--From Subrina Jahan 😋

--7th Batch

All Questions from Slide 1-4

Chapter 1

What is Information Retrieval, define its main goal, and why is it important in the context of data and information management?

Information Retrieval (IR) is the process of obtaining relevant information from a large collection of data or documents based on a user's query. It focuses on the structure, organization, storage, and retrieval of unstructured or semi-structured data, such as text, documents, or multimedia content.

Importance in Data and Information Management:

- IR helps users quickly find relevant information from large datasets.
- It manages unstructured and semi-structured content like web pages, PDFs, and emails.
- Supports better decision-making in fields like business, healthcare, and education.
- Powers search engines, enabling efficient large-scale document retrieval.
- Handles multimedia data (images, videos, audio) beyond just text.
- Provides personalized and context-aware search results to enhance user experience.

What are two differences between documents and database records?

Here are the key differences between **documents** and **database records**:

Document (IR)	**Database Record**
Contains unstructured or semi-structured text, such as articles or web pages	Contains structured data with fixed fields, like rows in a table
Written in natural language with varying formats and lengths	Stored in a defined schema with known field types (e.g., name, balance, date)
Searched using keyword-based or relevance-based methods	Searched using exact field matches or range queries in SQL
Example: A news article or a blog post	Example: A bank record with account number, name, and balance
May include metadata like title, author, or timestamp	Each entry follows a strict format with no free-form text
Used in Information Retrieval systems	Used in Database Management Systems (DBMS)

Why is comparing text in Information Retrieval inherently difficult?

- The same meaning can be expressed in many different ways using synonyms, paraphrasing, or sentence reordering. This makes exact word matching ineffective.
- Words can have multiple meanings depending on context. For example, “bank” can mean a financial institution or the side of a river.
- Text data is unstructured, unlike database records, making it harder to analyze and match precisely.
- Variations in tense, plural/singular, and derivations (e.g., “run”, “runs”, “running”) make matching difficult without stemming or lemmatization.
- Common words like “the”, “in”, “of” may not add value but appear frequently. IR systems must decide when to ignore or include them.
- User queries may contain spelling mistakes or informal language that differs from how it appears in documents.

Compare site search to web search, vertical search, and enterprise search.

1. **Site Search:**
   - Focuses on searching within a single website or domain.
   - Used to help users find information quickly on that specific site.
   - Example: Search box on an e-commerce or news website.
2. **Web Search:**
   - Searches the entire public World Wide Web.
   - Covers billions of web pages across diverse topics.
   - Examples include Google, Bing, and Yahoo.
3. **Vertical Search:**
   - Specialized search limited to a specific domain or industry.
   - Provides more focused, relevant results in areas like travel, health, or real estate.
   - Examples: Kayak (travel), WebMD (health), Zillow (real estate).
4. **Enterprise Search:**
   - Searches internal documents and data within an organization.
   - Helps employees find company-specific information securely.
   - Examples include intranet search systems or document management tools.

Describe the tasks in Information Retrieval (IR)

Information Retrieval (IR) systems are designed to perform various tasks that help users find, filter, and organize information. The major tasks in IR include:

1. Ad-hoc Search:

- This is the most common IR task.
- The system retrieves a ranked list of relevant documents based on a one-time, user-submitted query.
- Example: Searching “climate change” in Google.

2. Filtering:

- The system automatically delivers relevant information to users based on their interests or profiles.
- It’s a continuous process where new documents are matched against a standing user profile.
- Example: News apps recommending articles based on your reading habits.

3. Classification:

- Documents are assigned to one or more predefined categories or labels.
- This helps in organizing information and automating tasks like spam detection.
- Example: Classifying emails as “spam” or “primary inbox”.

4. Question Answering (QA):

- Instead of returning documents, the system tries to return a specific and direct answer to a user's question.
- Example: Asking “National Bird of Bangladesh?” and getting “Doel”.

Define 'relevance' in Information Retrieval, and discuss the challenges involved in making accurate relevance judgments.

In Information Retrieval (IR), relevance refers to how well a retrieved document satisfies a user’s information need. A document is considered relevant if it contains the information that the user was looking for when they submitted a query.

Types of Relevance:

1. Topical Relevance:

- This means the document is related to the same topic as the user’s query.
- For example, if the query is " climate change ", a report discussing UV ray would be topically relevant.

2. User Relevance (Situational Relevance):

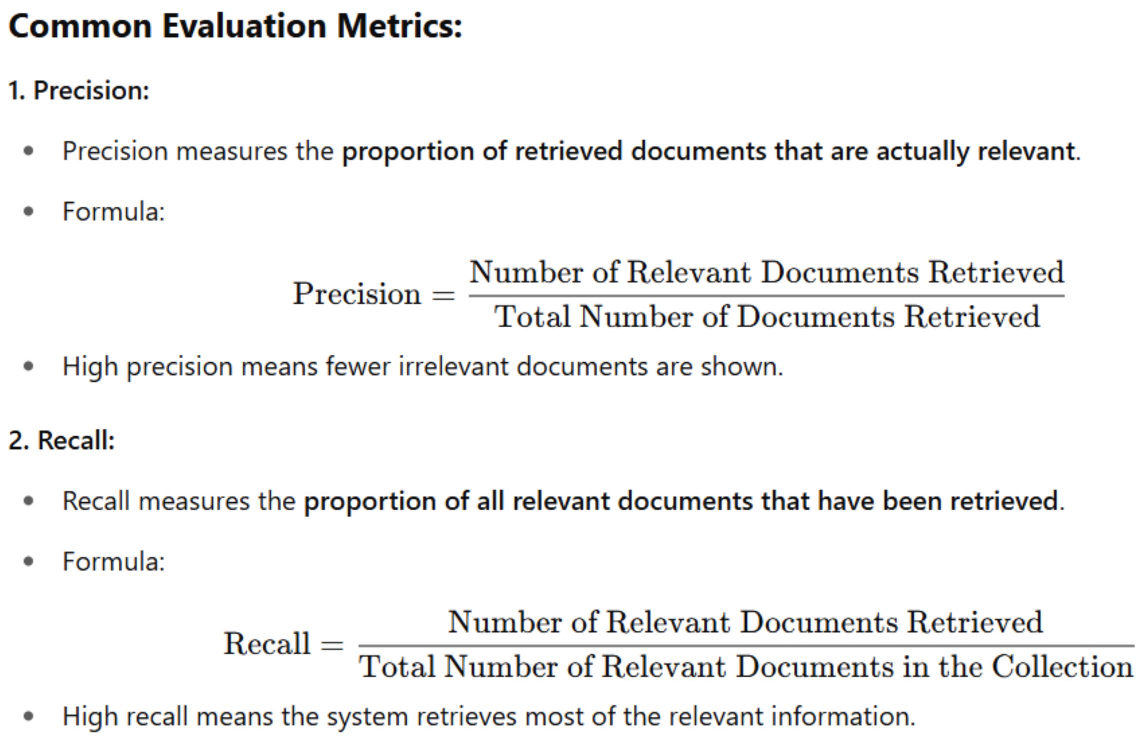
- This goes beyond just the topic and includes the user's context, intent, task, background knowledge, and preferences.
- A technically detailed article might be relevant to a researcher but not to a school student, even if both are asking the same question.

Challenges in Making Accurate Relevance Judgments:

- Short keyword queries often lack clarity about the user's actual need.
- Relevance is personal and context-dependent, varying from user to user and even over time.
- The system often doesn't know enough about the user's background or current task.
- A user's definition of what's relevant may evolve during a search session.
- Relevance can include freshness, credibility, readability, novelty, and even presentation style — not just topic matching.

Explain how Information Retrieval systems are evaluated, detailing metrics such as recall and precision

Evaluating Information Retrieval (IR) systems is essential to measure how well the system retrieves relevant documents in response to user queries. The goal is to assess both effectiveness (quality of results) and efficiency (speed and performance). Evaluation is typically based on test collections that include a set of documents, queries, and human-judged relevance labels.



Common Evaluation Metrics:

1. Precision:

- Precision measures the **proportion of retrieved documents that are actually relevant**.
- Formula:

$$\text{Precision} = \frac{\text{Number of Relevant Documents Retrieved}}{\text{Total Number of Documents Retrieved}}$$

- High precision means fewer irrelevant documents are shown.

2. Recall:

- Recall measures the **proportion of all relevant documents that have been retrieved**.
- Formula:

$$\text{Recall} = \frac{\text{Number of Relevant Documents Retrieved}}{\text{Total Number of Relevant Documents in the Collection}}$$

- High recall means the system retrieves most of the relevant information.

Role of User Needs in Information Retrieval (IR) Systems

Information Retrieval (IR) systems are designed to help users find information that satisfies their **needs**. Understanding and addressing **user needs** is at the core of building effective IR systems. These needs are often expressed through **queries**, but the true information need may be deeper, vague, or context-dependent.

Explain the Query refinement techniques

Query refinement techniques are used in IR systems to improve the initial user query in order to retrieve more accurate and relevant results. These techniques help interpret vague, short, or poorly structured queries by enhancing or correcting them.

1. Query Expansion

- Adds related or synonymous terms to the original query.
- Helps retrieve documents that use different vocabulary for the same concept.
- Example: Expanding "heart attack" to include "myocardial infarction".

2. Relevance Feedback

- Uses user responses (like clicked results or marked relevant documents) to improve future searches.
- The system learns what the user considers relevant and adjusts the query accordingly.

3. Query Suggestion (Auto-Completion)

- Offers alternative or more complete query suggestions as the user types, based on popular or related queries from past user data.
- Example: Typing “solar” suggests “solar panel cost”, “solar energy in Bangladesh”, etc.

4. Spell Correction

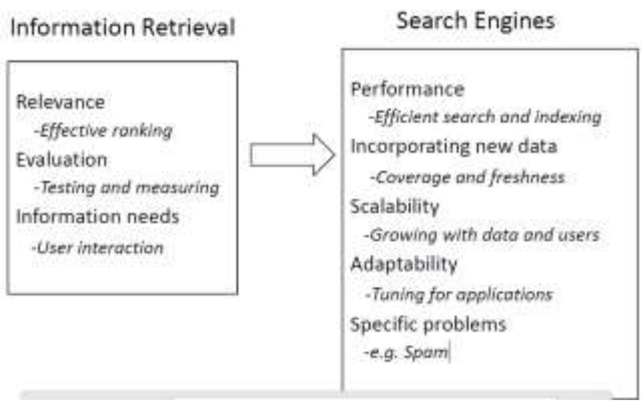
- Automatically detects and corrects spelling errors in the query.
- Example: Correcting “enviroment” to “environment”.

