixed memory capacity, so increament the memory consumption value every time we see a token.
* if the memory consumption is more than the memory limit, we create a new partitions,
* finally we use a disk based merge to merge all the partitions to form the final index.

**buildIndex\_mergeBase(inputTokenizer, memoryLimit)**

**{**

**n := 0;**

**position := 0;**

**memoryConsumption := 0;**

**while (inputTokenizer.hasNext()) {**

**T := inputTokenizer.getNext();**

**obtain dictionary entry for T;**

**create new entry if necessary;**

**append new position to T's posting list**

**position++;**

**memoryConsumption++;**

**if (memoryConsumption > memoryLimit) {**

**createIndexPartition();**

**}**

**}**

**if (memoryConsumption > 0) {**

**createIndexPartition();**

**}**

**mergeIndexPartitions(I[0],...,I[n-1])**

**// to make final index I\_final;**

**}**

**// in the partition create phase, we check for all terms in the memory index and put their Posting List in the disk and then put the dictionary in the disk. Then reset the memory consumption parameter and do for next index**

**createIndexPartition()**

**{**

**create empty on disk inverted file I[n];**

**sort in-memory dictionary entries in lex order;**

**for each term T in dictionary {**

**add T's posting list to I[n];**

**}**

**delete all in memory posting lists;**

**write the dictionary to disk**

**reset the in-memory dictionary;**

**memoryConsumption := 0;**

**n++;**

**}**

**mergeIndexPartitions([I[0], ..., I[n-1]])**

**{**

**create empty Inverted File I\_final;**

**for (k = 0; k < n; k++) {**

**open partition I[k] for sequential processing;**

**}**

**currentIndex := I[0];// anything other than nil so go through loop once**

**while (currentIndex != nil) {**

**currentIndex = nil;**

**for (k = 0; k < n; k++) {**

**if (I[k] still has terms left) {**

**if (currentIndex == nil ||**

**I[k].currentTerm < currentTerm) {**

**currentIndex := I[k];**

**currentTerm := I[k].currentTerm;**

**}**

**}**

**}**

**if (currentIndex != nil) {**

**I\_final.addPostings(currentTerm,**

**currentIndex.getPostings(currentTerm));**

**currentIndex.advanceToNextTerm();**

**}**

**}**

**delete I[0], ..., I[n-1];**

**}**

**Part 2: in logarithmic merging :**

* To begin, we imagine our index is split into generations *g*which are whole numbers 1,2...
* When we write a partition, the postings go on disk into a generation 1 index.
* We then check are there two generation 1 indexes? If yes, we merge them to make a generation 2 index and then delete the generation 1 indexes.
* We then repeat the process by checking if we now have two generation 2 indexes. If so, merge them and so on.
* At any given time, we will have at most logarithmic in the number of partitions we have written many active generations.
* I.e., a small number, so to look up a token we can look it up in each of these indexes and concatenate these posting lists.
* The number of postings transferred from/to disk using this strategy is Θ(*N*⋅log(*N/M*))

**Part 3: Example:**

**1 2 4**  we have gen1 , gen2 and gen4 paritions in disk.

**3 4**  After 1st partition is written to disk, we have 2 gen1 partitions which merge to form 1 gen2 index. Now we have 2 gen2 in disk which merge to form gen3 index in disk. So we have in step 2 gen3 and gen4.

**1 3 4** Next partition is written to disk , now we have gen1 , gen3 and gen 4

**2 3 4**  Next partition is written to disk, the disk would now contain 2 gen1 partitions which merge to gen2 index. Finally we have gen2, gen3 and gen4.

1. What is a generalized concordance list? Using the operators we had for such lists, express the query, "Return all intervals that begin with a <problem> and end with a </problem>" that contain the term "statement". Pick one of the GC-operators and explain how it could be implemented.

Sol: A generalized concordance list is a sets of intervals in which no interval in the set may have another interval from the set nested within it, which means Generalized concordance lists (GC-lists) are sets of intervals that do not have any nested intervals within them.

The query “Return all intervals that begin with a <problem> and end with a </problem>” can be expressed using the before and then contained in operator ….

we can construct the query for intervals that start with <problem>, end with </problem>, and contain 'statement' in this way

1: We first need to find intervals that represent a problem, which we can express using the "Before" operator:

("<problem>" ... "</problem>")

2: Next, we want to filter these intervals to only those that contain the term 'statement'. This can be done using the "Contained in" operator:

"statement" ⊲ ("<problem>" ... "</problem>")

3: Putting it all together, the query in GC-operators would look like this:

"statement" ⊲ ("<problem>" ... "</problem>")

pi(S) = takes starting point of all intervals in the GC list S

Ex: pi([3,4], [5,8],[6, 9]) = ([3,3],[5,5],[6,6])

Implementation

pi(S){

resultGC={}

for(i to n ) in S//for all intervals in S

resultGC.add(S[i][0],S[i][0]);

Return resultGC

}

1. What is Gap Compression? What is LLRUN? What is a Golomb code? Make sure to go into details. Give an example for each explanation.

Sol: **Gap compression**:

Gap compression is a technique to reduce the storage space required for inverted indices. The idea is to store the difference between consecutive values (called gaps) rather than absolute values themselves.

Example:

If document IDs are: 3, 4, 8, 12, 15 then the gap compressed version of this is: 3, 1, 4, 4, 3.

By storing gaps instead of absolute values, better compression can be achieved.

**LLRUN**:

If the gaps in a given list do not follow a geometric distribution then Golomb codes may not lead to very good results. Huffman coding yielts to optimal compression rates but it is difficult to apply this to a given sequence of (delta values). Instead of applying Huffman coding to the gap values directly it is possible to group gaps of similar size into buckets and have all gaps in the same bucket share code word in the Huffman code.

For eg: if all gaps from the interval [2^j,2^j+1) have approximately the same probability we could create bucekts B0 B1...with Bj [2^j,2^j+1). All the delta values from the same bucket Bj share the same Huffman code Wj

their encoded representation is Wj followed by the J bit representation of the respective gap value as binary number this method is called as LLRUN.

Ex: Let delta values = 2, 3, 4, 8, 9, 16, 17

Buckets:

B0 = [1, 2)

B1 = [2, 4)

B2 = [4, 8)

B3 = [8, 16)

B4 = [16, 32)

Let Huffman codes be:

W0 = 00

W1 = 01

W2 = 10

W3 = 110

W4 = 111

Encoded representation of each gap is:

2 (from B1): 01 followed by 10 (binary for 2) -> 0110

3 (from B1): 01 followed by 11 -> 0111

4 (from B2): 10 followed by 00 (since B2 starts from 4, subtract 4 from gap to get the binary) -> 1000

8 (from B3): 110 followed by 000 (binary for 8, subtract 8 from gap) -> 110000

9 (from B3): 110 followed by 001 (binary for 9, subtract 8 from gap) -> 110001

16 (from B4): 111 followed by 0000 (binary for 16, subtract 16 from gap) -> 1110000

17 (from B4): 111 followed by 0001 (binary for 17, subtract 16 from gap) -> 1110001

**Golomb** **code**:

Suppose the delta values follows a geometric distribution Pr[delta = k] = (1-p)^k-1\*p. If a collection has N documents and the term t appears in Nt documents, the probability p that t occurs in a given document chosen at random is Nt/N.

If all documents are independent of each other, the probability of seeing a gap of size k between 2 subsequent occurrences of t is Pr[delta = k] = (1 – Nt/N)^k-1.(Nt/N).

Most of the gaps fall around a common length. This length is called the modulus. One way to encode the gaps is to:

1) Determine an appropriate modulus

2) Split each delta-value k into 2 components: a quotient q(k) and a remainder r(k) where:

q(k) = (k-1/M), r(k) = (k-1)modM

Encode k by writing q(k) + 1 in unary followed by r(k) as a floor(logM)) bit or ceil[logM] bit number.

For general M, the above code is called a Golomb code.

Ex: If M = 3, and gap k = 4 then:

q(4) = (4-3)/3 = 1 and r(4) = 3mod3 = 0

q(4) + 1 = 2 in unary is 001.

log2(3) = 1.58. Floor is 1, ceil is 2.

If r(k) is written as 2 bits, Golomb code is 00100. If 1 bit, then 0010.

1. Suppose we have m=4 many accumulators. Give examples that would trigger the QUIT and the CONTINUE accumulator pruning heuristics if term at a time processing was being used.

Sol: we have amax =4 ,

QUIT heuristic states that when |Acuurent|>= amax, no new accumulators is created and the result of the current accuulators is the final result and we break from the prog

CONTINUE heuristic states that when |Acuurent|>= amax, no new accumulators is created but we continue to calculate teh score of teh remaining terms and update in the current accumuators

Ex:

We have query apple, banana cherry hello there

Lets Posting list size of the above terms in the inverted index is

Apple : 20

Banana: 40

Cherry: 70

hello : 60

There : 20

As per term ata time processing , we sort the terms of the query in incraetsing orer of their size of PL.

So we have apple, there banana, hello cherry

First apple’s score gets calculated and put to acc 1 as |accurent| + Napple <=amax

then there’s score gets calculated and put to acc 2 as |accurent| + Nthere <=amax

then banana score gets calculated and put to acc 3 as |accurent| + Nbanana <=amax

then hello score gets calculated and put to acc 4 as |accurent| + Nhello <=amax

//lets say |acurrent| + Ncherry > amax and |acurrent| < amax, means some space is still there in the accumulars but not enough to put the entire posting list of Ncherry, then we using pruning to get the best documents for cherry

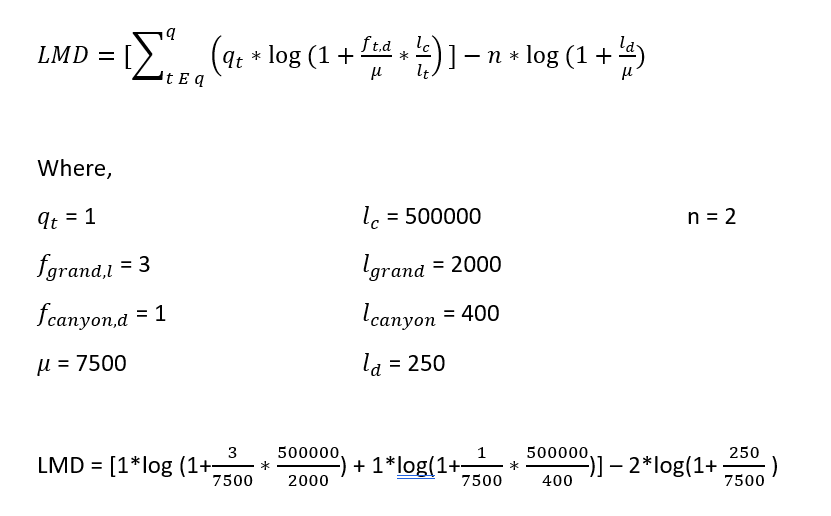
In QUIT:

no new accumulator will be created and we return the final result from the first 4 accumulators score

IN CONTINUE:

no new accumulator will be created and we will continue to calculate the score for cherry for all its documents not only the top ones, if the score for cherry for other terms is better then the accumulators will have the updated scores for this term.

