Indexing and Searching Techniques

1. Explain the different kinds of searching techniques in IR.

1. Boolean Search

- Uses logical operators (AND, OR, NOT) to combine terms.
- Retrieves documents that exactly match the logical condition.
  - AND → retrieves docs containing all terms.
  - OR → retrieves docs containing at least one term.
  - NOT → excludes docs with a certain term.
- Advantage: Simple, fast, precise control.
- Disadvantage: No ranking, results may be too broad or too narrow.
- Example: "Cloud AND Security" → retrieves only documents containing both words.

2. Sequential Search

- Also called linear search; checks each document one by one.
- Query is compared directly with document text.
- Advantage: Very easy to implement, no preprocessing required.
- Disadvantage: Very slow for large document collections.
- Example: Searching for the word "Education" in an unindexed text file.

3. Serial Search

- Similar to sequential, but applied across multiple sources in order.
- First source is searched, then the next, until all are covered.
- Advantage: Good for distributed data or multiple libraries.
- Disadvantage: Time-consuming, duplicates may appear.
- Example: Searching "Cloud Computing" across different university library databases.

4. Cluster-Based Retrieval

- Cluster-based retrieval is a searching technique in IR where documents are grouped into clusters based on similarity (like topics, keywords, or features).
- The idea comes from the Cluster Hypothesis:
  Documents that are similar to each other tend to be relevant to the same query.

Process

1. Clustering Phase
   - The document collection is divided into groups (clusters).
   - Documents inside a cluster are similar, but different clusters are dissimilar.
   - Algorithms used: Single Link, Complete Link, k-means, etc.
2. Query Processing Phase
   - When a user submits a query, instead of searching the whole collection, the system first finds the most relevant cluster(s).
   - Then, documents are retrieved from those clusters.

Example :- Suppose we have documents about:

- Cloud Computing, Artificial Intelligence, Healthcare, Education.

Clusters formed:

- Cluster 1 → Cloud Computing docs
- Cluster 2 → Artificial Intelligence docs
- Cluster 3 → Healthcare docs
- Cluster 4 → Education docs

If the user query is "Cloud in Healthcare", the system looks at Cluster 1 (Cloud) and Cluster 3 (Healthcare), then retrieves relevant docs from those clusters, instead of scanning the entire collection.

5. Query Languages

- Formal languages used to frame queries for retrieval.
- Examples: Boolean queries, SQL, probabilistic query languages.
- Advantage: Allows complex and powerful searches.
- Disadvantage: Users must know the query syntax.
- Example: SQL → SELECT * FROM docs WHERE year BETWEEN 2020 AND 2024;

6. Pattern Matching

- Finds text using patterns, wildcards, or regex.
- Can be exact match or approximate match.
- Advantage: Useful for searching word variations.
- Disadvantage: Can be slow for big datasets.
- Example: Regex comp.* → matches computer, computing, computation.

7. Structural Queries

- Queries are made on document structure (tags, metadata, fields).
- Works well for XML, JSON, HTML documents.
- Advantage: Very precise, retrieves specific parts of documents.
- Disadvantage: Needs structured/semi-structured data.
- Example: XML query //book/title → retrieves only book titles.

2. List and explain the types of queries:-

A query is the expression of a user's information need. In ISR, queries can take many forms depending on how the user interacts with the system. The main types are:

1. Keyword Queries

- Most basic and widely used query type.
- User specifies one or more keywords.
- The system retrieves documents that contain those keywords (either exactly or through indexing).
- Often used in search engines, library catalogs, and databases.

- Advantage: Simple to use, requires no special knowledge.
- Limitation: May return too many or too few results, and cannot express complex relationships.
- Example: Query = "Cloud Computing" → retrieves all documents containing the words Cloud and Computing.

2. Boolean Queries

- Uses logical operators (AND, OR, NOT) to combine search terms.
- Provides strict, rule-based retrieval.
- Results are exact matches according to the logical expression, not ranked by relevance.

- Advantage: Very precise, allows users to control inclusion/exclusion.
- Limitation: No ranking of results, can be too strict or too broad.
- Example:
  - "Cloud AND Security" → retrieves only documents that contain both words.
  - "AI OR Machine Learning" → retrieves documents containing either term.
  - "IoT NOT Healthcare" → excludes documents related to healthcare.

3. Natural Language Queries

- User writes queries in normal human language.
- The system uses Natural Language Processing (NLP) to interpret the meaning.
- Common in modern search engines and virtual assistants.

- Advantage: Very user-friendly, no need to know query syntax.
- Limitation: Accuracy depends on system's NLP capability.
- Example: Query = "What are the applications of IoT in education?" → system interprets the meaning and retrieves relevant documents.

4. Proximity Queries

- Search is based not just on the presence of keywords, but also their closeness in text.
- Useful for phrase searching or context-based retrieval.
- Often expressed with operators like NEAR, WITHIN, or distance numbers.