Define what a search engine is in the context of IR, explain how it relates to Information Retrieval

A **search engine** is a software system that applies **Information Retrieval (IR)** techniques to find and rank relevant documents from a large collection, such as the web, in response to a user's query.

Relation to IR:

- **Search engines are the practical implementation of IR concepts.** They use IR methods like indexing, query processing, ranking, and evaluation to retrieve useful information efficiently.
- **IR focuses on algorithms and models**, such as term weighting (e.g., TF-IDF), relevance ranking, and user intent understanding.
- **Search engines apply these models at scale**, handling billions of documents and users.



How do Search Engines Utilize Algorithms to Prioritize and Rank Search Results Based on Relevance and User Intent?

Search engines use **ranking algorithms** to decide the order in which documents appear in response to a user's query. These algorithms analyze many factors to measure **relevance** and estimate **user intent**, aiming to show the most useful results first.

1. Relevance-Based Ranking

- **TF-IDF (Term Frequency–Inverse Document Frequency):** Scores documents based on how often query terms appear and how unique those terms are across the collection.
- **Vector Space Model:** Treats documents and queries as vectors and ranks based on their similarity (e.g., cosine similarity).
- **BM25 & Other Retrieval Models:** Advanced models that use statistical text features to rank documents based on term relevance and document length.

2. User Intent Understanding

- **Query Analysis:** The system interprets the query to detect **intent** (e.g., informational, navigational, or transactional).
- **Personalization:** Uses user history, location, or device type to adjust rankings.
- **Behavioral Data:** Algorithms learn from previous user behavior—e.g., which results users clicked on most for similar queries.

3. Other Ranking Factors

- **PageRank / Link Analysis:** Considers the number and quality of links pointing to a page as a sign of authority.
- **Freshness:** Recently updated or published content may rank higher, especially for time-sensitive queries.
- **Spam Detection:** Filters out low-quality or manipulative.

Identify and explain the major issues and challenges faced by Information Retrieval systems and search engines

- **Performance** – Ensuring efficient search and fast indexing to handle high query loads and large datasets.
- **Incorporating New Data** – Maintaining good **coverage** (how much of the web is indexed) and **freshness** (how recently it was updated).
- **Scalability** – Adapting to billions of documents and millions of users while maintaining speed and reliability.
- **Adaptability** – Tuning components like ranking algorithms, query processing, and interfaces for different applications.
- **Specific Problems** – Tackling issues like **spam**, duplicate content, and adversarial attacks that harm search result quality.

Define 'spam' in web search, explain why it is a major issue, list different types of spam, and describe the concept of "adversarial IR.

Definition of Spam: In web search, spam refers to any content that is intentionally created to manipulate search engine rankings unfairly, often providing little or no value to users. It aims to trick the ranking algorithm to gain higher visibility.

Why Spam Is a Major Issue:

- Reduces Search Quality: Spam clutters results with low-quality or irrelevant content.
- Wastes Resources: It consumes indexing and crawling capacity.
- User Mistrust: Poor results affect user experience and trust in the search engine.

Types of Spam:

- Spamdexing (Term Spam): Stuffing pages with repeated or misleading keywords.
- Link Spam: Creating fake or irrelevant backlinks to boost PageRank.
- Cloaking: Showing different content to search engines than to users.
- Hidden Text/Links: Inserting keywords in invisible text or tiny font to fool crawlers.
- Duplicate Content: Copying large portions of content across multiple pages or sites.

Adversarial IR (Adversarial Information Retrieval):

Adversarial IR is a subfield of Information Retrieval that studies how to defend search engines against spam and manipulation. It treats spammers as adversaries who try to exploit system weaknesses. Techniques in adversarial IR involve:

- Spam detection and filtering algorithms.
- Learning models that adapt to new spam tactics.
- Identifying abnormal linking or term usage behavior.

In the context of information retrieval, how do search engines address the challenge of balancing precision and recall, and what techniques are employed to optimize the search experience for users?

In Information Retrieval (IR), **precision** and **recall** often conflict — improving one may reduce the other. Search engines must balance both to ensure users get **relevant** results (**precision**) while not missing other useful ones (**recall**).

1. The Challenge:

- **High Precision:** Shows only the most relevant documents, but may **miss some useful ones**.
- **High Recall:** Retrieves many relevant documents, but **also brings in irrelevant ones**.

Search engines aim for a **trade-off** — enough relevant documents without overwhelming the user with noise.

2. Techniques to Balance and Optimize:

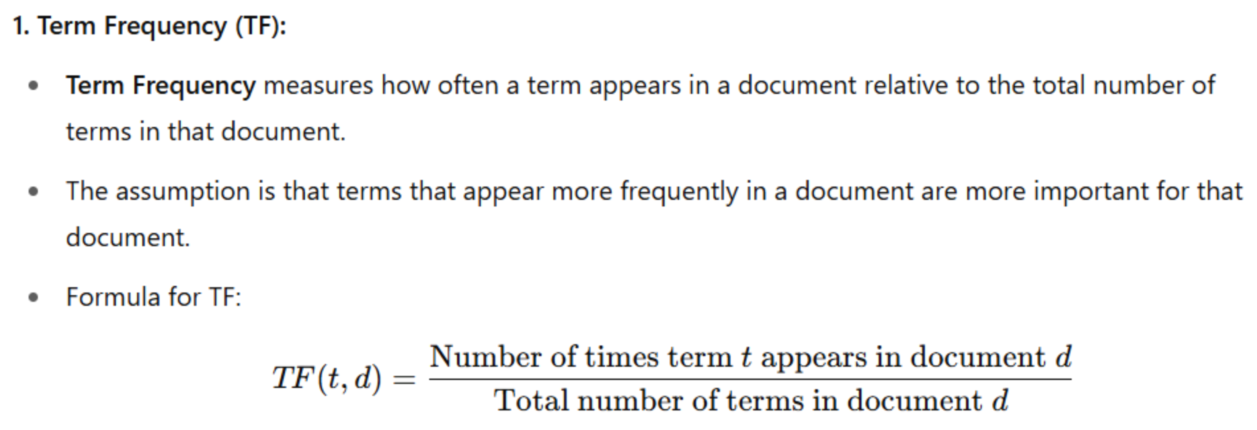
1. **Ranking Algorithms:**
   - Use scoring models (e.g., TF-IDF, BM25) to sort results by relevance, improving **precision in top results**.
2. **Query Refinement:**
   - Techniques like **query expansion**, **spell correction**, and **relevance feedback** help retrieve more relevant documents, **boosting both precision and recall**.
3. **Personalization:**
   - Adjusts rankings based on user behavior or location to improve precision **for that specific user**.
4. **Snippets and Highlighting:**
   - Helps users quickly judge result relevance, improving **perceived precision**.
5. **Machine Learning Models:**
   - Learn from user interaction (clicks, dwell time) to optimize future rankings and balance relevance better.

Explain difference between data retrieval and information retrieval.

Data Retrieval	**Information Retrieval**
Extracts specific data from structured databases using exact queries.	Retrieves relevant documents or information from unstructured data sources.
Works with structured data stored in tables with well-defined fields.	Deals with unstructured or semi-structured data like text and multimedia.
Uses exact queries matching predefined fields (e.g., SQL queries).	Handles flexible queries, often based on keywords or user intent.

Aims to retrieve precise data for reporting, calculations, or updates.	Focuses on finding the most relevant content for broader information needs.
Provides results that exactly match the query criteria.	Returns ranked results based on relevance, not exact matches.
Uses tools like SQL and relational databases for efficient querying.	Employs text indexing, natural language processing, and machine learning.

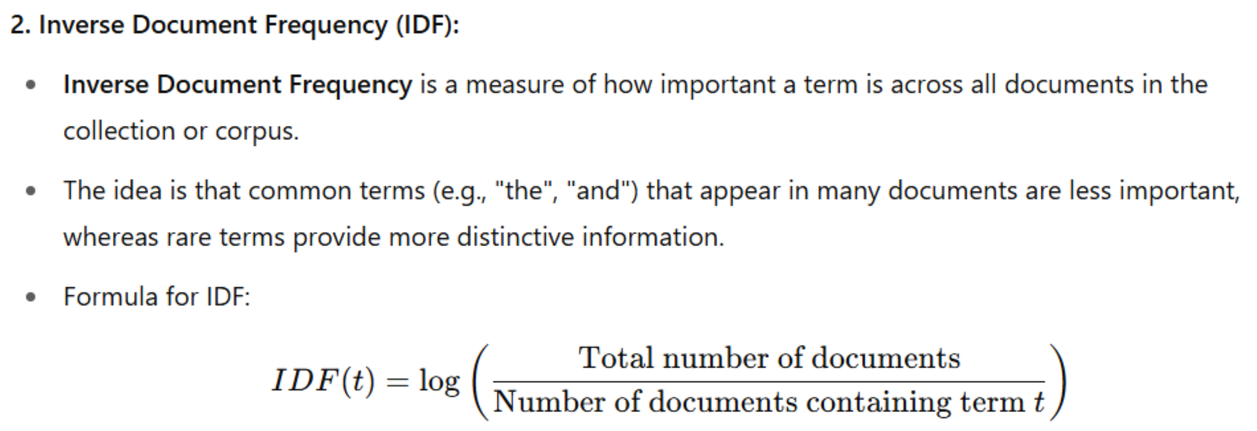
What is a term frequency-inverse document frequency (TF-IDF) score, and how is it used in IR?