1. Suppose there is a 500,000 word corpus. Document *d* is 250 words long. The terms grand and canyon occur 2000, 400 times respectively. In document *d* grad occurs thrice and canyon once. Let *μ*=7500. What is the LMD score for *d*  for the query "crand canyon"?

Sol: 

1. Derive how the factor P2 is estimated in the DFR formula.

Sol: P2 is the probability that a term might occur at least once more if it has already occured a lot of times in the current document.

Suppose we know an event can take one of two values (say the sun rises in the morning -- it might happen, it might not).

If we have observed that the event has occurred in the last m−1 trials. The odds we assign to it occurring in mth trial in the absence of any additional information is m/m+1.

Because, the rule of succession says since we know both outcomes are possible, it is as if we had an event in addition to our m−1 trials in which the outcome failed, and so we should estimate the probability as: m/m+1.

Using this we can estimate P2 as ft,d/ft,d +1.

1. **Explain how hub and authority scores are calculated in HITS.**

**Sol:** [**final thread**](https://sjsu.instructure.com/courses/1572810/discussion_topics/5103103)

1. Describe where/in what steps a distributed file system might be used in the computation of Page rank as a map reduce job.

Sol: In PageRank algorithm, total three explicit steps need distributed file system (DFS) to store the PageRank field and they are

1-3) After mapReduce computation of A \* current\_r, dangling correction \* current\_r, and teleporter\* current\_r

all pairs (nid, node) are stored in DFS with computed PageRank values from each step

For each mapReduce job, outputs from map functions will be stored into DFS to avoid possible loss.

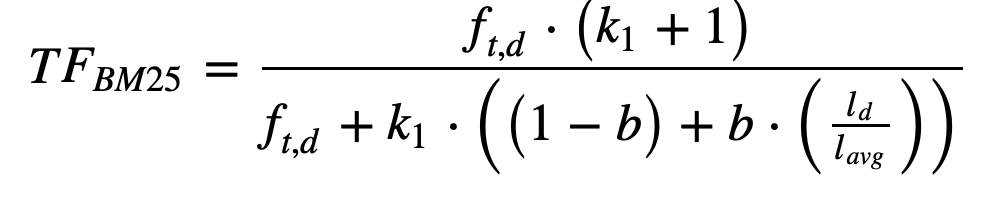
1. Briefly define size specific P@k and aggregate P@k and explain why they might be used.

Sol: Size specific P@k : Instead of finding P@k use P@floor(pn), where p is k/N and n is batch size. Issue: diff batch can have diff number of rel docs still k is same.

* *aggregate p@k* - we chose threshold for score per Batch such that scoring value s > t . T is chosen such that k=pN docs will have score s>t.
* Why used? in traditional method of batch filtering to find P @k, the issues are non-consistent result based on batch size, b) small size batches resulted in |rel|=0 and AP undefined, c) P@k dependent on batch size . So we can use above 2 ways to resolve these issue

PAST years

1. **Prove there is an upper bound that a single term's BM25 score for a single document can contributed to the overall BM25 score for that document with respect to a query.**

Sol**:** We have, 

By L'Hopital's rule for limits, take derivatives of the numerator and denominator with respect to f\_t,d and we get the value k1+1 as the value of the limit. Normally k1=1.2.

1. **Give the context in which one might use accumulator pruning for ranking. Then explain and give an example of how the accumulator pruning algorithm from class works.**

Sol: when there are not enough accumulators to store The posting list of the terms in the query, we use pruning to only put teh most important postings in the accumulator which can contribute to the top k result.

Say we have acurrent as the current accumulator space and amax as the max accumulator that can be given to solve term at a time query processing.

So if acurrent + Nt > amax > acurrent, we use accumulator pruning.

rankTermAtATimeWithPruning(term vector, amax, k, u){

//we sort the term vector in increasing order of PL size

//u is some contant after which we update the threshold value

//we create 2 empty accumulators acc and acc’

for(1 to n we loop for all terms in the term vector){

Nt is posting list of term t[i]

//we check current acc capacity

If |acc|+Nt < amax{

//Then no need to prune

for(each doc d in the PL){

acc[i].d=d

acc[i].score=BM25 score

}

//push all result to acc’ from acc

Else if |acc|+Nt==amax{

//recahed limit so return result

//cal scores of all doc in acc and return acc

}

Else //|acc|+Nt> amax{

//we prune

//consider a threshold value T, set with some initial value

//have some ftd based on which we decode if the document can be added to the acc

//we cal score for each doc and see if that score is better than T and store to acc

}

//we update the threshold after seeing u postings

// so the next time we need to prune , we use the updated threshold

}

//finally we transfer all acc’ score to acc

//and loop fr next term in the query

}

Sort the result basing on acc scores

Return top k documents

}

Ex:

We have query apple, banana cherry hello there

Lets Posting list size of the above terms in the inverted index is

Apple : 20

Banana: 40

Cherry: 70

hello : 60

There : 20

As per term ata time processing , we sort the terms of the query in incraetsing orer of their size of PL.

So we have apple, there banana, hello cherry

First apple’s score gets calculated and put to acc 1 as |accurent| + Napple <=amax

then there’s score gets calculated and put to acc 2 as |accurent| + Nthere <=amax

then banana score gets calculated and put to acc 3 as |accurent| + Nbanana <=amax

then hello score gets calculated and put to acc 4 as |accurent| + Nhello <=amax

//lets say |acurrent| + Ncherry > amax and |acurrent| < amax, means some space is still there in the accumulars but not enough to put the entire posting list of Ncherry, then we using pruning to get the best documents for cherry

In QUIT:

no new accumulator will be created and we return the final result from the first 4 accumulators score

1. **Express the following as region algebra queries: (a) Your first name within 8 terms of your last name. (b) Your zip code in atags before the phrase "doxing is fun".**

Sol: a) ([[8]△[7]△[6]△[5]△[4]△[3]△[2]△[1]) ◁ ("FIRSTNAME" … "LASTNAME")

b) "Doxing is fun" … (("<zipcode>" … "</zipcode>") ▷ "94085")

1. **Give a concrete example involving your name of how a Canonical Huffman code might be written as a preamble to encoding a string.**

Sol: canonical hoffman code means we have the symbols in sorted lex order in every level

{a,2/6} {h, 1/6}, {i, 1/6} , {l, 1/6}, {s, 1/6}

{a, 2/6} {hi, 2/6} {ls 2/6}

{hils, 4/6}

{ahils, 1}

{(a:0), (h:001), (i:101), (l:011), (s:111)}

We already know the symbols in the preamble and as its canonical huffman code, we know the symbols are in lex order, so we can avoid using the symbols in the preamble and can simply put the code

{(0), (100), (101), (110), (111)}

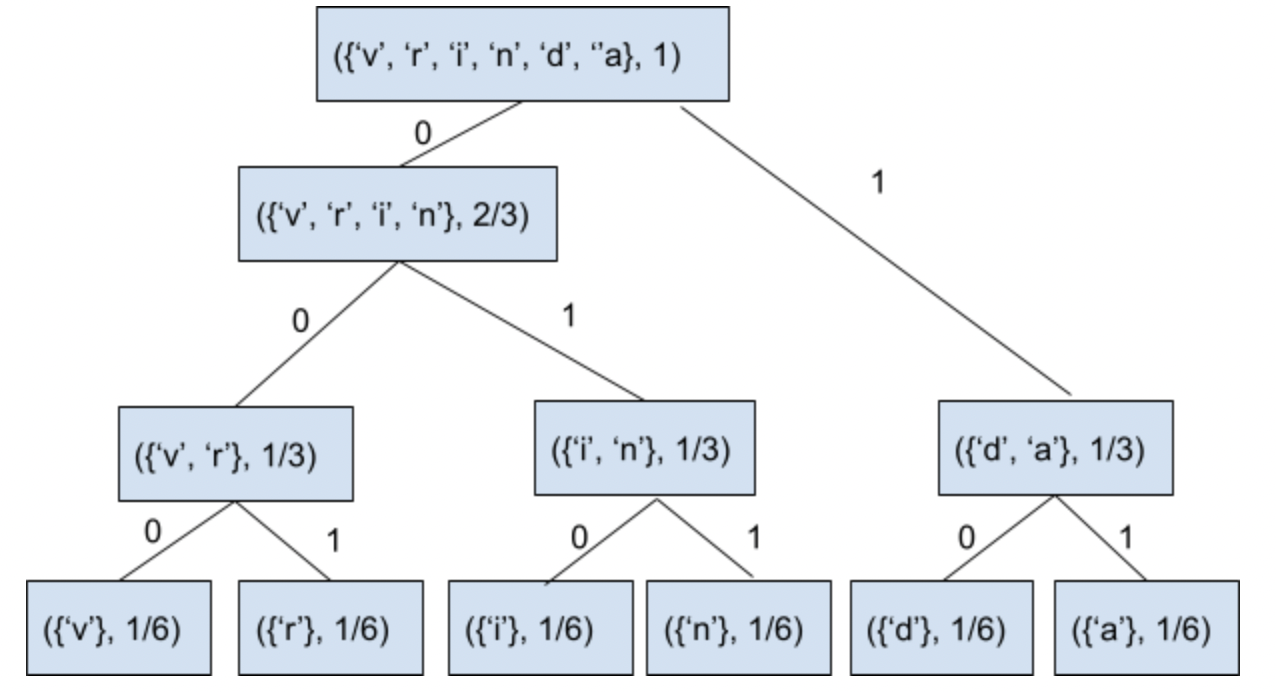
Finally just the length of the codes are written

(1, 2, 3, 3, 3)

Vrinda

Letter probabilities:

({'v'}, 1/6), ({'r'}, 1/6), ({'i'}, 1/6), ({'n'}, 1/6), ({'d'}, 1/6), ({'a'}, 1/6)



Preamble: ((‘v’, 000), (‘r’, 001), (‘i’, 010), (‘n’, 011), (‘d’, 1), , (‘a’, 11))

1. Briefly describe how each of the following coding schemes for compression lists work: (a) γ-code, (b) LLRUN, (c) Rice code.

Sol: a) y-code : its a gap compression technique used to store the inverted indexes gaps in the PL. here the gaps or delta values are converted to unary followed by binary value and we drop the significant 1 bit fro the binary. Example 5 : binary is 101, unary length is 2 so 001, y-code is 001 01

b) LLRUN : its a global parametric gap compression technique used to store inverted index PL that groups gaps in terms of buckets in the form of [2^j,2^j+1), the gaps are stored in teh form of huffman code of teh bucket followed by the binary code of the gap.

c) Rice code: its a local parametric gap compression technique used to store the inverted index using the common length of the gap or modulus M in power of 2 floor and ceil of log M range. This can be used in the gaps are in the form of geometric distribution of the form (1-p)^(k-1) \*p where p is Nt/N

1. Give a situation with numbers where it would make sense to rebuild an index rather than remerge.

Sol: if the deletions are more than insertions in the inverted index, its better to rebuild thna remerge as the remrge will take more time to merge all the documents

It was found that merging that ¾ more time for deleted documents handling

1. **Distinguish intra-query and inter-query parallelism as far as information retrieval goes. What bottlenecks exists to intra-query parallelism if partition-by-document is used.**

Sol: inter-query parallelism:

By creating n replicas of the index and assigning each replica to a separate node, we can realize an n-fold increase of the search engine’s service rate without affecting the time required to process a single query.

