Lalitha Priya Bijja
101168225

CSC 785 - INFORMATION STORAGE AND RETRIEVAL

1.

A phrase query is a type of search query in which the search engine looks for the exact occurrence of a sequence of words in the specified order. Unlike a regular search query, which may return results containing the individual words in any order, a phrase query requires that the words appear consecutively and in the specified sequence. Phrase queries are important in certain situations where the precise arrangement of words is crucial for the accuracy of the search results.

Here are some situations where phrase queries are essential:

Exact Match Requirements:

Example: If you are searching for a specific book title or a famous quote, using a phrase query ensures that the results match the exact sequence of words.
Eg: searching for "To be or not to be" in quotes will return results related to Shakespeare's famous soliloquy.

Names and Titles:

Example: When searching for a person's full name or the title of a specific document, using a phrase query helps filter out irrelevant results. Eg: searching for "Albert Einstein" as a phrase will prioritize results where the full name appears together.

Programming Code:

Example: In programming, searching for a specific code snippet or error message often requires using a phrase query to find the exact sequence of characters. Eg: searching for "SyntaxError: invalid syntax" in quotes will yield results related to that specific error message.

Product Names:

Example: Searching for a specific product or brand name may require a phrase query to distinguish it from results that mention the individual words separately. For instance, searching for "iPhone 13" as a phrase will yield more relevant results related to that specific model.

In these situations, using phrase queries helps narrow down search results and improves the precision of the search by focusing on the specific arrangement of words. It is especially useful when you need to find information with a high degree of accuracy and avoid false positives that might occur with a more general search query.

2.

angels: 2: ⟨36,174,252,651⟩; 4: ⟨12,22,102,432⟩; 7: ⟨17⟩;

fools: 2: ⟨1,17,74,222⟩; 4: ⟨8,78,108,458⟩; 7: ⟨3,13,23,193⟩;

fear: 2: ⟨87,704,722,901⟩; 4: ⟨13,43,113,433⟩; 7: ⟨18,328,528⟩;

in: 2: ⟨3,37,76,444,851⟩; 4: ⟨10,20,110,470,500⟩; 7: ⟨5,15,25,195⟩;

rush: 2: ⟨2,66,194,321,702⟩; 4: ⟨9,69,149,429,569⟩; 7: ⟨4,14,404⟩;

to: 2: ⟨47,86,234,999⟩; 4: ⟨14,24,774,944⟩; 7: ⟨199,319,599,709⟩;

tread: 2: ⟨57,94,333⟩; 4: ⟨15,35,155⟩; 7: ⟨20,320⟩;

where: 2: ⟨67,124,393,1001⟩; 4: ⟨11,41,101,421,431⟩; 7: ⟨16,36,736⟩

Solution :

'fools rush in'

2: (1,2,3) 4: (8,9,10) 7: (3,4,5)
(13, 14, 15)

3.

(a) precision = 4/10 = 40%, recall = 4/4 = 100% (system 1)
precision = 4/10 = 40%, recall = 4/4 = 100% (system 2)
Both the systems have same precision and recall

(b)

Compute precision and recall at top-5 (ranked results).
— precision = 2/5 ,recall = 2/4 (system 1)
— precision = 2/5, recall = 2/4 (system 2

(c)