1. Term Frequency (TF):

- **Term Frequency** measures how often a term appears in a document relative to the total number of terms in that document.
- The assumption is that terms that appear more frequently in a document are more important for that document.
- Formula for TF:

$$TF(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$



2. Inverse Document Frequency (IDF):

- **Inverse Document Frequency** is a measure of how important a term is across all documents in the collection or corpus.
- The idea is that common terms (e.g., "the", "and") that appear in many documents are less important, whereas rare terms provide more distinctive information.
- Formula for IDF:

$$IDF(t) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing term } t}\right)$$

TF-IDF Score:

- **TF-IDF** is the product of TF and IDF. It combines both the frequency of a term within a document and its rarity across the entire corpus.
- The TF-IDF score helps identify the most important terms in a document in relation to the entire document collection.
- Formula for TF-IDF:

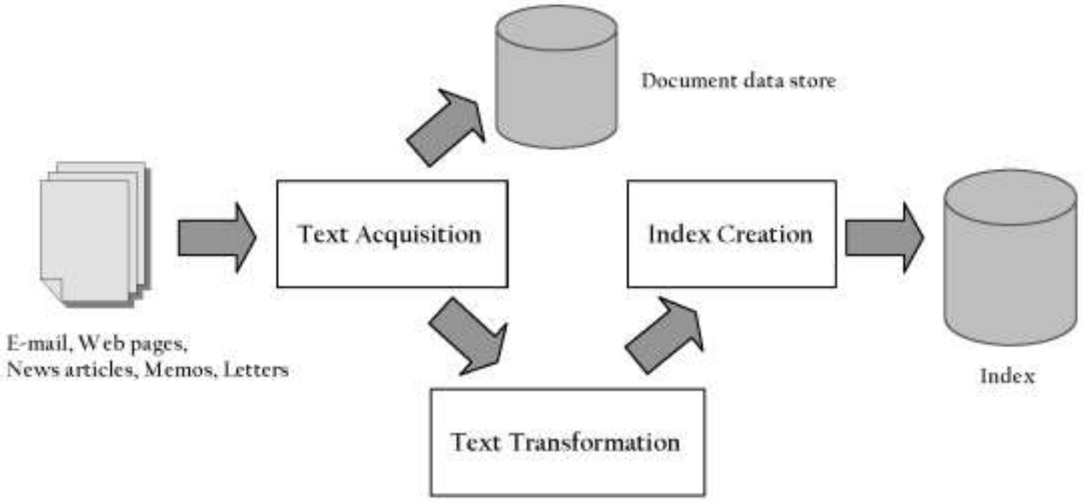
$$\text{TF-IDF}(t, d) = TF(t, d) \times IDF(t)$$

This score is used to rank the importance of terms for document retrieval, ensuring that terms with both high frequency in a specific document and low frequency across other documents are given higher importance.

Chapter 2

Explain indexing of search engine with figure.

Indexing is an essential process in search engines that organizes and structures data to enable fast and efficient information retrieval. It involves creating a data structure, known as an **inverted index**, which maps terms to the documents in which they appear.



1. Text Acquisition

The process begins with collecting documents from various sources such as **websites, blogs, PDFs, or news feeds**.

- Tools like **web crawlers** identify and fetch documents.
- All content is converted into a **uniform format** (e.g., HTML or XML) while preserving essential **metadata** like title, date, or author.
- Example: A search engine downloads a webpage about "machine learning".

2. Text Transformation

After acquisition, the content is preprocessed for indexing.

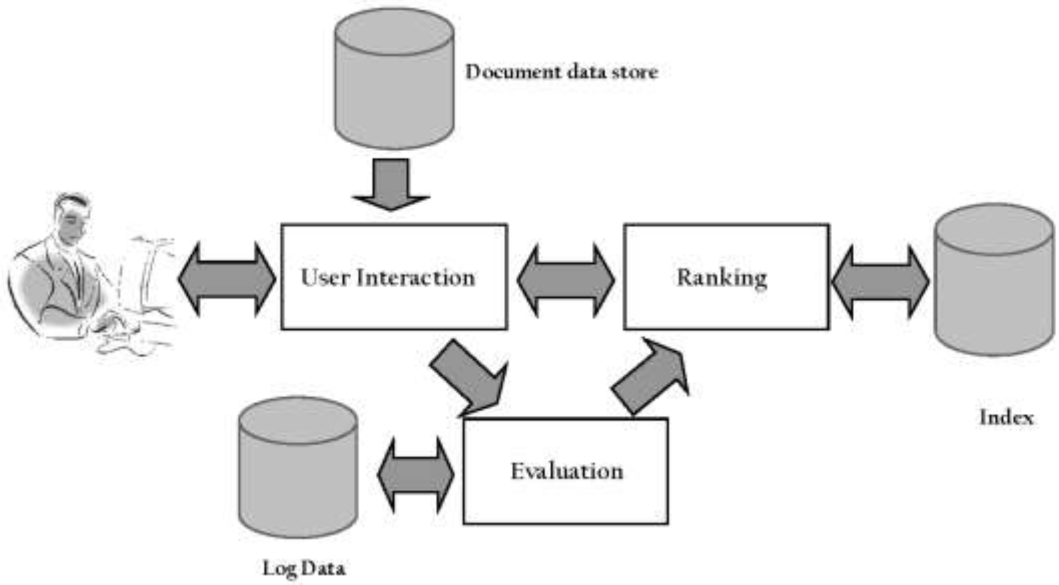
- **Parsing** identifies structural elements (titles, headings, links).
- **Tokenization** breaks text into words or terms.
- **Stop word removal** eliminates common words (e.g., "is", "the").
- **Stemming** reduces words to root forms (e.g., "computers" → "computer").
- **Link analysis** helps assess document popularity and context (useful for PageRank).
- **Example:**
  Original: *"Machine learning is a subset of artificial intelligence."*
  After transformation: ["machine", "learn", "subset", "artifici", "intellig"]
-

3. Index Creation

In this phase, efficient data structures are built to support fast search.

- **Document statistics** like term frequency and positions are collected.
- **Weighting techniques** such as **TF-IDF** assign importance to terms.
- A **forward index** (doc → terms) is converted into an **inverted index** (term → doc list).
- **Compression** is applied to save storage and improve speed.
- Example:
  - Term: machine → Appears in documents 1, 3, and 5.
  - Term: learning → Appears in documents 1, 2, and 4.

Explain Query process of search engine with figure.



The Query Process in a search engine refers to the series of steps that occur from the moment a user submits a search query to when they receive a list of ranked results. This process involves understanding the user's intent, transforming the query for better interpretation, searching the indexed data, ranking the results, and finally evaluating the system's performance based on user feedback.

OR,

The **Query Process** refers to the series of steps that take place between **user query submission** and **display of ranked search results**. It ensures that user intent is understood and that the most relevant results are delivered efficiently.

1. User Interaction

- The user enters a query through a search box or voice command.
- Queries may be simple (e.g., "coffee shops nearby") or complex (e.g., Boolean queries).
- Before searching, the query is transformed using:
    - **Spell checking**
    - **Query suggestion**
    - **Query expansion**
    - **Relevance feedback**

2. Ranking

- The transformed query is matched against the indexed documents.
- **Ranking algorithms** score documents based on relevance (e.g., using **TF-IDF**, BM25).

- Other factors like **PageRank** and **user behavior** may influence ranking.
- Results are then sorted and displayed in order of relevance.

3. Evaluation

- The system monitors performance in two ways:
  - **Offline Evaluation**: Uses test queries and relevance judgments.
  - **Online Evaluation**: Uses real-time user data (click logs, session time).
- Measures like **precision**, **recall**, and **response time** assess **effectiveness** and **efficiency**.

Explain the process of weighting terms in indexing, specifically defining TF-IDF (Term Frequency-Inverse Document Frequency) and its calculation

In information retrieval, **term weighting** is essential to determine the importance of a word in a document relative to a collection (corpus). One of the most widely used term weighting methods is **TF-IDF (Term Frequency–Inverse Document Frequency)**.

TF measures how often a term appears in a document. The assumption is that a term appearing more frequently in a document is more important.

TF(Term Frequency)
= Number of times the term appears in the document/ Total number of words in the document

IDF measures how unique or rare a term is across all documents in the corpus. Common terms that appear in many documents have a lower IDF.

IDF(Inverse Document Frequency)

=log10(Total number of documents/Number of documents containing the term)

Consider a document containing 100 words, wherein the word apple appears 5 times. Then calculate the term frequency for apple in that document. Again, assume you have 10 million documents and the word apple appears in 1000 of these. Then, calculate the inverse document frequency. And finally, calculate the TF-IDF weight.

TF(Term Frequency)
= Number of times the term appears in the document/ Total number of words in the document

Number of times "apple" appears = 5, Total number of words in the document = 100

TF=5/100 == 0.05

IDF(Inverse Document Frequency)

=log10(Total number of documents/Number of documents containing the term)

Total number of documents = 10,000,000, Number of documents containing "apple" = 1,000

IDF=log10(10,000,000/1,000)

=log10(10,000)=4

TF-IDF Weight:

TF-IDF=0.05×4=0.2

What is inverted index and stemming? Give an example

Inverted Index

An **Inverted Index** is a data structure used in search engines to efficiently retrieve documents containing specific words or terms. It maps each unique term in a collection of documents to a list of documents in which the term appears. This allows for quick full-text searches, as it eliminates the need to scan entire documents sequentially.

Example of Inverted Index: Suppose we have the following documents:

- Document 1: "apple is sweet"
- Document 2: "apple and orange are fruits"
- Document 3: "banana is yellow"

The inverted index for these documents would look like this:

Term	Document List
apple	Document 1, Document 2
is	Document 1, Document 3
sweet	Document 1
and	Document 2
orange	Document 2
are	Document 2
fruits	Document 2
banana	Document 3
yellow	Document 3

Using this structure, if a user searches for "apple," the search engine directly retrieves **Document 1** and **Document 2**.

##Importance of Inverted Index in Search Engine Architecture

- Enables a search engine to quickly find documents for a user query.
- Instead of scanning all documents, it can retrieve just the posting lists.
- Only stores relevant mapping between terms and document IDs.
- Can include term frequency, positions, or weights for scoring documents.
- Supports Operations: Boolean queries (AND, OR, NOT), Phrase and proximity search, Wildcards and partial matches

Stemming

Stemming is the process of reducing a word to its base or root form. It is used to group words with the same base meaning but different grammatical forms. Stemming helps in improving search results by matching related terms, even if the user does not use the exact word.

Example of Stemming

- Words: **running, runs, ran**
- Stemming Output: **run**

If a search query includes "running," stemming ensures that documents containing "run" or "ran" are also retrieved.

What is a web crawler? Mention different types of crawlers used in IR.

A **web crawler** (also known as a **web spider** or **web robot**) is an automated program used by search engines to systematically browse and collect information from websites on the World Wide Web. It starts by fetching a list of URLs and follows hyperlinks within those pages to discover new pages. The content of these pages is then indexed, enabling search engines to provide relevant results to users.

There are several types of crawlers used in **Information Retrieval (IR)**:

Web Crawlers:

- Crawl the public web by following hyperlinks.
- Used by general-purpose search engines (e.g., Googlebot).
- Focus on large-scale coverage and keeping content fresh.

Single-Site Crawlers:

- Designed to crawl a specific website.
- Used in site-specific search engines to index only one domain.

- Useful for organization-based search.

Topical or Focused Crawlers:

- Crawl selectively based on a specific topic or category.
- Used in vertical search (e.g., only for news, sports, health).
- Use content classifiers to follow topic-relevant links only.

Enterprise Crawlers:

- Crawl internal enterprise-level systems, databases, and document repositories.
- Provide search functionality for organizational data (e.g., HR systems, shared drives).

Desktop Crawlers:

- Crawl data stored on a single user's personal computer.
- Used in desktop search applications like Windows Search.

What is the function of an RSS reader in information retrieval systems?

An **RSS (Really Simple Syndication) reader** plays a key role in **text acquisition** for Information Retrieval (IR) systems by automatically collecting **real-time document streams** such as news articles, blog posts, podcasts, and videos.