This type of parallelism is referred to as inter-query parallelism, as multiple queries can be processed in parallel but each individual query is processed sequentially.

Intra-query parallelism: By splitting the index into n parts, each node works only on its own small part of the index. This approach is called intra-query parallelism. Each query is processed by multiple servers in parallel. This improves the engine’s service rate as well as the average time per query.

Bottlenecks:

Document partitioning works best when the index data on the individual nodes can be stored in main memory.

suppose, queries were on average 3 words and we want the search engine to handle 100 queries per second. Due to queueing effects, we cannot achieve a utilization of more than 50% typically, without experiencing latency jumps.

So a query load of 100qps translates to a required service rate of at least 200 qps. For 3 word queries, this translates to at least 600 random access operations per second. Assuming an average disk latency of 10ms, a single hard disk drive cannot perform more than 100 random access operations per second, one sixth of what we need on each of our nodes.Adding more machines doesn't affect the minimum of what each machine must do, so this is a bottleneck for the document partitioning approach.

1. **Explain how hub and authority scores are calculated in SALSA.**

Sol: we have adjacency matrix for the nodes/documenst in the web, L. we have LT as L transpose. We normalize this matrix wrt row and column and we get Lr, where each row is nromalized and LTc where each column is normalized.

Where xi(k) is the authority score and yi(k) is the hub score

xi(k)=Axi(k-1) and yi(k)=Hyi(k-1)

yi(k)= hub score tells how good the authority are linked to the webpage

xi(k) = authority score tells how many hubs are linked to the webpage

xi(0)= 1/num of webpages

yi(0)=LcTxi(0)

Authority matrix is given by A = LTcLr

Hub matrix is given by H = LrLTc

We then iterate for k times till

||xi(k)-x(i)(k+1)|| <e

||yi(k)-y(i)(k+1)|| <e

1. Briefly define aggregate P@k and aggregate AP and explain why they might be used.

Sol: Aggregate P@K is used for finding P@k for batch filtering where as per score per batch we determine k = pN. The score per document in the batch should exceed the threshold per batch T. why used ? this is used as normal P@k had many baches with |rel| as 0 which gave undefined average precision for the batch and the precision function was not consistent with batch size.

Aggreate AP is used to find the aggregate of the average precision per topic to determine how good the IR system performs. Why used? Many performance measures are used to determine the performance of the IR system , so aggregate AP will give how the IR system behaves, it can help in batch filtering to find the IR system performance by aggregating the AP for various topics in various batch.

1. **Briefly explain and give example of the following (1pt each): (a) sort-based dictionary, (b) per-term index, (c) move-to-front heuristic, (d) merge based index construction.**

Sol: a) sort based dictionary: The dictionary of the inverted index stored in sorted order of the terms and PL. Data is stored in tree structure and is good for prefix search. Terms are stored in contiguous memory location.

Ex:

“A”: 1,2,4,5

“Are” : 5,9,12

“Bear”: 67, 78

Here if i want to get prefix search for a, i get a and are list.

b) per-term index: also called as self indexing where we store like every 5000th posting in the mem and rest block will have a pointer from this posting which will contain remaining i-5000i terms in disk. When needed to search , we bring the block of 5000 postings to mem , and search.

“The” ->1, 5001, 10001 : in mem

The posting from 1 will be pointed to 2 to 5000 postig from the : in disk

The posting from 5001 will be pointed to 5002 to 10000 postig from the : in disk

c) move to front heuristic : Hash based dictionary uses this heuriistic for while collision happens. In this, when a term is searched, the term is brought to the fron of the list in the hash map, as there is chance of seeing the term again.

Ex: let 0 1 2 be hashes and we add documents %3 to the hash maps

(dociid, pos in the document)

0 -> (2:3), (3:5)

1 ->

2->

When same document has same term in different pos (3,9) then this happens

0 -> (3:5,9), (2: 3)

1 ->

2->

d) merge based index construction : Index construction where we store inverted indexes in disk partitions once they reach a memory limit in the memory index creation process. Then after no terms needs to be put to the inverted idnex, we use disk merge to merge all the disk paritions to 1 inverted index.

Ex:

Say memory limit is 2

The: 2, 4 // in mem

We create a disk partition for the inverted index

//disk

Partition: 1

The: 2, 4

Then the: 5, 6

Another partition is created in disk

Partition: 2

The: 5,6

Say no more terms to index

We merge all disk partition

P1 and P2

The: 2, 4, 5, 6

1. Prove limft,d→∞TFBM25(t,d)=k1+1.**Briefly explain the Maxscore heuristic as used in document at a time query processing.**

Sol: part 1 done above.

Maxscore heuristic states when a document’s score for a query term t exceeds 2.2 log(N/Nt) then we drop the term from the query and score teh document as per other query terms and later add the term t’s score to the doc score. If the score of the document doesnt come good from oter query terms then the documents is not good enough to be in top k results. This helps to reduce the calculation process by limiting the contribution a term to the document score.

1. Give an example of a parametric and of a non-parametric code suitable for GAP compression. **Briefly explain and give an example of vByte and Simple-9.**

Sol: parametric : can give golomb code/ rice code, LLRUN

Non parametric : y, w, delta code

vByte : byte type encoding of gaps of the posting list. Here we use the first bit of the byte to show if the next 7 bits are the last to contain the payload or the gap value will be carried t the next byte.

Ex 5: 101

0 0000101 vByte

Simple 9 is word byte encoding of the gaps in the inverted index PL. where we use the first 4 bits of the 32 bits to show how many chunks will the nxt 28 bits be divided into the respreent the payload.

Ex: if 0000 first 4 bits are 0 then next 28 bits can have 1 bit information each.

1. **Explain what kind of incremental updates logarithmic merging is a suitable algorithm for. Explain how this algorithm works.**

Sol: Incremental updates that are suitable for logarithmic merging is when we have little amount of ram and enough disk space to write to. Merging Indexes are also more suitable towards updates and are not as efficient with frequent deletions. When the indexing process runs out of memory, it writes it make an on disk partition with the label 1. Any existing duplicated labels are merged and the resulting merged index gets a label +1. This process is repeated until there are not duplicate labeled index partitions.

Unordered list item We can speed up the merging process by looking ahead and merging together. If you see that you will have to merge twice in a row, you can merge all of the three partitions in one operation.

1. **Explain one method to estimate P1 and one method to estimate P2 in the divergence-from-randomness approach to coming up with a relevance measure.**

Sol: P1 is the self information measure which states how many times a term appears ina document and log P1 gives the bits fo information needed to store this data.

We can state that Nt1+Nt2+Ntn = Nt, means the term is present in Nt docs and they are distributed in some fashion. We use binomial coeffints to solve this. # ways to chose document d for term t = (N +lt -1)C (lt) \*. lt is length of PL terms in the corpus.

Given this we can find the number of ways to find distribution of term t in N-1 documents

(N-1 + lt-ftd -1) C (lt-ftd) \*\*

We can find P1 using \*\*/\*

From this -log P1 = log (1 + lt/N) + ftd log(1 + N/lt)

P2 is called the elitness part of the document measure. If a term ispresnr in the document ftd-1 times then there is a possibility to get one more occurrence of the term t in the document d, this is given by the formula m/m+1 or ftd/1 +ftd

1. **What is Batch filtering? What is aggregate P@k?**

Sol: Batch filtering : processing of accumulating incoming documents to batches and processing inverted index and ranking them. This is used when there can be a delay in filtering process. Each batch is processed differently for rank, score P@k etc.

1. **What file does the trec\_eval software take as input? Briefly give the format of each. What kind of output does it produce?**

Sol: trec eval software takes 2 files, top and rel where top file is the ranking of the documents along with the score calculated per document in the top k results using some ranking method. And rel file is used to give a manual relevance of each document in the corpus.

Trec\_rel file format : qid iter docno rel

Trec\_top format : qid iter docno rank sim run\_id

The trec eval software then runs the files and provides P@10 , MAP score and more based on the result.

1. Suppose 'a' occurs with probability 0.65 and 'b' with probability 0.35. Suppose our symbols are bi-grams of these two letters. Draw the Huffman tree for these bi-grams. What would the preamble for the canonical Huffman code look like?