- Advantage: Improves accuracy by considering context.
- Limitation: More complex to use, may not be supported in all systems.
- Example: “Cloud NEAR Computing” → retrieves documents where “Cloud” and “Computing” appear close to each other, e.g., in the same sentence.

5. Range Queries

- Useful for numerical or date fields.
- Retrieves documents with values within a given range.
- Often used in databases and digital libraries.

- Advantage: Allows filtering based on numeric criteria (like year, price, marks).
- Limitation: Only useful when structured data is available.
- Example: “Year BETWEEN 2020 AND 2024” → retrieves all documents published between 2020 and 2024.

3.Explain the concept of suffix trees in information retrieval

- A suffix tree is a data structure used to store all suffixes of a given string (text/document).
- Each path from the root to a leaf in the tree represents a suffix of the text.
- It allows very fast searching of substrings, pattern matching, and indexing.

Why Used in IR?

- In ISR, suffix trees help in:
    1. Substring searching – finding words or phrases inside documents quickly.
    2. Pattern matching – matching complex text patterns (useful in DNA search, plagiarism detection, etc.).
    3. Efficient indexing – building indexes for large text collections.

Working Example :- Suppose we have the text: “banana”

All suffixes are:-

- banana
- anana
- nana
- ana
- na
- a

These suffixes are stored in a tree form (trie-like structure).

- Root → “b”, “a”, “n” branches.
- From each branch, paths continue to form complete suffixes.

So, if we search for substring “ana”, we directly follow the path from root → “a” → “na” → found!

Advantages

- Very fast substring/pattern search (time complexity: O(m), where m = length of query).
- Useful for large text collections in IR.
- Can handle repeated patterns efficiently.

Disadvantages

- High memory usage (takes more space than the original text).
- Complex to implement compared to normal indexing.

Applications in ISR

- Search engines for substring search.
- Plagiarism detection – finding copied text portions.
- Bioinformatics – DNA/protein sequence matching.
- Text mining and data compression.

4. Concept of inverted index file. How can be used in information retrieval.

1. Concept of Inverted Index

- An inverted index is the most widely used data structure in Information Retrieval systems (like search engines).
- It is called inverted because instead of storing documents → terms, it stores terms → documents.
- It maps each term/keyword to a list of documents in which that term appears.
- Similar to the index at the back of a book, where each word points to the pages (documents) where it occurs.

2. Structure of an Inverted Index

It has two main parts:

1. Dictionary (Vocabulary)
    - Stores all unique terms from the document collection.
2. Posting Lists
    - For each term, a list of documents (and positions) where the term occurs.

Example:- Suppose we have 3 documents:

- D1: “Cloud computing is powerful”
- D2: “Big data uses cloud”
- D3: “Cloud is the future”

Inverted Index:

Term	Posting List (Documents)
cloud	D1, D2, D3
computing	D1
is	D1, D3
powerful	D1
big	D2
data	D2
uses	D2
future	D3

3. How it is Used in IR

1. Query Processing
    - User enters a query, e.g., “cloud AND data”.
    - System looks up the posting lists of cloud and data.

2. Matching
    - Finds the intersection of posting lists.
    - Here: “cloud” → {D1, D2, D3}, “data” → {D2}.
    - Common = {D2}.
3. Ranking (Optional)
    - Documents can be ranked using term frequency (TF), inverse document frequency (IDF), or probabilistic models.
4. Retrieval
    - Final ranked or filtered list is shown to the user.

Advantages

- Very efficient for large document collections.
- Supports Boolean queries, phrase queries, ranked retrieval.
- Simple structure, easy to implement.

Disadvantages

- Needs more storage space for posting lists.
- Must be updated frequently when documents are added/removed.

5.Explain the concepts of signature files in information retrieval.

- A signature file is a filtering technique used in Information Retrieval before searching the full documents.
- Instead of searching through every document, the system searches through signatures (compact representations) of documents.
- It helps reduce the search space and speeds up retrieval.

2. Concept

- Each document is represented by a bit string (called a signature).
- A hash function converts the keywords of a document into bits set in the string.
- When a user enters a query, a query signature is created in the same way.
- Documents whose signatures match the query signature are considered candidates.
- Only these candidate documents are checked in detail for exact matches.

This process is called filtering.

3. Example :- Suppose we have 3 documents:

- D1: “Cloud computing is powerful”
- D2: “Big data uses cloud”
- D3: “Future of cloud and computing”

Steps:

1. Each keyword (cloud, computing, data, etc.) is hashed into a bit position in a fixed-length bit string.
2. If the word is present in the document → the corresponding bit is set to 1.

Example (bit strings of length 8):

- D1 signature: 10100110
- D2 signature: 11010001
- D3 signature: 11100100

Query: “cloud computing” → query signature = 10100100.

- Compare with doc signatures → D1 & D3 match → these are candidate documents.