Functions of an RSS Reader in IR:

- The RSS reader subscribes to **RSS feeds** and receives updated content **as soon as it's published**.
- Provides a continuous flow of new **XML documents** to the IR system without needing manual crawling.
- Ensures that collected documents follow a **standardized XML format**, making them easier to parse and index.
- Helps maintain the **freshness** of the index by instantly adding newly published content.

Explain the importance of link analysis in web search

Link analysis is a fundamental technique in web search that uses hyperlinks and anchor text between web pages to assess the importance, popularity, and relationships among web documents. It plays a major role in improving the relevance and ranking of search results in Information Retrieval (IR) systems.

Importance of Link Analysis:

    - Pages with many **incoming links** (especially from trusted sites) are considered more **authoritative** and **relevant**.
    - Algorithms like **PageRank** use this idea to rank pages higher in search results.
    - Helps distinguish between high-quality and low-quality pages even if they contain similar keywords.
    - Link-based metrics (like PageRank) don't depend on the query, so they can help rank pages **before the search happens**.
    - The clickable text of links (anchor text) often provides **additional context** and improves the indexing of the target page

Why are compression and index distribution used in search engine indexes, and how do they help search engines?

In modern search engines, managing massive amounts of data efficiently is crucial. To handle this, **compression** and **index distribution** are essential techniques used in the indexing process.

1. Compression:

- Compression reduces the size of the inverted index by encoding document IDs and term positions more efficiently.
- It helps by:
    - **Saving storage space**, which is vital when handling billions of documents.
    - **Improving search speed** by reducing disk read times.
    - **Allowing more data in memory**, leading to faster query processing.
- Common techniques include **variable-byte encoding** and **delta encoding**.

Example:

Instead of storing doc IDs [100, 103, 107], the gaps [100, 3, 4] are stored for better compression.

2. Index Distribution:

- Index distribution spreads the search index across **multiple servers or nodes**.
- It supports:
    - Handles huge volumes of data and user queries.
    - Queries are processed faster across servers.
    - If one node fails, others can still operate.
- Distribution methods include:
    - **Document-based**: Different servers store different document sets.
    - **Term-based**: Servers are assigned specific terms.
    - **Replication**: Indexes are duplicated for reliability and load balancing.

Example:

A search request is split between several servers, each processing part of the index, and the results are merged for seamless output

What is a parser and a tokenizer in text transformation? Explain with example.

In text transformation, a **tokenizer** and a **parser** are key components used to process and prepare raw text for indexing in an Information Retrieval (IR) system.

- A **tokenizer** is responsible for splitting the input text into smaller units called tokens, which are usually words or terms.

For example, the sentence "The quick-brown fox's jump" would be tokenized into ["The", "quick", "brown", "fox", "jump"], ignoring punctuation and special characters.

- A **parser** analyzes the structure of the text, especially when markup languages like HTML or XML are involved. It identifies structural elements such as titles, headings, and paragraphs using tags.

For instance, from <h1>Overview</h1>, a parser would extract "Overview" as a heading.

Explain link analysis and stopping.

1. Link Analysis:

- **Definition:**
  Link analysis is a technique primarily used in web search to evaluate hyperlinks between web pages to determine their importance, popularity, and relevance.
- It helps search engines understand which pages are more authoritative based on how many other pages link to them.
- **Example – PageRank:**
- Google's PageRank algorithm assigns a score to each page depending on the **quality and quantity of incoming links**.
- **Anchor Text Usage:**
- Link analysis also considers the **anchor text** (text of hyperlinks), which can provide extra context about the target page and improve indexing and ranking accuracy.

2. Stopping:

- **Definition:**
  Stopping refers to the removal of very common words (called **stop words**) from text during indexing.
- **Common Stop Words:**
- Examples include "the", "is", "in", "and", etc., which usually do not carry meaningful information for retrieval.

- **Benefits:**
  - Reduces the **size of the index**.
  - Speeds up **retrieval and matching**.
  - Often does not impact relevance negatively.
- **Limitation:**
  In some cases (e.g., the phrase **"To be or not to be"**), stop words can be meaningful. So stopping must be applied carefully depending on the application.

Explain how user interaction works.

User interaction in an Information Retrieval (IR) system refers to the communication between the user and the system to perform a search and view results.

1. **Query Input**: The user begins by entering a search query through an interface. This can be a simple search bar or a more advanced form depending on the application. Some systems support structured query languages for complex searches.
2. **Query Transformation**: After receiving the input, the IR system may apply transformation techniques such as spell checking, query suggestions, and query expansion. These help improve the accuracy and relevance of the search by refining the initial query.
3. **Results Output**: The system then displays a ranked list of documents. Each result usually includes a snippet showing where and how the query terms appear in the document.

Explain ranking and evaluation

1. Ranking:

- Ranking determines the order of documents shown for a user query.
- Each document is scored based on relevance using a **ranking algorithm**.
- A common method is **TF-IDF** (Term Frequency-Inverse Document Frequency), where documents with frequent and distinctive terms get higher scores.
- **Example:** For the query *"science books"*, documents that mention these terms more often and uniquely will rank higher.
- The goal is to display the **most relevant documents first**.

2. Evaluation:

- Evaluation measures the **performance** of an IR system.
- **Effectiveness** is checked using metrics like **Precision**, **Recall**, and **F1-Score**.
- **Efficiency** is measured by **response time** and **throughput**.
- Evaluation helps improve the system and ensures **accurate and fast results** for users.

Chapter 3

What is a web crawler? Explain its working process, its major jobs

A **web crawler** (also called a spider or bot) is an automated program used by **search engines** to browse the web systematically and collect data from web pages for **indexing** in an Information Retrieval (IR) system.

Working Process:

1. **Start with Seed URLs:**
    - The crawler begins with a list of known web addresses called seed URLs (e.g., https://www.example.com).
2. **Download Pages:**
    - It sends HTTP requests to download content from those pages.
3. **Parse and Extract Links:**
    - It scans the HTML of downloaded pages to extract all hyperlinks.
4. **Queue New URLs:**
    - Newly found links are added to a **URL queue** to be crawled next.
5. **Follow Politeness Policies:**
    - The crawler respects the **robots.txt** file and adds delays between requests to avoid overloading servers.
6. **Store Data:**
    - Downloaded pages are stored and processed for **indexing**, and metadata like titles and content are extracted.
7. **Repeat the Process:**
    - This loop continues until the crawler reaches a pre-defined limit or all reachable pages are visited

Example:

Suppose a crawler starts with this seed URL: https://www.university.edu

- The crawler downloads the homepage.
- It finds links to pages like:
    - https://www.university.edu/departments
    - https://www.university.edu/admissions
    - https://www.university.edu/faculty
- These links are added to the queue and visited one by one.
- Each of those pages contains more links (e.g., specific professor profiles), and the crawler continues discovering and storing them.

Major Jobs of a Web Crawler:

1. **Document Discovery:**
   - Finds and collects **new and updated** web content for indexing.
2. **Coverage:**
   - Ensures that a large and relevant portion of the web is crawled.
3. **Freshness Maintenance:**
   - Re-visits previously crawled pages to check for **updates or deletions** using methods like HTTP HEAD requests.
4. **Respect Robots.txt:**
   - Follows rules in a website's robots.txt file, which tells the crawler which pages to access or avoid.
5. **Content Normalization:**
   - Converts and stores collected data in a consistent format for further processing.

Web crawlers use politeness policies for not too busy the web server to the other end. Here, crawler thread is implemented. Describe this policy. What is the importance of politeness policies in web crawling?

Politeness policies are crucial rules followed by web crawlers to ensure that web crawlers behave ethically by not sending too many requests to a web server in a short time, which could **overwhelm or crash** the server. These policies are implemented using **crawler threads** and timing controls.

Crawler Thread Implementation:

1. **Thread-Based Crawling:**
   - A web crawler typically uses **multiple threads** to download pages in parallel.
   - Each thread processes one URL at a time from a **shared URL queue** (also called the frontier).
2. **Host-Based Scheduling:**
   - To follow politeness, the crawler tracks **last-access time** for each domain.
   - Threads are **paused** before requesting another page from the same host, allowing time gaps (e.g., 10 seconds).
3. **Politeness Delay:**
   - A **minimum delay** is enforced between two requests to the same domain (e.g., 5–10 seconds).
   - This avoids flooding a single server with multiple rapid requests.
4. **robots.txt Respect:**
   - Each thread checks the website's **robots.txt** file before crawling.
   - It ensures the thread doesn't access disallowed paths.

Importance of Politeness Policies:

1. **Avoids Server Overload:**

    - By limiting the frequency of requests to the same server, politeness policies prevent crashing or slowing down websites.
2. **Ensures Fair Resource Usage:**
    - Helps crawlers share server bandwidth with **human users** and other bots without interference.
3. **Respects robots.txt:**
    - Politeness includes obeying site-specific rules defined in a robots.txt file, which may **disallow access** to certain pages.
4. **Maintains Crawler Reputation:**
    - Following polite behavior keeps the crawler from being **banned or blacklisted** by web administrators.
5. **Supports Legal and Ethical Crawling:**
    - Prevents the crawler from violating website **terms of service**, which may lead to legal consequences.
6. **Improves Crawling Efficiency:**
    - By pacing requests, crawlers can schedule visits during **low-traffic hours**, reducing server load and avoiding unnecessary blocks.

A crawler following a 10-second delay between requests to the same domain ensures that the server is not overwhelmed, while also remaining compliant with the site's robots.txt restrictions.

Explain step-by-step procedure to retrieve the relevant information from the following Web page: https://bu.ac.bd/?ref=teacher_profile_data_cse

Step 1: Open the Web Page

- Access the webpage using a browser or a simple HTTP request tool.
- Goal: view or fetch the page's HTML content.

Step 2: Analyze HTML Structure

- Inspect the page using browser developer tools.
- Identify where the required information (e.g., teacher name, email) is located—usually inside <div>, <table>, or <span> tags.

Step 3: Parse the HTML

- Use a web scraping tool like **BeautifulSoup** (Python) or browser-based extension.
- Extract the tags that contain data like name, designation, contact.

Step 4: Extract Data Fields

- Focus on key information:
    - Teacher's Name

- Designation
- Department
- Email Address
- Research Interests

Step 5: Clean and Organize

- Remove unnecessary characters or tags.
- Format the data into a structured format (e.g., table, CSV, JSON).

Step 6: Use or Store the Data

- Save it for academic profiling, departmental websites, or search indexing.