Sol: as we are using bigrams

prob of ab = 0.65\*0.35 =.2275

Prob of aa = 0.65 \* 0.65=.4225

Prob of bb = 0.35 \* 0.35=.1225

Prob of ba = 0.65 \* 0.35=.2275

Huffman tree

Aa, ab, ba, bb :1

Ab, ba, bb: .5775

ba, bb:.35

aa ab ba bb

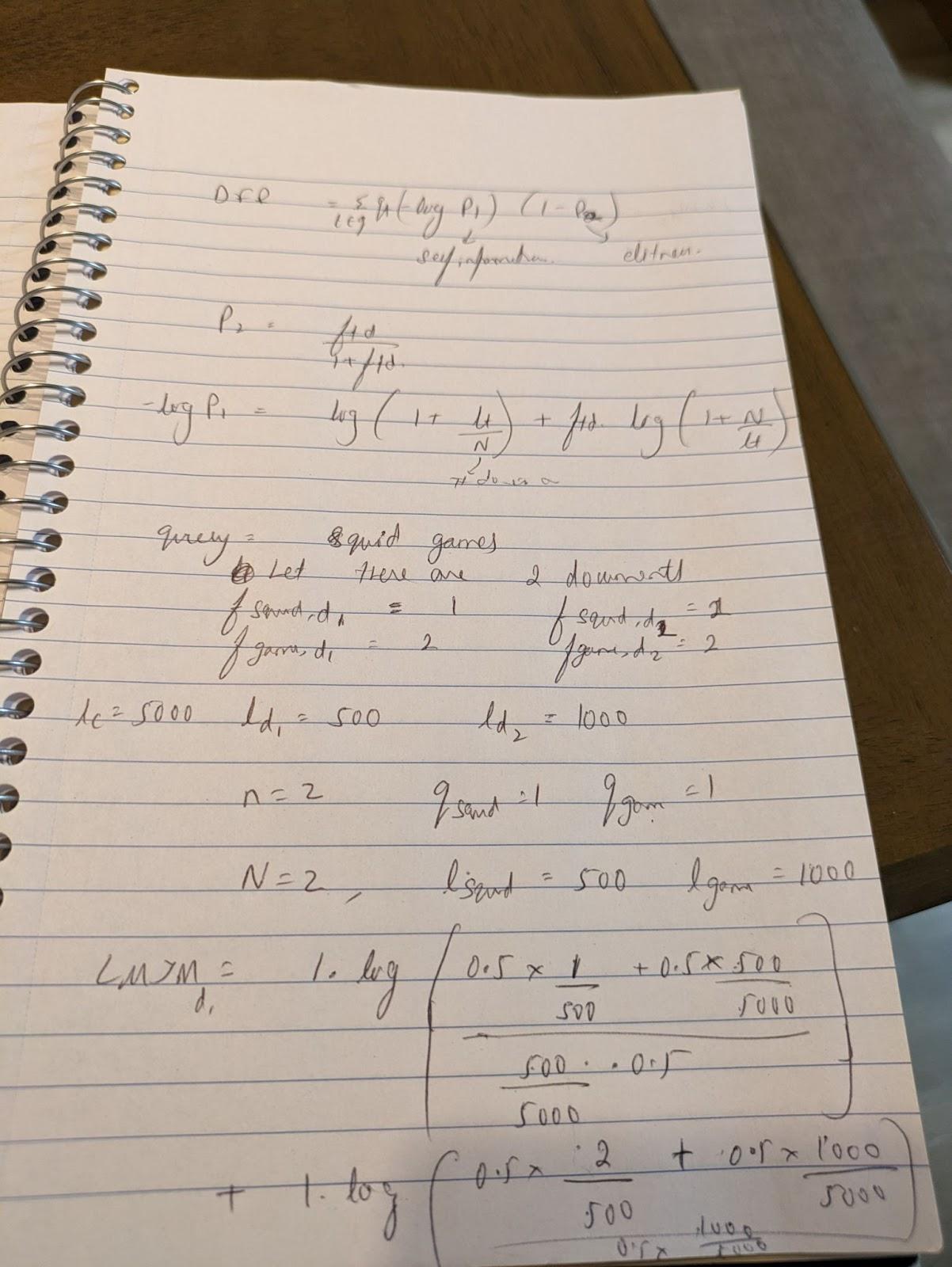
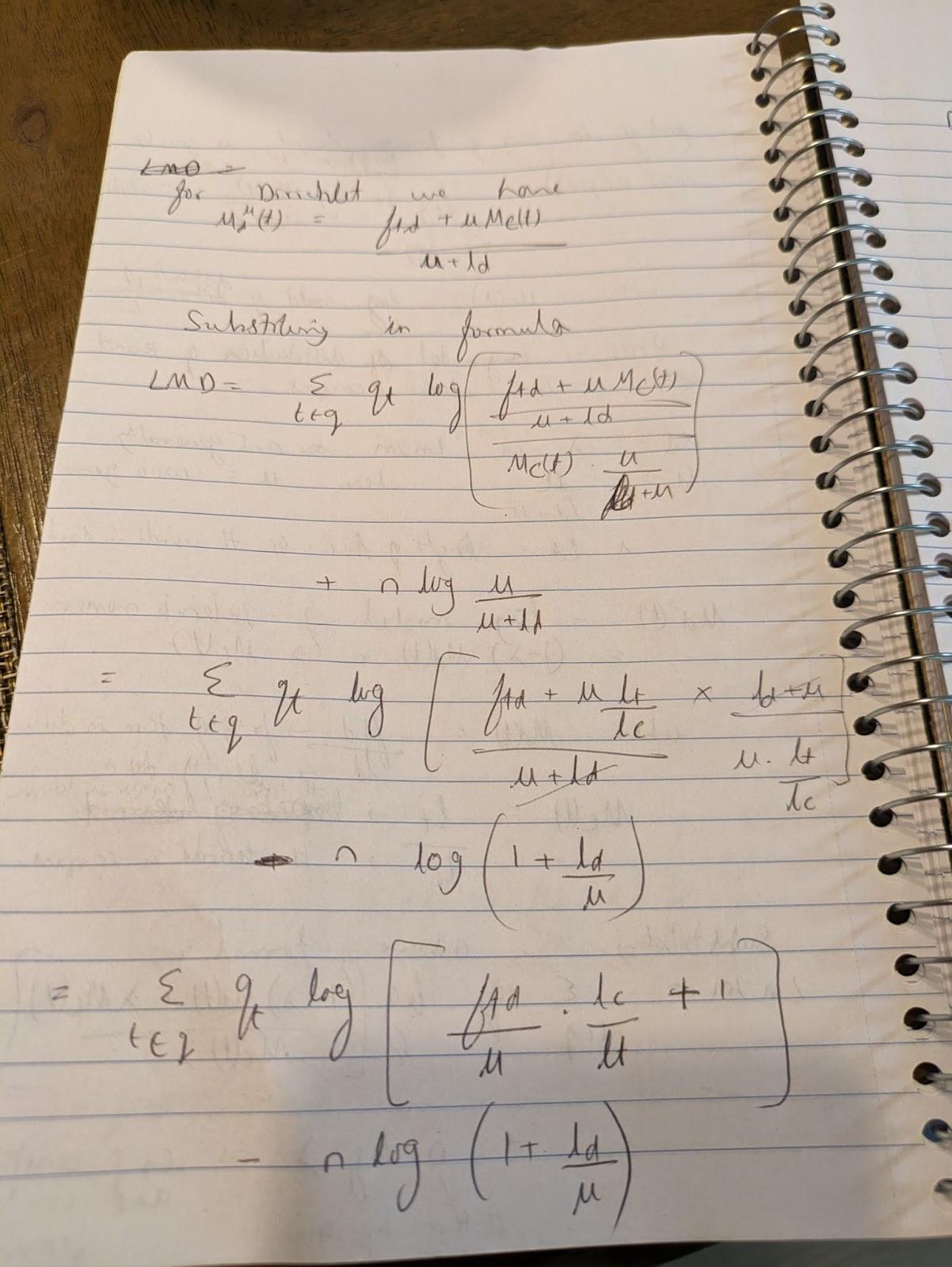
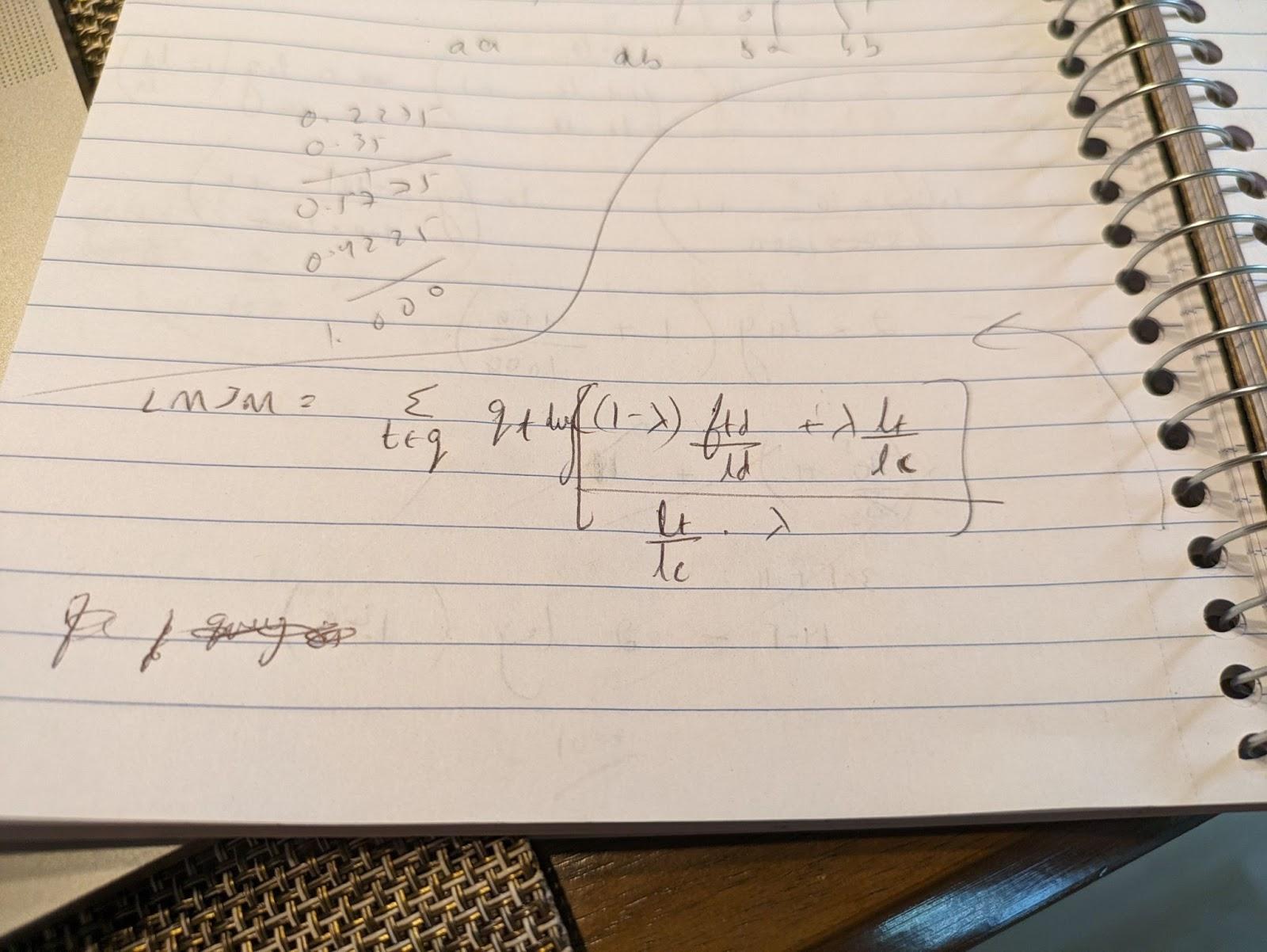
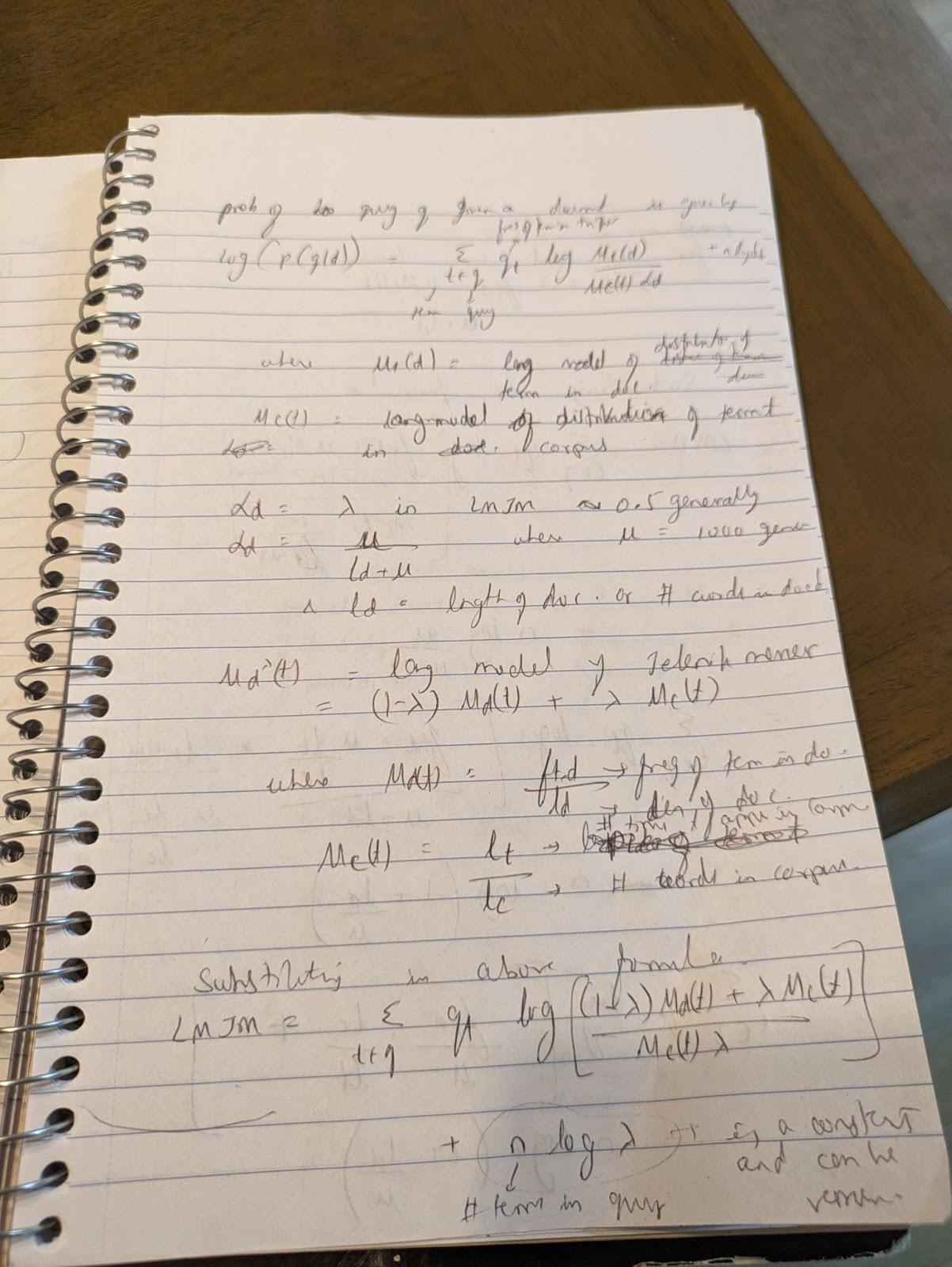
Canonical form is