System 1

F-score = 2 × (4/10× 4/4) / (4/10 + 4/4)
= 2 * 0.4 / 1.4
= 0.571

System 2

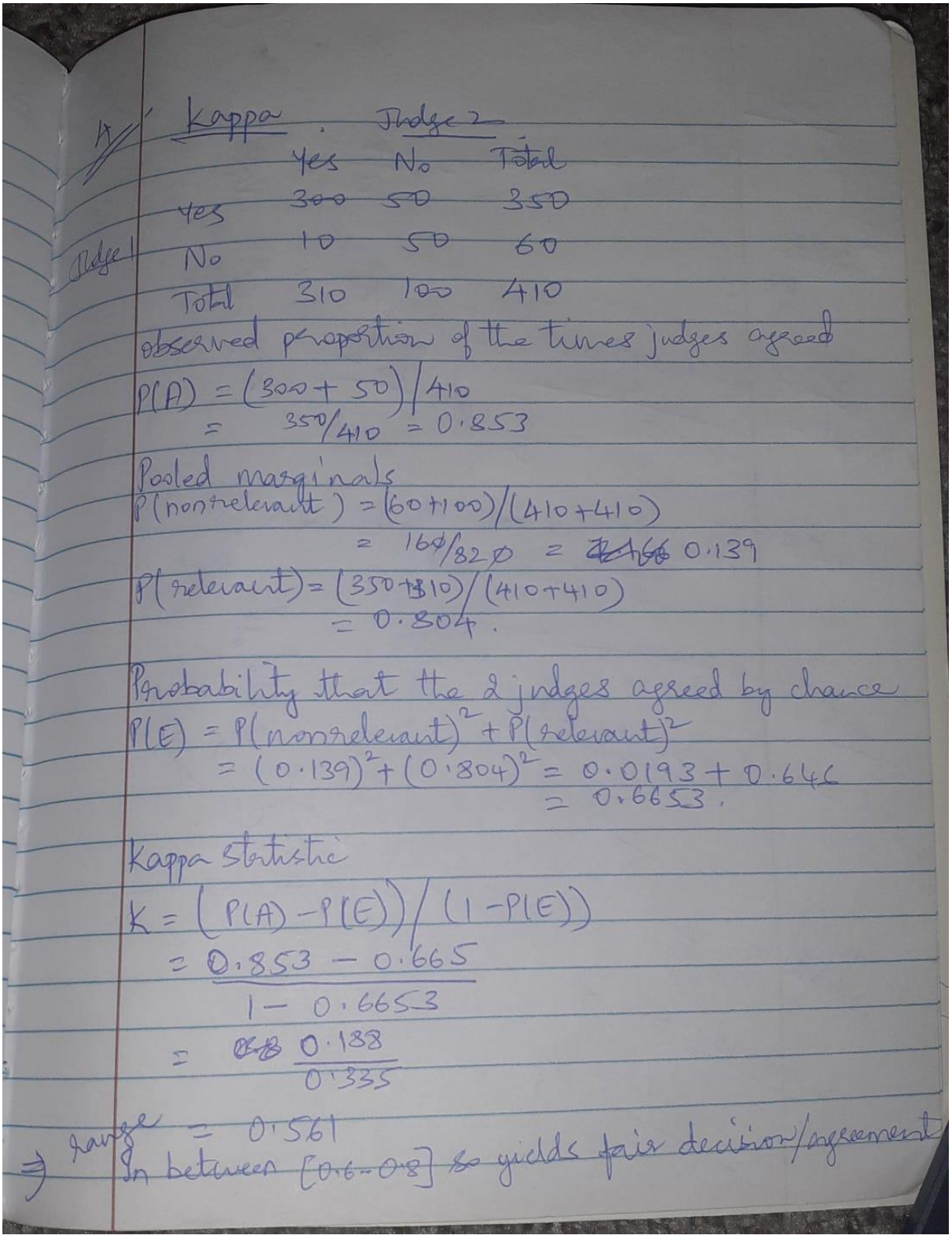
F-score = = 2 × (4/10× 4/4) / (4/10 + 4/4)
= 2 * 0.4 / 1.4
= 0.571

(d)

MAP =

Mean average precision (MAP).
Average precision is the average of the precision value obtained for the set of top k documents exist- ing after each relevant document is retrieved, and this value is then averaged over information needs

4.



5.

Map Reduce Architecture

MapReduce is a programming model and associated implementation designed for processing and generating large-scale data sets on distributed clusters of computers. The MapReduce architecture consists of two main phases: the Map phase and the Reduce phase.

1. Map Phase:

Input Data Division: The input data, often a large dataset, is divided into smaller chunks.