Differentiate search engine and search engineers

Search Engine	**Search Engineer**
A **search engine** is a software system that retrieves information from a large collection of data (e.g., the web) based on user queries.	A **search engineer** is a person (software developer or specialist) who designs, builds, and maintains search engines or search systems.
It performs tasks like crawling, indexing, ranking, and retrieving data.	They implement algorithms, optimize indexing, improve ranking, and ensure performance.
Examples include Google, Bing, DuckDuckGo.	Example roles: search backend developer, relevance engineer, ranking model researcher.
Acts as the **product/tool** used by millions of users.	Acts as the **creator/maintainer** of that product or tool.
Focus is on **functionality** and **user experience**.	Focus is on **architecture**, **algorithms**, and **efficiency**.

How do search engines prioritize which pages to crawl first, and what factors influence this decision? How do search engines determine the freshness of a web page, and why is this an important factor in ranking?

Search engines use a **crawler (or spider)** to discover and index web pages. They prioritize which pages to crawl based on several factors:

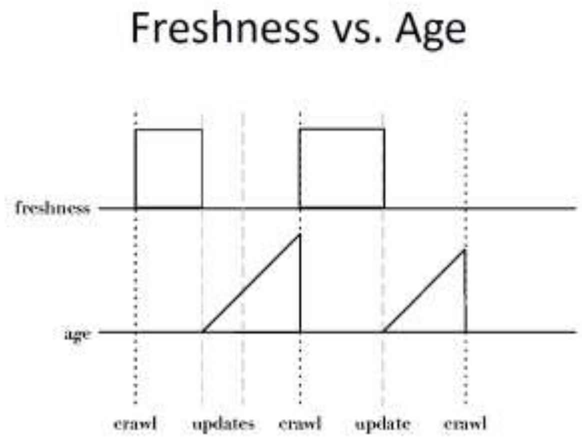
1. Pages with high PageRank or many inbound links are considered more authoritative and are crawled more frequently.
2. Pages known to update frequently (like news sites) are revisited more often.
3. Each site is given a limited number of pages that can be crawled within a time period to avoid overloading servers.
4. Website owners can influence crawling through sitemaps and robots.txt
5. Crawlers use past patterns to decide when to crawl next.

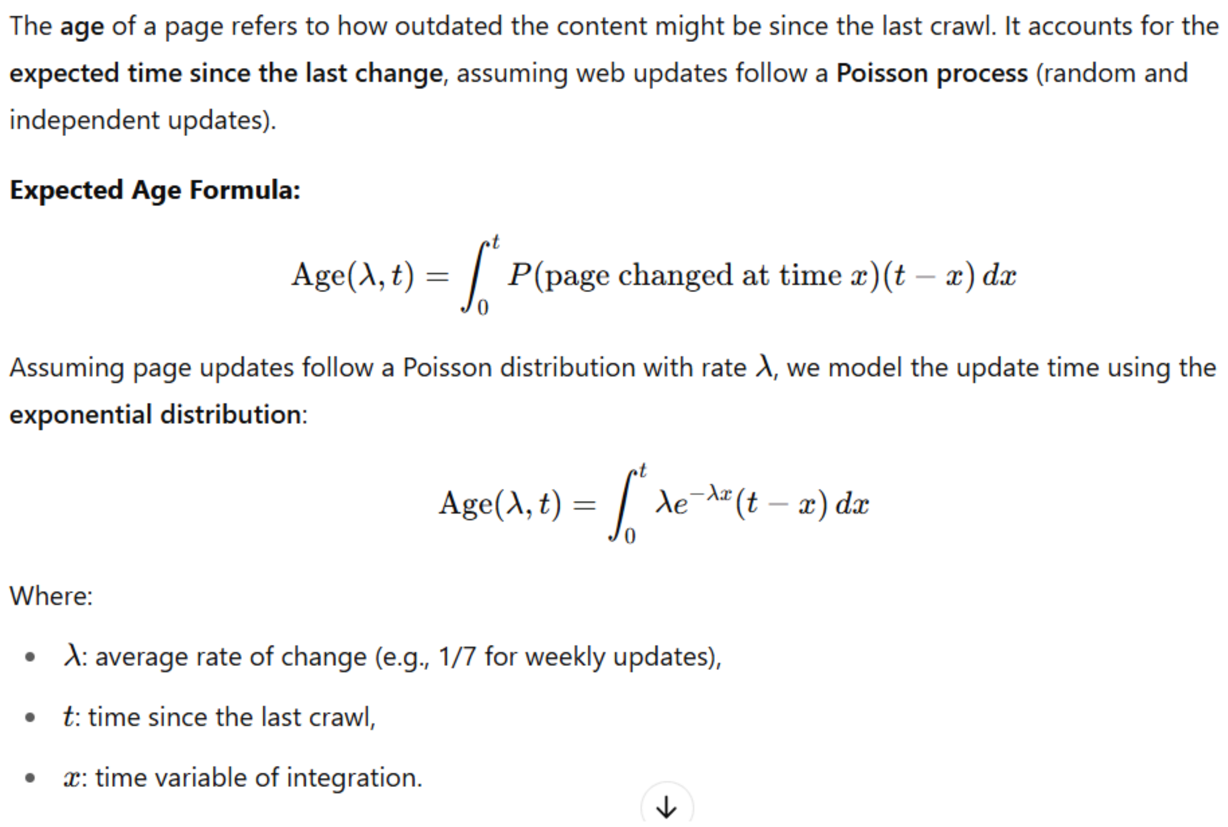
Search engines measure the **freshness** of a web page to determine how current or up-to-date its content is compared to its last crawl. Freshness is important because users prefer the latest information, especially for news, trends, and frequently updated topics.

Determining Freshness:

- Web crawlers periodically revisit pages to check for updates.
- The **HTTP HEAD request** is used to check the "Last-Modified" date without downloading the full page.
- Due to limited resources, not all pages can be checked frequently. So, more important and dynamic pages are prioritized.

However, optimizing only for freshness can lead to poor crawling decisions. Therefore, **age** is used as a more robust metric.





The **age** of a page refers to how outdated the content might be since the last crawl. It accounts for the **expected time since the last change**, assuming web updates follow a **Poisson process** (random and independent updates).

Expected Age Formula:

$$\text{Age}(\lambda, t) = \int_0^t P(\text{page changed at time } x)(t - x)\, dx$$

Assuming page updates follow a Poisson distribution with rate λ, we model the update time using the **exponential distribution**:

$$\text{Age}(\lambda, t) = \int_0^t \lambda e^{-\lambda x}(t - x)\, dx$$

Where:

- λ: average rate of change (e.g., 1/7 for weekly updates),
- t: time since the last crawl,
- x: time variable of integration.

This formula calculates the expected **staleness** of a page and helps search engines decide **when to recrawl** it next.

Define web crawler. What are the two major jobs of any Web Crawler? How do you calculate the age of any Crawler?

A web crawler is a program that systematically browses the web to download pages and discover links for building a searchable index. It begins with a set of initial URLs (seeds) and processes them to retrieve content and discover more links. Crawlers also implement politeness policies to avoid overwhelming web servers and revisit pages periodically to ensure data freshness.

Two Major Jobs of Any Web Crawler:

1. **Fetching Web Pages:**

- The crawler retrieves web pages by sending HTTP requests (usually GET requests) to web servers.
- This allows it to collect and store the content of these pages for indexing and analysis.

2. **Discovering New URLs:**

- While processing downloaded pages, the crawler identifies and extracts hyperlinks (anchor tags).
- These new URLs are added to a queue for future crawling, enabling the crawler to explore the web further.

The age of a crawler refers to the expected age of a web page at a given time after it was last crawled. It is used to measure how outdated the information in the crawler's index is.

The expected age A(t) of a page is given by:

$$\mathrm{Age}(\lambda, t) = \int_0^t \lambda e^{-\lambda x}(t - x)dx$$

Where:

t: Time elapsed since the page was last crawled.

λ: Mean change frequency (e.g., if a page changes once a week, λ=1/7). Pages with higher change frequencies (λ) require more frequent revisits to minimize their age.

In crawling the Web, freshness is important to be maintained. Distinguish between fresh copy and stale copy with proper example.

Search engines measure the **freshness** of a web page to determine how current or up-to-date its content is compared to its last crawl.

- Web pages are constantly added, modified, or deleted.
- A web crawler must revisit pages it has already crawled to check if they've changed, in order to maintain the freshness of its document collection.
- If a previously crawled page changes but is not updated in the search engine, it becomes a stale copy.

Fresh Copy	**Stale Copy**
A **fresh copy** is an up-to-date version of a web page stored in the search engine's index.	A **stale copy** is an older version of a web page that has since changed on the actual website.
It reflects the **current content** of the webpage.	It contains **outdated content** and may mislead users.
Indicates the crawler has **recently revisited** the page.	Indicates the page hasn't been crawled in a **long time**.
Helps maintain **relevance and accuracy** in search results.	Can lead to **irrelevant or incorrect** search results.

Example:

- Suppose this web page: URL: www.example.com/news.html
- On June 1, it says: “New smartphone coming next week.”
- Crawler downloads and stores it on June 1.
- On June 5, webpage changes to: “Smartphone successfully launched.”

Two cases:

- If the crawler revisits and updates on June 5 → Fresh copy
- If it doesn’t revisit or delays too long → Shows outdated text → Stale copy

What is focused crawling? What are the features that must be added to any crawler?

Focused crawling is a type of web crawling strategy where the crawler aims to download only web pages that are relevant to a specific topic or domain.

- Instead of crawling the entire web, it selectively chooses which links to follow based on the relevance of the content.
- Used in vertical search engines or topic-specific applications (e.g., health, legal, academic content).

How It Works:

- Starts with seed URLs related to the topic.
- Analyzes page content using a text classifier.
- Follows links only from relevant pages to stay within the topic.
- Evaluates and scores new URLs based on their likelihood of being relevant.

Example:

If the topic is “Artificial Intelligence,” a focused crawler will prioritize links related to AI, such as:

- www.technews.com/ai-future
- www.university.edu/ai-research

It will skip pages about unrelated topics like cooking or sports.

Features to Add to Any Crawler:

1. **Politeness Policy:**
    - Respect delays between requests and follow robots.txt.
2. **URL Frontier Management:**

    - Efficiently manage the queue of URLs to avoid duplicates and maintain crawl order.
3. **Duplicate Detection:**
    - Detect and avoid crawling exact or near-duplicate content.
4. **Freshness Tracking:**
    - Keep track of when a page was last visited to prioritize updates.
5. **Multithreading:**
    - Use multiple threads for faster crawling while maintaining politeness.
6. **Page Parsing and Link Extraction:**
    - Extract valid hyperlinks from HTML pages to discover new URLs.
7. **Content Filtering (for focused crawling):**
    - Evaluate and rank the relevance of each page to the target topic.
8. **Error Handling & Recovery:**
    - Handle server errors, broken links, and unexpected content gracefully.