Aa ab ba bb : in lex order

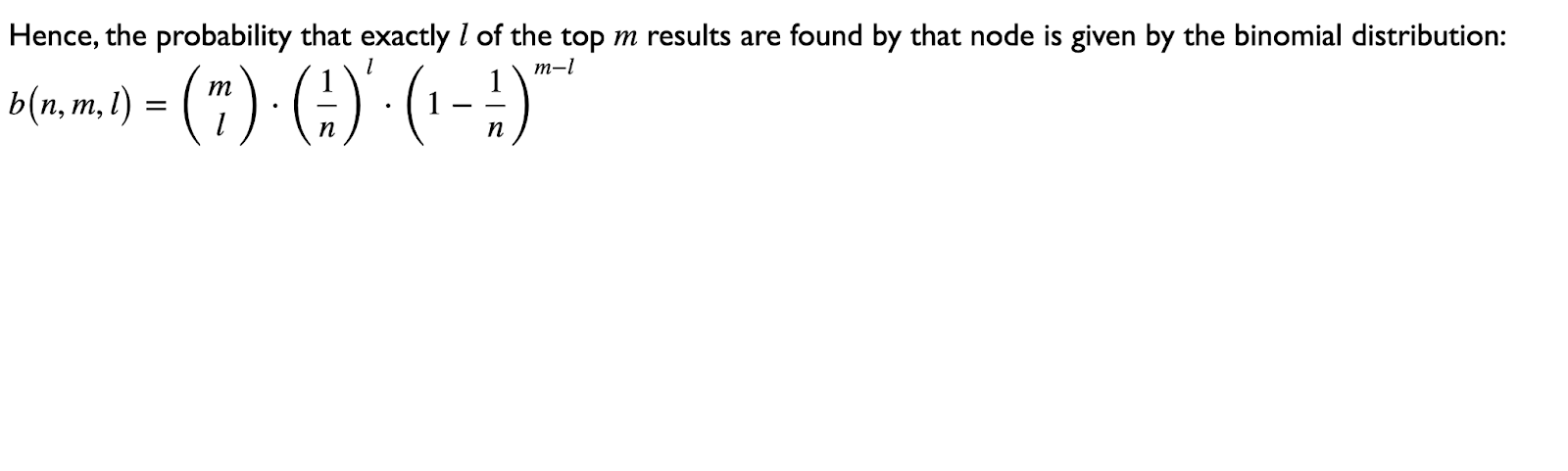
Preamble : (0, 10, 110, 111) and its prefix , we just write the lengths of the codes

(1, 2, 3, 3). Prefix free codes.

1. **Give the following ranking functions together with common values for any magic numbers they have: (a)LMD, (b) LMJM, (c) DFR. Define each of the variables your formulas have and give an example of using them.**

Sol: 

1. Suppose we are using document partitioning to split our index across 5 machines. When a user enters a query our search engines returns the top 10 results. What is the probability machine 1 has half of the top 10 results?

Sol: 

Where n=5, m=10, l=5

b(n,m,l) = (10 5) (1/5)^5 (1- 1/5)^(10-5)

1. Briefly explain how the page rank algorithm might be implemented as a map reduce job.

Sol: Let rbar be the vector for page rank across all documents in the corpus. A is a normalized adjacency matrix. We have rbar= Arbar.

Node has the page rank field for the given node id

Page rank algo needs to calculate Arcurrent and store the result of the (nid, node ) in DFS in the mapreduce job

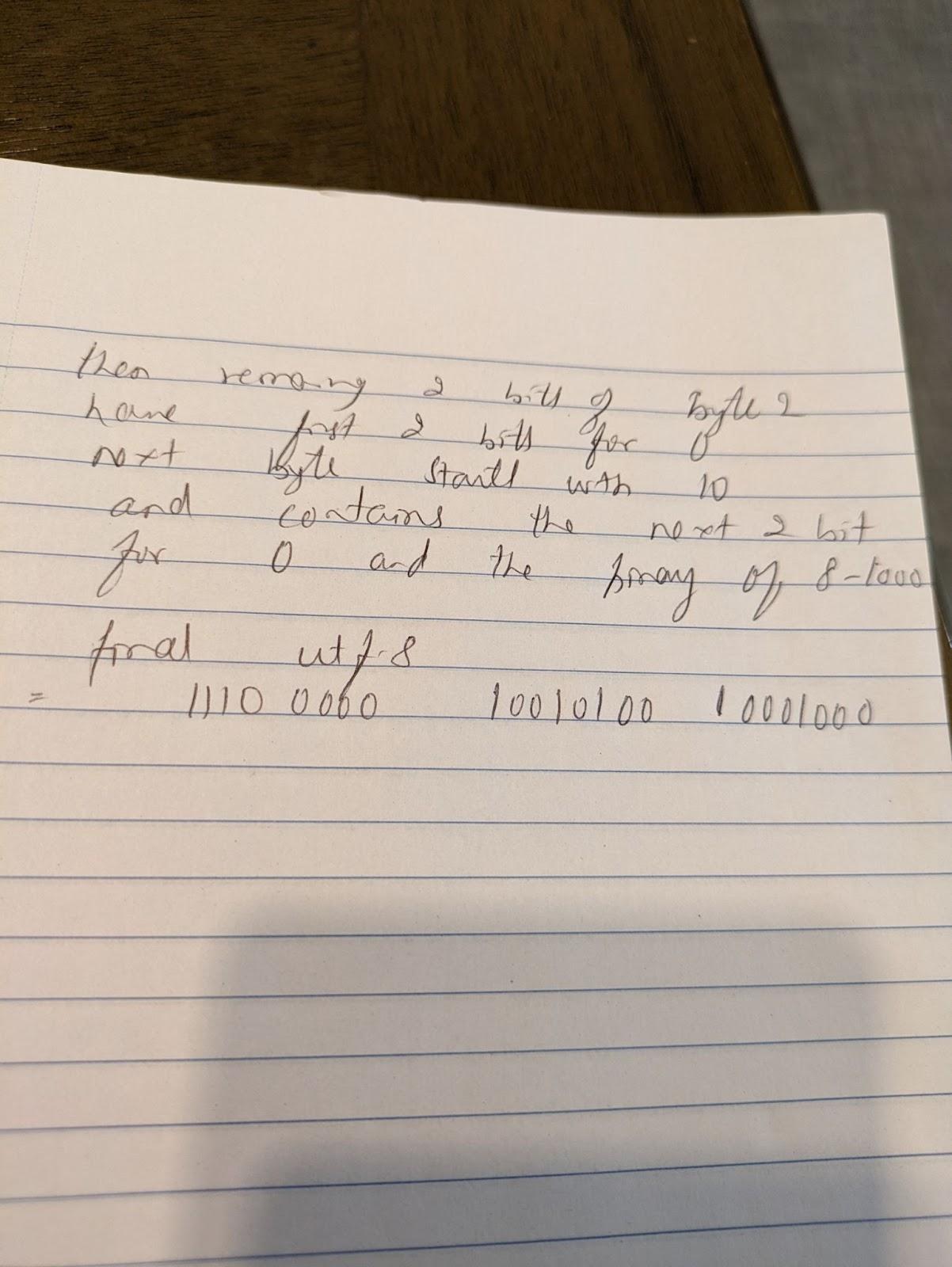
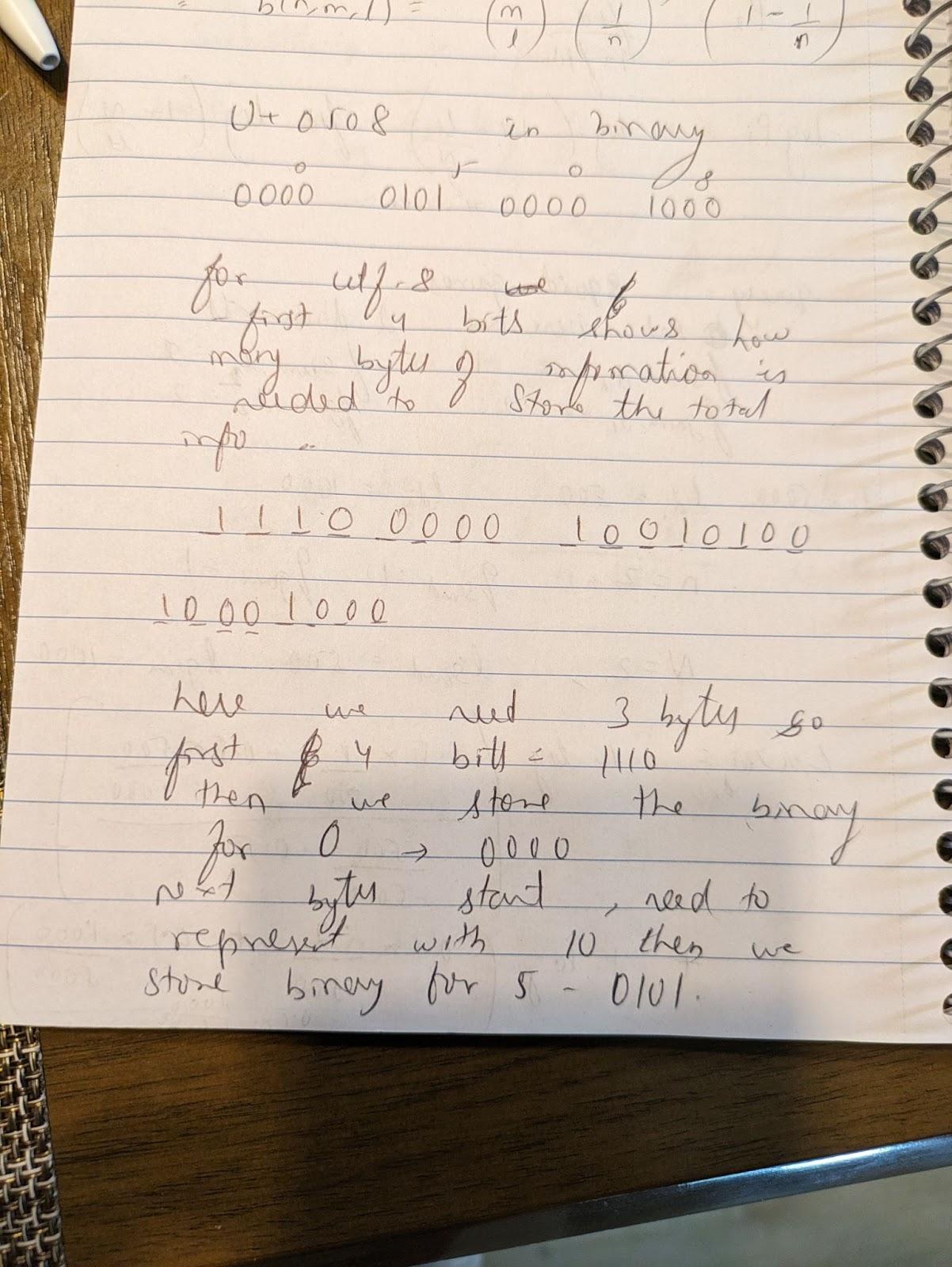
Then the dangling matrix correction is needed where we multiply dangling matrix to recurrent and store the (nid, node) in DFS in mapreduce