Map Function: A user-defined map function is applied to each chunk independently.

In the context of distributed indexing, this phase typically involves tokenizing and processing individual documents. The goal is to extract relevant information, such as terms and their positions within documents.

Intermediate Key-Value Pairs: The output of the map function is a set of intermediate key-value pairs. The key is often a term, and the value is the associated information (e.g., document ID and position).

2. Shuffle and Sort Phase:

Shuffling: The intermediate key-value pairs are shuffled across the network based on their keys. This ensures that all occurrences of a particular term are grouped together.

Sorting: The shuffled data is sorted by key.

3. Reduce Phase:

Reduce Function: A user-defined reduce function is applied to the sorted and shuffled data.

In the context of distributed indexing, this phase involves aggregating information related to each term, creating an inverted index.

The inverted index typically includes a term and a list of document IDs where that term appears, possibly with additional information like term frequency or position.

Key Characteristics of MapReduce in Distributed Indexing:

Scalability: MapReduce is designed to scale horizontally, allowing the processing of vast amounts of data by adding more machines to the cluster.

Fault Tolerance: MapReduce handles machine failures gracefully. If a node fails during processing, the framework redistributes the tasks to other nodes.

Parallelism: The map and reduce tasks can be performed in parallel on different nodes, improving efficiency.

Simplicity: MapReduce abstracts away many of the complexities of distributed computing, allowing developers to focus on defining the map and reduce functions.

Flexibility: The programming model is flexible and can be applied to various tasks beyond indexing, making it versatile in distributed computing environments.

6.

6.

Given,

$T = 10^5$ (100,000)

$b = 0.49$

$k = 44$.

tokens = 1,000,020

$\text{Vocab} = (44 \times 1{,}000{,}020)^{0.49}$.

$\cong$ 38,323.

7. (a) Collection size = 400,000

bytes = 20/term

doc frequency = 4 bytes.

pointing lists = 4 bytes.

$= M \times (20 + 4 + 4)$

$= 400{,}000 \times 28 = 11.2\text{MB}$.

(c) Yes.

$M \times (4 + 4 + 3 + 8)$

$= 400{,}000 \times (4 + 4 + 3 + 8)$

$= 7.6\text{MB}$.

4 bytes = frequency.

4 bytes = postings pointer.