What is hard and soft focused crawling?

Hard focused crawling is a strict approach to focused crawling where the crawler only downloads pages that are directly relevant to a specific topic or query.

Characteristics

- Uses a highly specialized text classifier to evaluate the relevance of pages.
- Pages that do not meet the strict relevance criteria are ignored or discarded.
- Suitable for applications requiring precise data, such as niche search engines or domain-specific research.

Soft focused crawling is a more flexible approach where the crawler downloads a broader set of pages, including pages that may not be directly relevant but are likely to link to relevant content.

Characteristics:

- Relies on the assumption that pages related to a topic often link to other relevant pages.
- Prioritizes popular or authoritative pages in the topic domain as seeds.
- Balances exploration and relevance to ensure comprehensive data collection.

Why is compression needed? What are distributed crawling and desktop crawls?

Compression is crucial in web crawling and search engine indexing to:

- **Reduce storage space:** Billions of web pages require massive storage. Compression reduces the size of documents and indexes.
- **Increase speed:** Compressed data loads and transfers faster between systems.

- **Improve efficiency:** Smaller indexes mean faster search and retrieval operations.
- **Save bandwidth:** While fetching and storing large documents, compression minimizes network and disk usage.

Example:
Compressing an inverted index with gap encoding or variable-length encoding can shrink its size by over 50%, allowing faster lookup during a search.

2. What is Distributed Crawling?

Distributed crawling involves using **multiple systems** that work together to crawl the web in parallel.

- Each crawler is assigned different portions of the web (e.g., based on domain or URL hash).
- Allows **scalable**, **fault-tolerant**, and **faster** crawling of billions of pages.
- Commonly used by large search engines like Google, Bing, etc.

Example:
One crawler may handle .edu domains while another handles .org, each running independently but contributing to the same index.

3. What is Desktop Crawling?

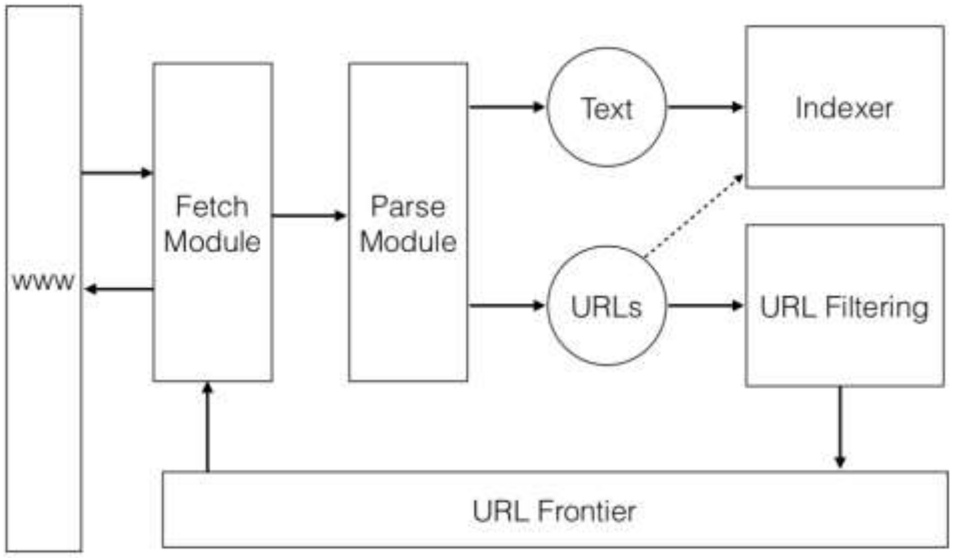
Desktop crawling is the process of indexing and searching files located on a **local personal computer or enterprise system**.

- Used for **personal search engines** or **enterprise search** tools.
- Crawls through local files like PDFs, Word docs, emails, and intranet pages.
- Supports search within an organization or a single device without needing web access.

Example:
Windows Search or tools like "Everything" index your files so you can quickly search documents stored on your own PC.

What is index compression? Summarize on the working of web crawler with its diagram.

Index compression is a technique used to reduce the size of an inverted index while maintaining its functionality for efficient search and retrieval. It is crucial for search engines that handle vast amounts of web data, as smaller indexes require less storage and enable faster query processing. Compression works by removing redundancy in data, optimizing the storage of postings lists and dictionaries. Popular algorithms like delta encoding and variable-length encoding are used to compress postings, while front-coding and hashing reduce dictionary size.



Working of a Web Crawler

A web crawler is a program that systematically browses the web to download pages and discover links for building a searchable index. It begins with a set of initial URLs (seeds) and processes them to retrieve content and discover more links. Crawlers also implement politeness policies to avoid overwhelming web servers and revisit pages periodically to ensure data freshness.

Steps:

1. **Initialization:** Start with a set of seed URLs and add them to a URL queue (also called a frontier).
2. **Fetching Pages:** Send HTTP requests, typically GET requests, to download the content of each URL.
3. **Parsing Pages:** Extract hyperlinks from the downloaded content and identify new URLs for crawling.
4. **Politeness Policy:** Follow rules like adding delays between requests to the same server to avoid overloading it.
5. **Revisiting Pages:** Periodically revisit previously crawled pages to check for updates and maintain freshness.
6. **Storage:** Save the downloaded content and metadata for further processing, such as indexing.

Analyze with the example about the sitemaps

A **sitemap** is an XML file that lists the URLs of a website's pages to inform search engines about the structure and content of the site. It helps crawlers discover and index web pages more efficiently.

Purpose of a Sitemap:

- Helps search engines **find all pages**, even those not linked directly.
- Provides **metadata** about each URL (e.g., last update date, change frequency, priority).
- Improves **crawling efficiency**, especially for large or complex sites.
- Supports SEO by ensuring **important pages get crawled and indexed**.

```
<url>
  <loc>https://www.example.com/</loc>
  <lastmod>2024-06-15</lastmod>
  <changefreq>monthly</changefreq>
  <priority>1.0</priority>
</url>
```

- <loc>: Page URL.
- <lastmod>: Date when the page was last updated.
- <changefreq>: Expected change rate of the page (e.g., daily, weekly).
- <priority>: Priority of the page compared to others on the same site (range: 0.0 to 1.0).

This helps crawlers decide:

- Which pages to crawl,
- How often to revisit, and
- Which pages are more important.

How do search engines handle duplicate content across different websites? What is PageRank, and how does it impact search results?

Search engines handle duplicate content across different websites by employing **different duplicate detection techniques**.

- **Exact Duplicate Detection:**
  Search engines use techniques like checksums (e.g., CRC) to generate a unique fingerprint for each document. If two pages have the same checksum, they are likely exact duplicates. These are filtered out during crawling or indexing to save resources.
- **Near-Duplicate Detection:**
  Near-duplicates have similar content with minor differences (e.g., formatting). Search engines use similarity measures like cosine similarity or shingling to detect them. Pages are considered near-duplicates if a high percentage (e.g., >90%) of words match.
- **Duplicate Handling Strategy:**
  From a group of duplicates, the most authoritative or relevant version is retained. Others are either ignored or ranked lower to prevent cluttering the search results.

PageRank is an algorithm developed by Google that assigns a score to each web page based on the number and quality of incoming links.

Each link from one page to another acts as a vote. Pages that receive links from many high-quality or highly-ranked pages get a higher PageRank. This ranking is calculated iteratively across the web graph.

Pages with higher PageRank are considered more authoritative and are more likely to appear near the top of search results. However, modern search engines also combine PageRank with other factors like content relevance, freshness, and user behavior.

What is 'noise' in web pages, and why is its removal important for search engine indexing?

Noise in web pages refers to elements such as ads, navigation menus, headers, footers, and unrelated links or images that are not part of the main content.

Importance of Removal:

- Noise can **mixed the relevance** of a page during indexing.
- It may lead to **inaccurate ranking**, where less useful pages appear higher.
- Removing or reducing noise helps the search engine **focus on meaningful content**, improving retrieval quality.

Techniques like content block detection are used to **identify and ignore non-content elements** during indexing.

Discuss the importance of document conversion and character encoding in web crawling, explaining why crawlers need to convert documents and how Unicode addresses encoding challenges.

1. Document Conversion:

- Web crawlers collect documents in various formats like HTML, XML, PDF, Word, etc.
- To index and process these uniformly, crawlers **convert them into a consistent format**, typically plain text with metadata (e.g., XML).
- This ensures all documents can be analyzed using the same IR techniques.

2. Character Encoding:

- Web content uses many encoding types (e.g., ASCII, UTF-16).
- Without proper decoding, crawlers may **misinterpret characters**, especially in multilingual content.
- **Unicode (e.g., UTF-8)** provides a universal standard, supporting text in all major languages, ensuring accurate text extraction and indexing.

Explain why storing downloaded pages is necessary for crawlers, how systems efficiently store millions of web pages (e.g., BigTable).

1. Importance of Storing Pages:

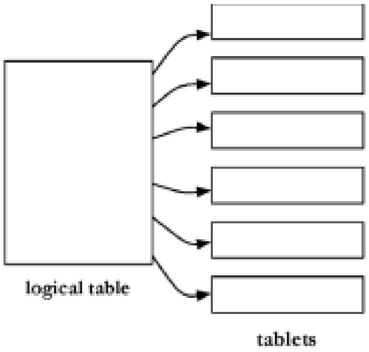
- Crawlers need to **store downloaded web pages** to:
  - Revisit them later for updates (freshness checks)
  - Extract metadata and links
  - Support re-indexing without re-downloading
  - Perform content analysis (e.g., duplicate detection, link analysis)

2. Efficient Storage Using BigTable:

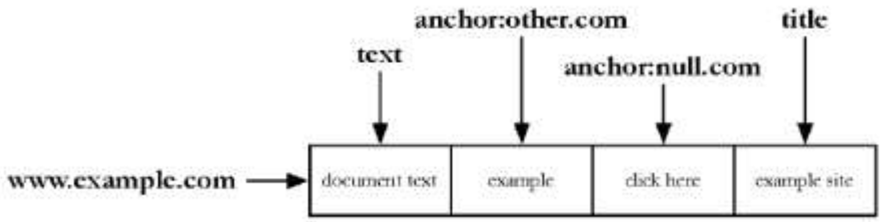
- **BigTable** is Google's scalable, distributed document storage system.
- Designed to **store, locate, and update** web pages across large collections.
- Built to run on **inexpensive hardware** with fault tolerance.

Key Features of BigTable:

- **Row-based storage:** Each row stores a single web page.
- **(Row key, column key, timestamp)** structure uniquely identifies every data cell.
- **Tablet partitioning:** Rows are split into **tablets**, distributed across servers.