The teleporter matrix is multiplied to the rcurrent and we store the (nid, node) generated out of the result in DFS in mapreduce

Then all the 3 results from these are merged to give the final page rank using the map reduce job in the reduce phase using nid as the key and nodes from all the 3 as values.

We continue until ||rcurrent-rold|| < e

1. How would the codepoint Ԉ, U+0508, be encoded in UTF-8?



1110 0000 10 0101 00 10 00 1000

1. **Briefly explain the following concepts: (a) per-term index and (b) dictionary interleaving**

Sol: a) done above

b) dictionary interleaving: store few dic terms and pointers to remaining terms in disk, in disk term and posting list will be appended together. Search down between 2 closest terms and binary search for posting list between pointers is searched for.

Ex:

//in memory

shake shakespeare

1 400

//in disk

Shake: 1, 4, 7, 22, shaker: 76,90, shakespeare: 400, 567

1. Explain and give a small example of how sort based index construction works.

Sol: we get each token and check in the dic if the term is there , if yes, we apend the term pos in the PL else we create a new term in the dic and add the postion to the pL. We repeat this until we have all the terms in the inverted index.

Then we sort the index wrt terms in ascending order.

For each term in the index, we store the PL in disk

Then delete the PL from mem

Then we store the dic terms to disk and delete it from in mem.

Ex:

The -> 23, 45

And -> 67, 89, 100

do-> 300, 450

The above index is created in mem,

We then sort the index wrt terms

And-> 67, 89, 100

Do -> 300, 450

The -> 23, 45

Then we store the PL per term in disk

67, 89, 1000

300, 450

23, 45

Then we store the terms in the disk

And-> 67, 89, 100

Do -> 300, 450

The -> 23, 45

1. Write down the BM25 formula. Give an example of using it for a query with at least two terms.

Sol: BM25= score is for all terms in the query, we do sum of (TFBM25. IDF)

IDF = log (N/Nt)

TF = (ftd \* (k1+1))/ (ftd + k1((1-b) + b(ld/lavg)))

Where is k1 is 1.2 n b =0.75

Query: the heat

Lets N=2

Nthe= 2

Nheat=1

fthe,d1= 2

Fthe, d2=1

fheat,1=0

Fheat, 2=1

ld=lavg

BM25 d1 = log(2/2) \* (2\*2.2/(2 + 1.2))

BM25 d2 = log(2/2) \* (1\*2.2/(1 + 1.2)) + log(2/1) \* (1\*2.2/(1 + 1.2))

So here d2 has a better score than d1 as it has both the terms

1. Write the following query using the region algebra: Lines spoken by Caesar in the play Julius Caesar.

Sol: ((<LINE>...</LINE>) <| ((<SPEECH>...</SPEECH>)|>((<SPEAKER>...</SPEAKER>) |> “Caesar”))) <| ((<PLAY>...</PLAY>) |> “Julius Caesar”))

1. Explain how each γ and Golomb code work to encode gaps. Give an example of how these codes could be used to encode the number 5. State any assumptions you use.

Sol: y code 5 : 001 101 -> selector in unary and body in binary and we drop the first bit in the body = 001 01

GOlomb code Lets M=4, k=5, assume M = 4 and we have the gaps in the form of Pr(delta=k)=(1-p)^(k-1).p where p=Nt/N

q(k)= k-1/M = 4/4 =1

r(k) = k-1%M = 4%4=0

q(k)+1 in unary followed by r(k) in floor or ceiling of log M bits

2 in unary = 001

Log M = log(4) = 2

Floor and ceil are same , we represent 0 in 2 bits

Golomb code = 00100

1. **Explain how document length normalization is incorporated into the basic DFR formula.**

Sol: the documents in the corpus are not of same length so we normalizez the document length so taht contribution of teh document to the query is in accordance with the length of tyej document. If a document has less number f words, and it has the query term , then it should have better score than a document with more words and having the query term.

F’t,d = ft,d log (1+lavg/ld) where lavg is the avg length of all documents in the corpus and ld is length of current document

Ex: ld1 = 10 lavg=20 ld2=20, let ftd be ssame in both documents

F’td1 = ftd.log(1+ 20/10) = ftd.log(3)

F’td2 = ftd.log(1+ 20/20) = ftd.log(2)

So d1 gets better relevance in caculation of DFR.

1. Define the terms and give an example (a) document partitioning, (b) term partitioning.

Sol: a) intra query parallelism don by paritionaing teh invert index wrt documents such that inverted index has partitions where each partitions has some set of documents in it is document partitioning.

Ex: Pariotjn 1 t1 -> 2, 4, 5 pariotnn 2 t1 -> 11, 45, 67

T2 -> 6, 9, 10 t2 -> 59, 490

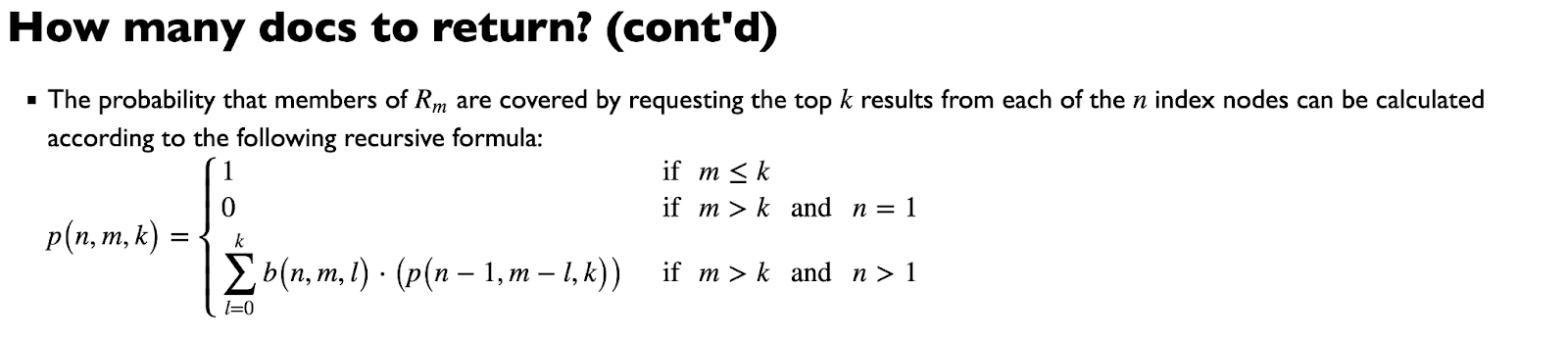
b) Trem partitioning: Process to intra query parallelism doen by paritoning the inverted index wrt terms so that each pariotion has a set of terms.

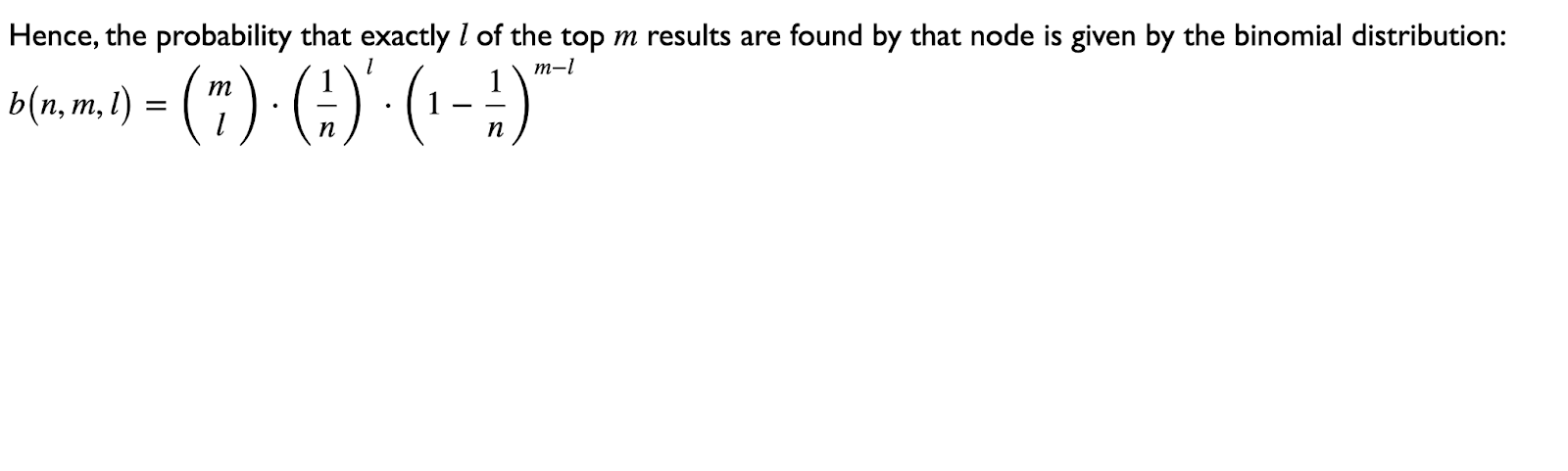
Partition 1: t1 -> 2, 4, 5, 11, 45, 67

Partition 2: t2-> 6,9, 10, 59, 490

1. Suppose we are computing queries using a parallel document partitioned set-up with 2 servers each returning 2 results, where we want the top 3 results. What are the odds that we succeed? (Work out using the recursive formula.)