3 bytes = term pointer

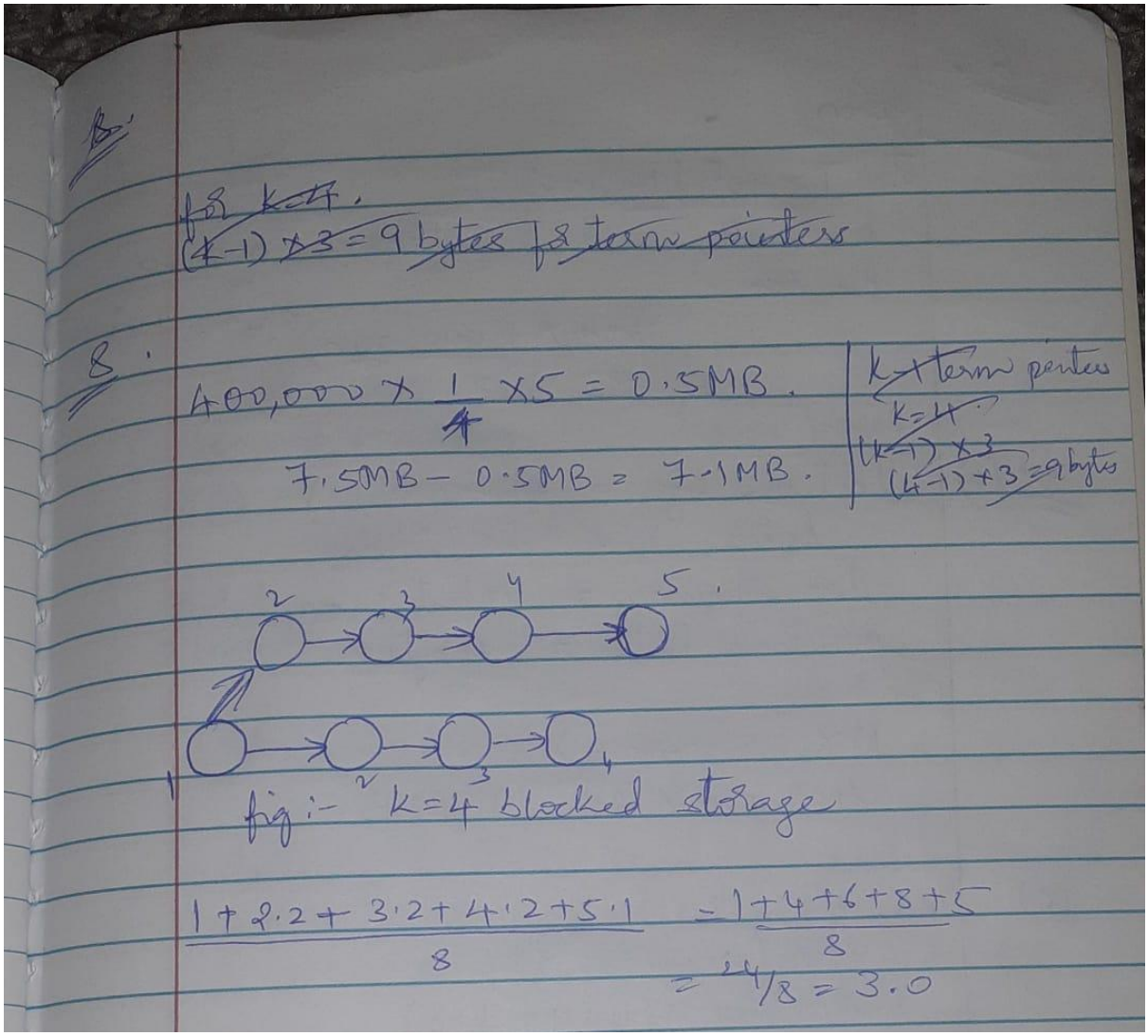
8 bytes = average for the term.

Reduced from 11.2 MB to 7.6 MB.

7(b).

Allocating a fixed-width entry of 20 bytes per term in a storage scheme is inefficient due to the average term length in English being around eight characters. This results in the wastage of approximately twelve characters per term. Moreover, this approach poses limitations on storing terms with lengths exceeding twenty characters, such as "hydrochlorofluorocarbons" and "supercalifragilisticexpialidocious."

8.



***correction**

7.6 MB – 0.5MB = 7.1 MB

9.

$$JC(A,B) = \frac{|A \cap B|}{|A \cup B|} \in [0,1].$$

query(q) = ides of March
doc1 (d1) = caesr died in March.
doc2 (d2) = the long March, not easy in winter

$JC(q/d1) = 1/6$
$JC(q/d2) = \frac{1}{9}$
~~Query~~ Doc1 is similar.

11.

Cosine for length-normalized vectors

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|}$$

$$= \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

→ Not discussed in class.

10.

Document 1

tf-idf(car) = 27 * 1.65 = 44.55

tf-idf(auto) = 3 * 2.08 = 6.24

tf-idf(insurance) = 0 * 1.62 = 0

tf-idf(best) = 14 * 1.5 = 21

Document 2

tf-idf(car) = 4 * 1.65 = 6.6

tf-idf(auto) = 33 * 2.08 = 68.64

tf-idf(insurance) = 33 * 1.62 = 53.46

tf-idf(best) = 0 * 1.5 = 0

Document 3

tf-idf(car) = 24 * 1.65 = 39.6

tf-idf(auto) = 0 * 2.08 = 0

tf-idf(insurance) = 29 * 1.62 = 46.98

tf-idf(best) = 17 * 1.5 = 25.5