- **Replication & Logs:** Changes are recorded in a **transaction log**; replicated storage enables fast recovery.
- **Flexible columns:** Allows many columns per row; column groups improve disk access efficiency.



Chapter 4

Explain Zipf’s Law and give an example to illustrate the law.

Zipf’s Law is an empirical law that describes the relationship between the rank of a word in a corpus and its frequency of occurrence. According to this law, the frequency of a word is inversely proportional to its rank in the frequency table.

That is,

$f(r) \approx k / r$

Where:

- $f(r)$ is the frequency of the word
- r is the rank of the word
- k is a constant specific to the corpus

This means that:

- A small number of words occur very frequently (e.g., “the”, “of”, “and”)
- A large number of words occur very rarely (some only once in the whole collection)

3. Example:

Suppose we analyze a large English text and find these word frequencies:

Rank	Word	Frequency
1	the	10,000
2	of	5,000
3	and	3,333
4	to	2,500

Here, each word’s frequency roughly follows 1/r pattern:

- 2nd is half of 1st,

- 3rd is one-third of 1st,
- 4th is one-fourth of 1st, and so on.

Importance in IR:

- Helps in **indexing and text compression**.
- Guides decisions on **stop words** and **term weighting**.
- Justifies using **logarithmic term frequencies** in ranking algorithms like TF-IDF.

Explain Heaps' Law and give an example to illustrate the law.

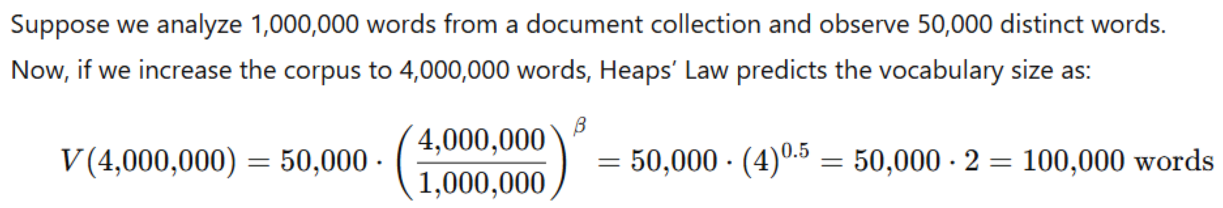
Heaps' Law describes the growth of vocabulary (number of distinct words) as the size of the text corpus increases. It states that as more documents or words are added to a corpus, the vocabulary size continues to grow, but at a decreasing rate.

The law is represented by the formula:

$V(n)=k\cdot n^{\beta}$

Where:

- V(n) = number of distinct words (vocabulary size)
- n = total number of words (tokens) in the corpus
- k, β = constants depending on the corpus . (usually, $10\leq k\leq 100$, $0.4\leq\beta\leq 0.6$)



Suppose we analyze 1,000,000 words from a document collection and observe 50,000 distinct words.

Now, if we increase the corpus to 4,000,000 words, Heaps' Law predicts the vocabulary size as:

$$V(4{,}000{,}000) = 50{,}000 \cdot \left(\frac{4{,}000{,}000}{1{,}000{,}000}\right)^{\beta} = 50{,}000 \cdot (4)^{0.5} = 50{,}000 \cdot 2 = 100{,}000 \text{ words}$$

This shows that vocabulary **grows slowly** as more data is added.

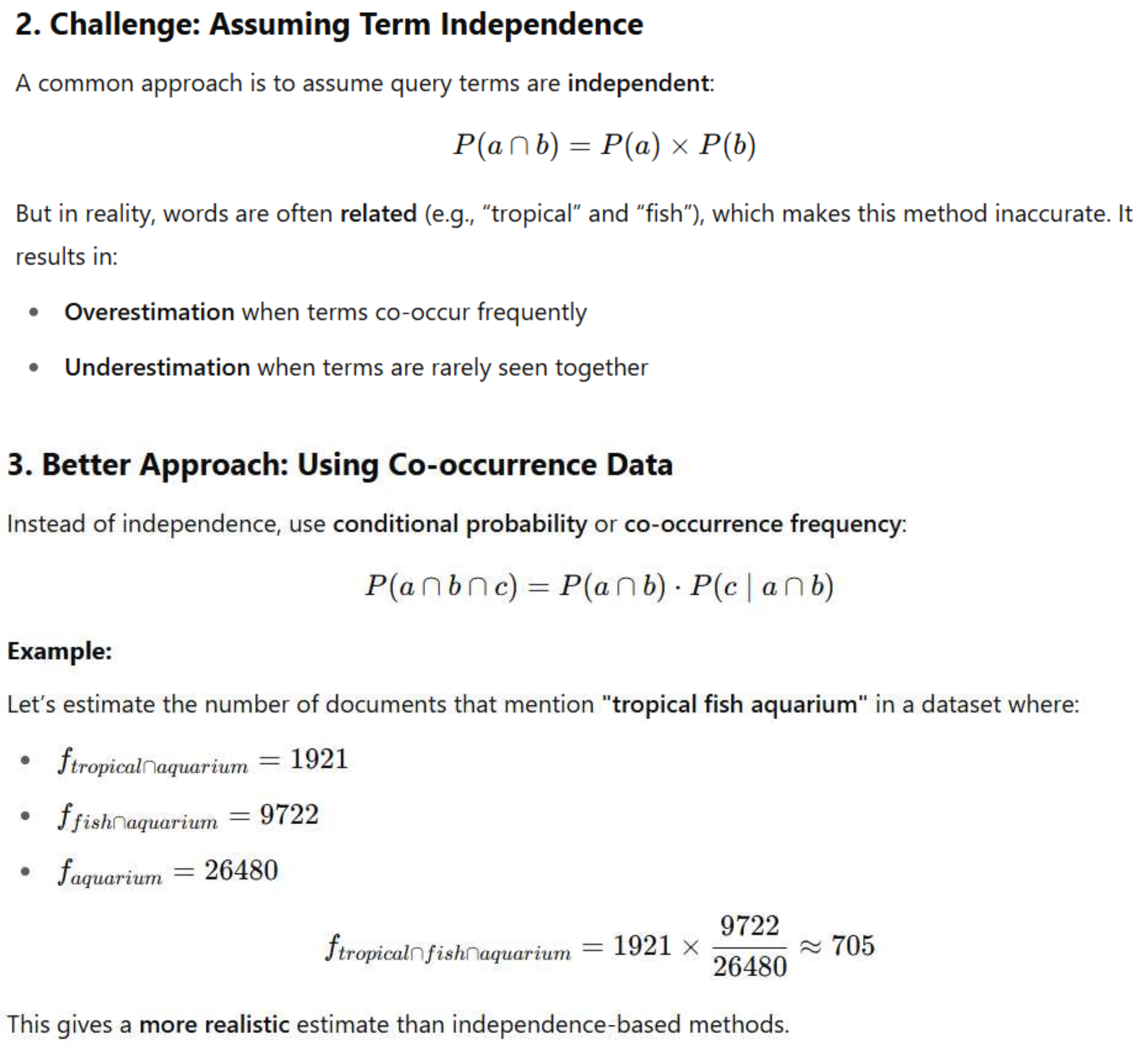
Implications:

- New words continue to appear as the corpus grows.
- Important for estimating storage needs and index size.
- Helps design scalable indexing systems in Information Retrieval.

Explain the challenges of estimating result set size for multi-term queries, especially when assuming term independence, and discuss how these estimations can be improved.

Result set size estimation refers to predicting how many documents match a multi-term query When a user submits a multi-term query (e.g., "tropical fish aquarium"), the search engine must estimate how many documents match **all** query terms. Estimating this size helps in:

- Optimizing ranking and resource allocation
- Deciding whether to perform full or partial evaluations



2. Challenge: Assuming Term Independence

A common approach is to assume query terms are **independent**:

$$P(a \cap b) = P(a) \times P(b)$$

But in reality, words are often **related** (e.g., “tropical” and “fish”), which makes this method inaccurate. It results in:

- **Overestimation** when terms co-occur frequently
- **Underestimation** when terms are rarely seen together

3. Better Approach: Using Co-occurrence Data

Instead of independence, use **conditional probability** or **co-occurrence frequency**:

$$P(a \cap b \cap c) = P(a \cap b) \cdot P(c \mid a \cap b)$$

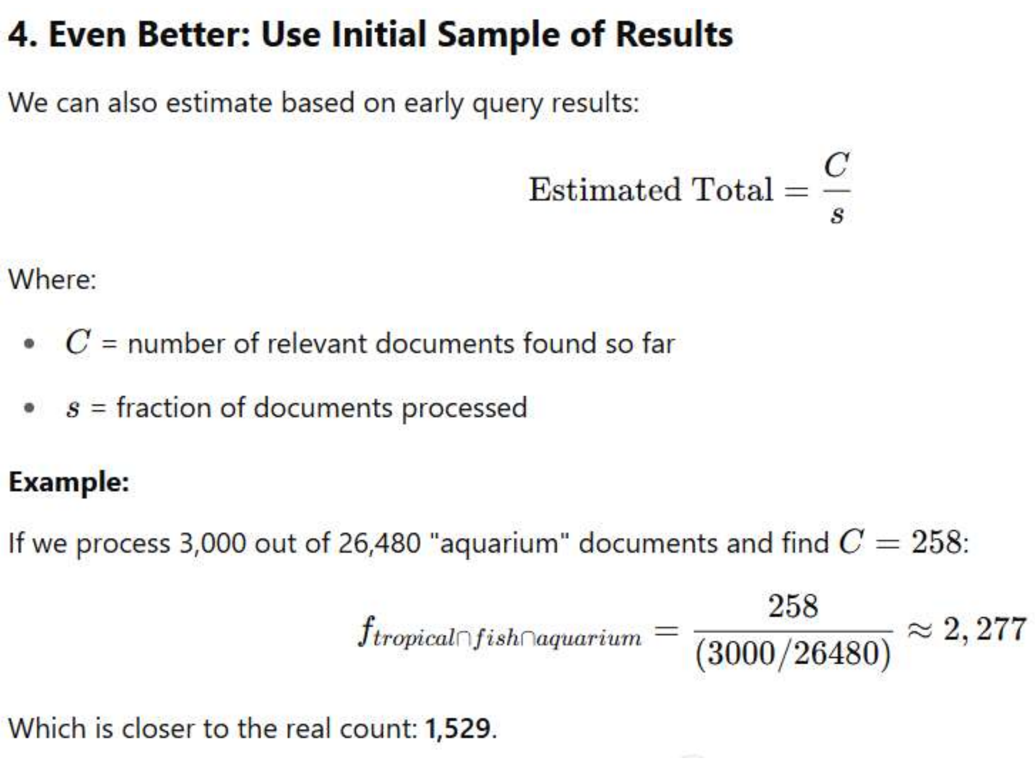
Example:

Let’s estimate the number of documents that mention **"tropical fish aquarium"** in a dataset where:

- $f_{tropical \cap aquarium} = 1921$
- $f_{fish \cap aquarium} = 9722$
- $f_{aquarium} = 26480$

$$f_{tropical \cap fish \cap aquarium} = 1921 \times \frac{9722}{26480} \approx 705$$

This gives a **more realistic** estimate than independence-based methods.