Sol: n=2 servers, k=2 , m=3





=b(2,3,0).p(1,3,2) + b(2,3,1).p(1,2,2)= 0 + b(2,3,1).1 = 3/8

1. **Explain how query processing can be done in the term partitioning set-up and where one can have intra-query parallelism.**

Sol : The invreted index is paritonined wrt terms. So each partition will have a subset of terms.

a)When a query is sent to the receptionist( the controller that decides which partition the query terms be sent to) , it decides to send the first query term to the partition that contains teh terms and

b) the scoring s done for that term for alk documents in that parition that contains the term and result is stored in accumulator and send to receptinis,

c) receptionist the send the next query term to the next partition that does b)

d) finally all the query terms are processed and the receptionist meges these scores and provide the top k documents in descending orer of score

This can be modified by sending multiple query terms to different partitions parallely so that time to process a query is reduced and the resul from each of their accumulators is merged and the final set of documents is sent to the user.

1. Give a map reduce algoritm which takes as inputs (docid, documnent) tuples and output (term\_id, num\_occurrences\_term\*num\_occurrences\_term) pairs.

Sol: map(docid, document)

For each token in the document

output (token,1)

return

reduce(token, <v1, v2, v3…vn>)

count=0;

For i 1 to n

count+=vi

Return (token, count\*count)

1. Briefly explain how the OPIC Document Quality Measure works.

Sol: OPIC is a method of ranking documents .

a seed page is selected and provided as 10 dollar

The seed page say has 10 links to 10 different documents so the 10 dollars if divided by 10, so each link will get 1 dollar ecah

Each of these pages/ documents are put to priorityqueue which has page amount till now, amount since last use of page and sorted based on amount provided to them

The most important document is with most amount. If there is a link to this page from some other page, then amount of this page is increased

When a pae is polled from the queue, the second value that is amount since last used if made to 0.

This is done for all pages in teh queue and based on the links and mount finally stored per page the pages are ranked.

1. Define and explain the dangling node matrix and teleporter matrix components of the Google matrix.

Sol: A document that has no link to any other document shows as 0 valued rw in the adjacency matrix of documents link, called a dangling node. To resolve this , a dangling node matrix is used to multiply with the adjacency matrix, which would state that node can travel to any other node in the web..

The above resolution doesnt converge the matrix A , so Teleporter matrix is used to make sure a page if has no link to amy page , can teleport to any other page in the web.

Using **random surfer model**, we can decide that A will converge means if a page has no connected pages, then we assume its connected every other pages and can travel to any other pages. That page is a dangling node and we create A’bar stochastic using : A’bar = Abar + abar ((1/n)e^T), ai is 1 if i is dangling node. **dangling node matrix** abar ((1/n)e^T). To make Abar converge , we need below equation where H is a **teleporter matrix w**ith all entries as 1/num of total web pages, says a page can randomly jump to any other page and we get A’bar = aAbar + (1-a)Hbar

1. Express the following using our operators for GC-Lists. The contents of the body of any html document whose head contains at least two meta tags.

Sol: (<body>...</body>) <| (<html>...<html>) |> ((<h1>...</h1>) <| (<head>...</head>) |> (<p>...</p>))

1. Consider the list L=⟨5,8,15,25⟩Compute its Δ-list, then encode it using a γ-code.

Sol: Δ : <5,3,7,10>

Y code = unary of length of binary followed by binary and 1 is dropped from the binary

5= 001 01

3= 01 1

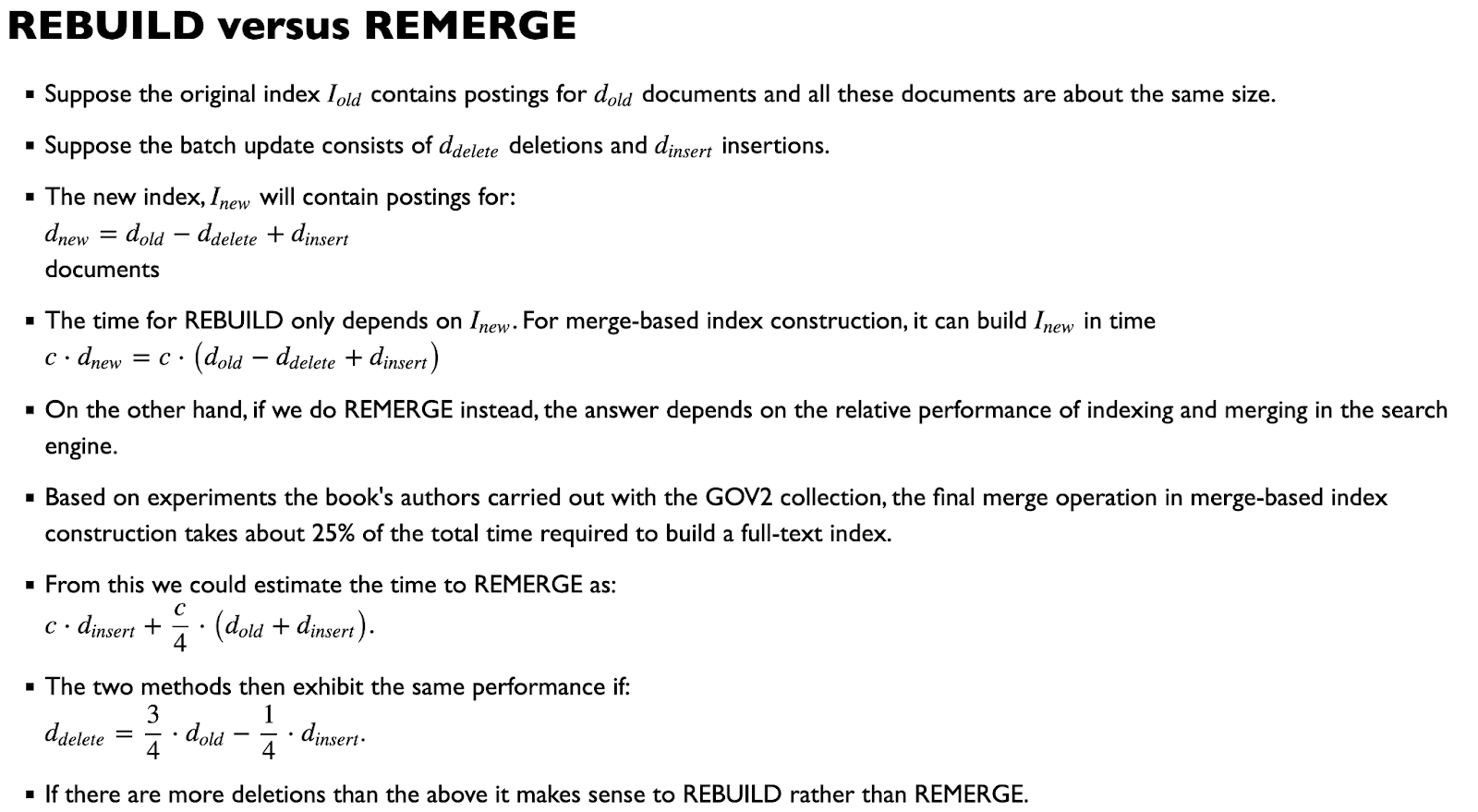
7= 001 11

10=0001 010

1. **Define the REBUILD and REMERGE batch update operations**. Give a concrete (non-trivial, i.e., some work is done) situation in which they would both have the same performance.

Sol: Keeping all the insert/deletion to the inverted index in a new inverted index and recreating the entire inverted index is rebuild. less code and faster

Keeping all insetion/deletion to the inverted index and merging to the old inevrted index these updates, is called remerge. Works better if fewer deletions



1. **Describe how the HITS algorithm measures the quality of a document. How does SALSA differ from HITS?**

Sol: part 1 done above

Part 2: SALSA uses normalized adjacency matrix LTc wrt column nd Lr wrt row and its immune to topic drift.

1. Explain the dictionary-as-a-string to storing an inverted indexes dictionary. Give one situation in which a sort-based dictionary might be preferable to a hash-based one, and vice-versa.

Sol: dictionary-as-a-string approach : as sort based dic storage needs the entries to be in sorted order and we don’t want to waste space, we store primary array with offset integers in sorted order that points to secondary array that actually has the terms and the pointers to posting list.

- 2 adjacent elements in primary array will help to tell the length of term in secondary array, so we don’t store the null element too

- works good for prefix lookups

Sorted based dic is good for prefix lookup as all terms are stored in ascending order in tree structure, so its easy for prefix match and lookup while hash based dic wont be able to help much here as we need to check per hash map terms to get the prefix match as the terms are not stored in contiguous fashion.

Hash based dic is good for single term search as it saves a lof of spce and time with good collision resolve heuristic like move to frint and insert at back. Non tree traversal required here, so very quick.

1. Give an example situation in which the BM25F score (assuming k\_1 =1.2 b=0.75) for a document for a multi-word query is lower than its cosine score. Then give a situation in which the opposite is true. Do the same for proximity scores.

Sol: give an example where terms of the query are in body and have a weight of say 0.2 , so here BM25F has less score and second example where terms of the query are in the heading and has a weight of 10 here score is more for BM25F

1. Let A be the GC-list of intervals [2i, 2i+1] where i between 0 and 100 inclusive. Let B be the set GC-list of intervals [2i+1, 2i+2] where i is between 0 and 100 inclusive. For each of the region algebra binary operator, op, say what the GC-list (A op B) would be.

Sol: A : [0,1], [2, 3], [4, 5], [6, 7], …[200, 201]

B: [1,2], [3, 4], [5, 6], [7, 8]....[201, 202]

Binary operators are contains in <| not contains in , and or and …

A Δ B : G(S)=intervals in S that contains interval in A and B = {}, as no interval in A and B that are common.

A ∇ B : G(S)=intervals in S that contains interval in A or B = [2i, 2i+2] for all i = 0 to 100

A…B : G(S) = interval in S that contains interval which starts with A interval and endds with B interval such that A’s interval ends before B’s interval starts.

= [2i,2i+2] for all i =0 to 100

A |> B: intervals in A that contains intervals in B, is {}

A not |> B intervals in A that doesnt contain intervals in B, is A

A <| B, intervals in A that are contained in B, is {}

A not <| B , intervals in A that are not contained in B, is A

1. What is a prefix-free code, what is an optimal code. **What is the Source Coding Theoem.**

Sol**: a)**When we encode some symbols in a list and any code is not a prefix of some other code in the list then its called prefix free code.

Ex: in huffman coding say symbols are a , b , c

C(a)=0

C(b)=10

C(c)=11

Here no code is a prefix of the other so we can easily decode to the original symbol given the code

b) optimal code :optimal code given symbols M(a) = 1/2^li where li= |C(a)| = depth(C(a)) = - log(M(a))

Avg bits per symbol doesn’t exceed H(A) = summation ai of i to n -(M(ai)log(M(ai))) called entropy for symbol source S.

c) **Source Coding Theorem**. Given a symbol source S, emitting symbols from an alphabet A according to a probability distribution PA, a sequence of symbols cannot be compressed to consume less than

H(A)=−∑PA(σ)⋅log(PA(σ)) bits per symbol on average. Here H(A) is called the **entropy** of the symbol source S.