4. Even Better: Use Initial Sample of Results

We can also estimate based on early query results:

$$\text{Estimated Total} = \frac{C}{s}$$

Where:

- C = number of relevant documents found so far
- s = fraction of documents processed

Example:

If we process 3,000 out of 26,480 "aquarium" documents and find $C = 258$:

$$f_{tropical \cap fish \cap aquarium} = \frac{258}{(3000/26480)} \approx 2,277$$

Which is closer to the real count: **1,529**.

Define tokenization, explain why it is more complex than it seems in text processing, and describe the main challenges and steps involved in the tokenization process for search engines.

Tokenization is the process of breaking a stream of text into individual units called tokens, which usually correspond to words or meaningful terms. It is the first step in text processing for search engines.

Why It Is Complex:

Although it seems simple, tokenization becomes difficult due to language ambiguity, punctuation, special characters, and context-dependent rules. For example

Challenges:

1. Ambiguity in word boundaries: e.g., “New York” (one phrase or two words?)

2. Hyphenated and compound words: e.g., “e-mail”, “pre-diabetes”, “Mazda RX-7”

3. Punctuation and special symbols: e.g., “Ph.D.”, “I.B.M.”, URLs, hashtags

4. Case sensitivity: e.g., “Apple” (company) vs. “apple” (fruit)

5. Apostrophes and possessives: e.g., “can’t”, “Bob’s”, “80’s”

6. Languages without spaces: e.g., Chinese, Japanese—no clear word boundaries

Tokenization Steps:

- Parsing: Identify useful content sections in documents (e.g., removing tags, metadata).
- Lowercasing: Convert all text to lowercase for uniformity.
- Splitting: Break text using delimiters (spaces, punctuation, etc.).
- Normalization: Handle hyphens, apostrophes, abbreviations.
- Filtering: Optional removal of unwanted tokens (e.g., short strings or stopwords).
- Consistency: Apply the same rules to both documents and user queries.

Describe stemming, including the differences between Porter and Krovetz stemmers.

Stemming is a text processing technique that reduces words to their root or base form (called the stem), aiming to group morphological variants (e.g., "running," "runs," "ran") under one common form ("run"). This helps improve recall in Information Retrieval by matching different forms of a word.

Porter Stemmer:

- An algorithmic stemmer developed in the 1970s.
- Uses a series of rules to iteratively remove suffixes (e.g., "ing," "ed," "s").
- Produces stems that may not be real words (e.g., "running" → "run," but "supplies" → "suppli").
- Known for simplicity and wide use in IR experiments.
- May produce errors that are hard to fix or modify.

Krovetz Stemmer:

- A hybrid approach combining dictionary lookup and suffix removal.
- Checks if the word exists in a dictionary; if yes, leaves it or replaces with exceptions.
- Otherwise, removes suffixes and verifies again with the dictionary.
- Produces actual words as stems rather than just stems.
- Lower false positive rate but somewhat higher false negatives compared to Porter.

Example:

- “running” → “run”
- “supplies” → “supply” (real word)
- “cats” → “cat”

What is a phrase in Information Retrieval, and why are phrases more effective than single-word terms?

A phrase in Information Retrieval is a group of two or more words that together convey a specific meaning, such as "black sea" or "New York City".

Recognizing phrases in text is a key text processing step in Information Retrieval (IR) because phrases like "New York City" or "climate change" carry more precise meaning than individual words.

There are three common approaches for phrase recognition:

1. **Part-of-Speech (POS) Tagging:**

A POS tagger labels each word with its grammatical role (noun, verb, adjective, etc.). Using this, syntactic rules can identify meaningful phrases like noun phrases or adjective-noun combinations.

Example: In “black cat”, “black” is tagged as adjective and “cat” as noun → a valid noun phrase.

2. **Word N-grams:**

This method generates sequences of N consecutive words (e.g., bigrams, trigrams). Frequently occurring combinations are treated as phrases.

Example: From “The big apple”, the trigram “The big apple” might be recognized as a phrase.

3. **Proximity Operators and Word Positions:**

During indexing, the positions of words are stored. At query time, search engines use proximity operators (like NEAR, WITHIN) to identify words that appear close together. If query words appear adjacent or near each other, they are treated as a phrase match.

Phrases are more effective than single-word terms because they provide context, reduce ambiguity, and lead to more precise search results. For example, searching for "big apple" retrieves documents about New York, while the separate words "big" and "apple" may return unrelated results.

What are N-grams in text processing? Give examples and explain how they are generated.

N-grams are sequences of N consecutive words from a text used in Information Retrieval. They are generated using a sliding window approach.

Examples:

- Unigram (1-word): "search"

- Bigram (2-word): "search engine"
- Trigram (3-word): "search engine optimization" For a sentence like “The cat sleeps”:
- Bigrams: “The cat”, “cat sleeps”
- Trigrams: “The cat sleeps” N-grams help identify common word patterns and improve query matching.

How are phrases recognized in text processing using N-grams and POS tagging?

Phrases can be recognized using two main methods:

1. N-grams: Predefined sequences of N words are indexed, and phrases are matched based on frequent combinations (e.g., “New York City” as a trigram).
2. POS Tagging (Part-of-Speech): A syntactic analysis is done using grammar rules to detect patterns like adjective + noun or proper noun sequences.
3. Example: In “black cat”, POS tagging labels “black” as an adjective and “cat” as a noun, identifying it as a noun phrase.

Compare N-gram and POS tagging approaches for phrase recognition in Information Retrieval.

Feature	**N-grams**	**POS Tagging**
Basis	Word sequence	Grammatical structure
Language-aware	No	Yes
Accuracy	May include meaningless phrases	Identifies meaningful phrases
Speed	Fast (no parsing needed)	Slower (requires parsing)
Example Output	"new york", "york city"	"New York City" (Proper Noun Phrase)

Both methods are useful. N-grams are simple and efficient, while POS tagging offers better precision in identifying valid phrases.

Discuss how document structure and markup (e.g., HTML/XML) aid in search indexing, explaining why some parts of documents are more important than others in search and how a document parser uses markup.

Markup languages like HTML and XML provide structural information about documents, which is essential in Information Retrieval for indexing and ranking. Search engines use this structure to identify important content within a page.

Importance of Document Structure:

Not all parts of a document carry equal weight. Some sections provide key information, while others are less relevant. For example:

- Titles (<title>) often summarize the topic.
- Headings (<h1>, <h2>) indicate main topics or subtopics.
- Anchor text in links (<a>) often describes the destination content.
- Bold or italicized words (<b>, <i>) may emphasize important terms.
- Metadata (<meta>) contains keywords, descriptions, and language settings.

Role of Document Parser:

A document parser processes markup to identify and extract structural elements. It uses tags to:

- Isolate text in important sections (e.g., headers, anchor text).
- Ignore irrelevant parts (e.g., navigation bars, ads).
- Tag content to treat it differently during indexing. E.g., Indexing title terms with higher weight or handling anchor text as descriptors of linked pages.

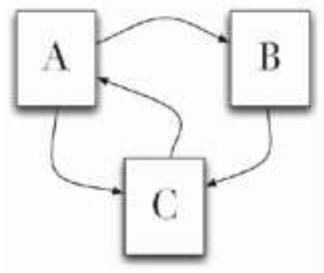
Example:
In a web page:

```
<title>Hello 2025</title>
<p>Explore more...</p>
<a href="new.html">View Other place</a>
```

A search engine will give more importance to “View Other place” during indexing than to the paragraph content.

Explain how PageRank works in web search. Describe the random surfer model, damping factor, handling of dangling links, and the iterative calculation process. Also explain the role of anchor text and link analysis in search engine ranking.

PageRank is a link analysis algorithm developed by Google to measure the importance of web pages based on their link structure. It assumes that a page is important if many other important pages link to it.



Random Surfer Model:

This model imagines a user randomly clicking on links. At each step:

- With probability λ (called the damping factor), the user jumps to a random page.
- With probability 1 − λ, the user follows a link on the current page.

Damping Factor (λ):

Typically set to 0.15, this factor prevents the surfer from getting stuck in loops or on pages without links. It ensures all pages have a minimum chance of being visited.

Dangling Links:

Pages with no outgoing links are called dangling nodes. They can trap the random surfer. To handle this, the algorithm redistributes their PageRank evenly across all pages.

Iterative Calculation:

PageRank is computed iteratively. Each page’s rank is updated based on the ranks of pages linking to it:

$$PR(u) = \frac{\lambda}{N} + (1 - \lambda). \sum_{v \in B_u} \frac{PR(v)}{L_v}$$

where:

- N = total number of pages
- v ∈ Bu (set of pages linking to u)
- Lv = number of outlinks from v

The process repeats until ranks converge (i.e., changes between iterations become very small).

Role of Anchor Text & Link Analysis:

- Anchor text (the clickable part of a hyperlink) helps describe the target page. It is used as additional text for indexing and ranking.
- Link analysis helps detect authoritative and popular pages. Pages with many high-quality inlinks get higher PageRank.

Describe the implementation steps of the PageRank algorithm.

The implementation of PageRank involves preparing the link structure and then applying iterative calculations to compute each page’s rank.

Step 1: Link Data Preparation

- Extract all links from documents using a parser (e.g., from HTML anchor tags).
- Filter out links that point to non-corpus pages.
- Store links as (source, destination) pairs.
- Initialize PageRank values: each page gets an equal value of 1/N, where N is the total number of pages.

Step 2: Iterative Rank Calculation

For each page u, compute the new PageRank using:

$$PR(u) = \frac{\lambda}{N} + (1 - \lambda). \sum_{v \in B_u} \frac{PR(v)}{L_v}$$

Where:

- λ = damping factor (typically 0.15)
- v ∈ Bu (pages linking to u)
- Lv = number of outlinks from page v

Steps:

1. For each page, distribute its rank across its outlinks.
2. Sum all incoming contributions to compute the new PR for each page.
3. Add damping factor and handle dangling links (pages with no outlinks).
4. Repeat until ranks converge.

Step 3: Convergence Check

- Use a threshold τ (e.g., 0.01) to check if PageRank values have stabilized: ||new − old|| < τ
- Continue iterations until this condition is met for all pages.