1. Suppose Pr(a) = 2/3 and Pr(b) = 1/3. How would the string abba be compressed using arithmetic coding?

Sol: arithmetic encoding : a) converts msg to intervals b) encode the binary intervals so created using simple bit seq

Here we have a message of 4 symbols so we create interval with 4 symbols in lex order like

Aaaa aaab aaba abbb… with each seq as prob of prob of give per symbol prob

Sum of prob all the seq symbols =1

We consider the interval where abba lies say I= [y, y+p), we use binary intervals in the form of I’ = [x, x+2^-q), where I is a subset of I’.

Then we encode this binary interval in bit seq

Example : let abba lies in interval

Seq before abba is abaa = ⅔\*⅓\*⅔ \*⅔ = 8/81

Abba = ⅔\*⅓ \* ⅓ \*⅔ = 4/81

So interval is [4/81, 8/81) = it will be converted to binary interval say [0.565, 0.625)

And we can represent it using bit seq like 0110, so we compressed the string using arthimetrc compression.

1. Suppose a term appears in roughly 1/2 of the documents. What would be the optimal modulus to choose for a Golomb code to compress its posting list?

Sol: Nt=1/2 N .If k=delta or gap value in hhe PL of the document for term t, to represent k+M\* integer we need extra 1 bit infor

P[k+M\*]= ½ P[k] and we get M\*= -log2/log(1-Nt/N) = -log 2/ log(1/2)

Golomb we use either floor or ceil of M\*.

1. Give a situation with numbers where it would make sense to rebuild an index rather than remerge.

Sol:for rebuild time taken to create new Index=

c(dnew) = c(dold + dinsert - ddelete)

For remerge we take 25% more time for merge step than creating whole new index.

c(dnew) = c(dinsert) + c/4 (dold + dinsert)

ddelete = ¾ dold - ¼ insert, then remerge and rebuild have equal performance

If ddelete = 12 and dold = 16, insert =4

Then c(dnew) rebuild is c(16 + 4 - 12) = c(8)

c(dnew) reimerge = c(4) + c/4(16 + 4) = c(4) + 5c = 9c

As u can see above remge takes more time than rebuild when delete is much more than insert.

1. **Briefly explain (a) BM25F, (b) Pseudo-relevance feedback.**

Sol: a) BM25 with field weights : document parts have different weights and score is calculated only on those parts basis and then score summed up.

b) we try to extend the query. We first take initial query, get top m results , then up to some term selection value, terms are added to the query based on relevance to the result.

1. In calculating BM25 what components might need to be pre-computed before query time? Of these which components might benefit from being calculated using a map reduce algorithm. Pick one of these and give a map reduce algorithm for it.

Sol: Document Length (avgdl): The average length of documents in the collection. Calculating this beforehand can improve efficiency during query processing.

1. Inverse Document Frequency (IDF): The weight assigned to each query term based on its rarity in the entire document collection. Pre-computing IDF values can reduce query time overhead.
2. Term Frequencies (TF): The number of times a query term appears in a document. Storing pre-computed TF values can speed up scoring during retrieval.
3. Free Parameters (b and k1): These parameters affect the BM25 score calculation. Pre-computing their optimal values can enhance query performance.

Among these, let’s focus on IDF. Calculating IDF using a map-reduce algorithm can be beneficial, especially for large-scale collections. Here’s a high-level outline of a map-reduce approach for computing IDF:

Map Phase:

* 1. Input: Document collection (split into chunks).
  2. For each document chunk:
     1. Tokenize the text into terms.
     2. Emit (term, 1) pairs for each term encountered.

Reduce Phase:

* 1. Input: Intermediate (term, count) pairs.
  2. For each unique term:
     1. Sum up the counts across all documents.
     2. Compute IDF using the formula: [ \text{IDF}(t) = \log\left(\frac{N}{\text{df}(t)}\right) ] where (N) is the total number of documents, and (\text{df}(t)) is the number of documents containing term (t).

Output:

* 1. A dictionary or key-value store with term-to-IDF mappings.

1. Briefly describe the Traffic Rank linear optimization problem.

Sol: count of number of webpages to a webpage determines the rank of the webpage. More the inut to the webpage , more popular the page and traffic rank is more. maximize entropy function -sum of all i,j(pij log pij), where pij is prob of all the incoming traffic to j from i. 4 constraints : pij>0 ; Sum i,j (pij)=1 ; sumi(pij) - sumj(pji)=0 ; pij =0 if no traffic to j from any i

- more incoming traffic to a page , more is its traffic rank

1. Ignoring dangling nodes and the random jump factor give the mapper and reducer for a Map Reduce implementation of page rank.

Sol:

map(nid, node N){

P = N/|A|

emit (nid, N)

For all nodes m in adj list

emit(node id m, p)

}

reduce(nid, <p1, p2, ..pn>){

For all p in p1 .. pn

Add scores s+=p

Node n.pagerank = s

Return (nid, Node n)

}

1. Explain how the dictionary-as-a-string can be used to save space on the construction of the index dictionary.

Sol; in sort based indexing , the terms in the dic are required to be in sorted order of the terms. But to implement this, all the spac required for each entry for sorting should be of same size.

Each term is of diffeemt length so, some space is wasted when extra padding is done for the smallest term. Instead we can use dictionary as a string where the sorting component is the integers in primary array pointing to secondary array where the actal terms and pL are present. This doesnt waste any space and calculations have shown that it saved almost 6 byte data per entry.

1. Give the formula for BM25. How can binary heaps be used to speed up document-at-a-time query processing?

Sol: document at a time query processing

- each doc scored using all query terms then all doc sorted based on score

- time = m.n +m log m, as we loop for all n query terms for all m documents so m.n and we use sorting of the documents based on rank so m log m

- we don’t need all m results as we need just top k m so use heap

- m is bound by Nq/n and Nq where Nq= Nt1 + Nt2 + ..Ntn , so time is Nq.n +Nq log Nq

- binary heap - has reheap which preserves heap property as min or max heap , log n steps

- we store the terms in the heap with increasing order of nextdoc value, we store results in a heap with only k results

1. What are some advantages of computing the disjunction of the query terms as opposed to the conjunction? What some advantages of computing the conjunction over the disjunction?

Sol: we get all documents that have atleats query term. Some queries might have some terms like world war 2 ex, we might have important documentsb with only war 2 as terms in them if we use disjuntive we will get the result but if we use conjuctive we might lose this doc

Conjunction is most valuable when we are doigvery strict search and we want to narrow our search space or documents to score, boolean query will return a few ocumenst that contain all thq uesry terms. So scoring of few documents becomes easier.

Disjunction (OR):

* 1. **Advantages**:
     1. **Increased Recall**: When using disjunction, the system retrieves documents containing **any** of the query terms. This leads to higher recall, ensuring that relevant documents are not missed.
     2. **Flexible Search**: Users can cast a broader net by combining multiple related terms with OR. For example, searching for “cats OR dogs” retrieves content related to either cats or dogs.
     3. **Tolerant to Synonyms**: Disjunction handles synonyms well. If a user searches for “automobile OR car,” both terms are considered.

Conjunction (AND):

* 1. **Advantages**:
     1. **Precision**: Conjunction ensures that retrieved documents contain **all** query terms. This leads to higher precision, as irrelevant documents are filtered out.
     2. **Narrowed Focus**: Users can narrow down search results by combining specific terms with AND. For example, searching for “machine learning AND neural networks” retrieves content specifically related to both topics.
     3. **Structured Queries**: Conjunction allows users to express complex queries by combining multiple criteria. It’s useful for advanced search scenarios.

1. Express the Shakespeare corpus query: 'the speaker who said "Romeo, Romeo, wherefore art thou Romeo?"', using our region algebra.

Sol: "Romeo, Romeo, wherefore art thou Romeo?"”<| (<LINE>...</LINE>) <| ((<SPEECH>...</SPEECH>) |> (<SPEAKER>...</SPEAKER>))

1. Suppose in a document we see 1 occurrence of an a, 2 occurrences of a b, ... 26 occurrences of a z. Draw the Huffman tree one would get following the algorithm from class.

Sol: start with

(a:1/ld) (b:2/ld) -> (a, b : 3/ld)

Then combine with next smallest and so on till all symbols are done and final sum is 1

Huffmans code will have 0 and 1 for each merge of node, read the code from root to the leaf node

1. For each of the following give the distribution for which it is an optimal code: (a) unary code, gamma code, delta code.

Sol: unary : geometric progression : Pr(k) = 1/(2^k)

Gamma : 1/(2k^2)

Delta : 1/(2k(log k)^2)

1. Explain why it is reasonable to guess that Δ-values for posting lists follow a geometric distribution.

Sol: normal geometric progression states that : Pr(k) = p.(1-p)^(k-1)

Its not unreasbale to state that the prob of a term to appear in a document is given by Nt/N we can replace this with p

1. What are some advantages and disadvantages of the IMMEDIATE MERGE versus NO MERGE index update strategies.

Sol: immediate merge : quik response when number fo documents is less , but when numbe rof documents are more, the number of disk operations ton read the index from disk to memory is much more than no merge and so gets slow

No merge: number of times call to disk to check for partitions and term’ sPL is much more so if affected in speed.

IMMEDIATE MERGE:

Advantages:

* **Better Query Performance:** IMMEDIATE MERGE ensures that the index is updated immediately after each modification operation (insert, update, delete). This can result in better query performance as the index is always up-to-date.
* **Consistent Query Response:** Since the index is updated immediately, queries are more likely to reflect the latest state of the data, providing a more consistent experience for users.
* **Real-time Applications:** For real-time or near-real-time applications where the most recent data is critical, IMMEDIATE MERGE is beneficial.

Disadvantages:

* **Overhead:** IMMEDIATE MERGE can introduce additional overhead as the index is updated frequently. This can impact the performance of write operations, especially in scenarios with high write activity.
* **Resource Intensive:** Keeping the index up-to-date in real-time may require more system resources, including CPU and memory, leading to potential performance bottlenecks.

NO MERGE:

Advantages:

* **Reduced Overhead:** NO MERGE delays index updates until a more convenient time, reducing the immediate overhead associated with each modification operation. This can result in better performance for write-heavy workloads.
* **Batch Processing:** NO MERGE allows the system to batch multiple index updates together, potentially optimizing the process and reducing the overall cost of index maintenance.

Disadvantages:

**Query Performance:** Since the index is not immediately updated, there may be a delay before queries reflect the latest changes. This delay can impact query performance, especially for read-heavy workloads.

**Data Inconsistency:** The delay in updating indexes might result in a temporary inconsistency between the actual data and the indexed data, which can be a concern in applications where real-time consistency is critical.

**Increased Latency:** For applications that require near-real-time data access, the delay in index updates introduced by NO MERGE can lead to increased latency in query responses.

1. Describe how the Hybrid Index Maintenance system works.

Sol: put everything in merge first, if exceed the size then switch to inpkace . ,most of the PL is small so immediate merge is done for them , if anyone exceeds the threshold for a PL, then in place is used. Getting best of both world. Not good for smaller mem. Need contiguous space.

1. Give a map reduce algorithm for coming up with the most common word in a collection of documents.

Sol: //we get freq count using map reduce job which is don before

//we ca have a global max to store the most common term with max freq