Types of Signature Files

1. Word signature – one signature per word, combined to form document signature.
2. Document signature – one bit string for the whole document.

Advantages

- Compact representation of documents.
- Faster than scanning full documents.
- Easy to generate and update.

Disadvantages

- May produce false matches (false positives) since signatures are approximate.
- Needs extra step of checking candidate documents.
- Less efficient than inverted index for very large collections.

Applications in ISR

- Used in database systems and older IR systems for fast filtering.
- Applied in multimedia retrieval (images, audio) where inverted index is harder to apply.

6. Explain exhaustivity and specificity with respect to Index term weighting.

1. Exhaustivity

- Means: How much of the document's content is covered by the index terms.
- High exhaustivity = many terms → covers all topics in the document.
- Low exhaustivity = few terms → may miss some topics.

Example: Doc: "Cloud computing in healthcare and education."

- High Exhaustivity → {cloud computing, healthcare, education, applications}
- Low Exhaustivity → {cloud computing}

2. Specificity

- Means: How exact or precise the index terms are.
- High specificity = very focused terms → narrow but accurate.
- Low specificity = broad terms → general and less accurate.

Example: Same doc:

- High Specificity → {cloud computing in healthcare, cloud computing in education}
- Low Specificity → {technology, computers}

Link with IR

- Exhaustivity → like recall (finding ALL relevant docs).
- Specificity → like precision (finding ONLY relevant docs).

7.Compare and contrast the basic concepts, strengths, and limitations of the Boolean Model, Vector Model, and Probabilistic Model. Provide insights into when and why each model would be chosen to optimize search results, considering factors like ranking accuracy and complexity.

1. Boolean Model

The Boolean model uses logical operators (AND, OR, NOT) to match queries with documents. Documents are either retrieved or not retrieved (binary result). It is simple and fast but does not provide ranking of results.

Representation

- Each document is a set of terms (often binary: term present/absent).
- The collection is an inverted index mapping each term → list of document IDs.

Query processing

- Parse the Boolean expression and perform **set operations** on posting lists:
    - AND → intersection, OR → union, NOT → set difference.

- Result is an **unordered** set of documents that satisfy the expression.

Mini example Docs:

- D1: “cloud computing is powerful”
- D2: “big data uses cloud”
- D3: “cloud is the future”
  Query: cloud AND data → posting(cloud)={D1,D2,D3}, posting(data)={D2} → result {D2}.

Strengths

- Simple, fast, and predictable.
- Excellent when exact control is needed (legal/medical search, compliance).

Limitations

- No ranking—all matches look equally good.
- Queries can be hard to craft; easy to get too many or too few results.
- No notion of partial relevance.

When to choose

- Small/medium collections with trained users; exact inclusion/exclusion is crucial; auditability matters.

Advantages

- Simple and fast.
- Easy to understand.
- Precise control (include/exclude terms exactly).

Disadvantages

- No ranking → all results equal.
- Partial matches ignored.
- Queries can become complex.

2. Vector Space Model (VSM)

In VSM, documents and queries are represented as vectors in multi-dimensional space. The similarity between them is measured using cosine similarity, and results are ranked based on relevance scores. It supports partial matching and ranking but requires weighting schemes like TF–IDF.

Uses **Cosine Similarity** to rank documents:

$$\text{sim}(q, d) = \frac{q \cdot d}{|q| \cdot |d|}$$

Where:

- $q \cdot d$ = dot product of query & document vectors
- $|q|, |d|$ = length of vectors

Mini example (intuition)
Query “cloud computing” gives high weights to *cloud*, *computing*.

- D1 has both terms → high cosine.
- D2 has *cloud* but not *computing* → medium cosine.
- D3 has *cloud* only → lower cosine.
  Ranked: D1 > D2 > D3.

Strengths

- Produces a ranking (much better user experience).
- Handles partial matches naturally.
- Transparent and easy to tune via term weighting (TF–IDF, normalization).

Limitations

- Assumes term independence (no semantics/synonyms).
- Needs careful weighting/normalization; high-dimensional vectors can be costly at scale.
- Pure VSM lacks probabilistic interpretation.

When to choose

- General web/library search; when you need ranked results without heavy probabilistic modeling; good default baseline.

Advantages

- Provides **ranking** (better user experience).
- Handles partial matches.
- Flexible with weighting schemes (TF–IDF).

Disadvantages

- Assumes independence of terms.
- Needs normalization and weighting.
- High-dimensional vectors (costly for very large collections).

3. Probabilistic Model

The probabilistic model estimates the probability of a document being relevant to a query. It ranks documents according to their likelihood of relevance, often using models like BM25. This approach improves accuracy but is more complex to implement and tune.

3. Probabilistic Model (Basic idea)

Probability that document d is relevant to query q:

$$P(R|d,q) = \frac{P(d|R) \cdot P(R)}{P(d)}$$

How it behaves

- Rewards multiple occurrences of a query term but with diminishing returns.
- Penalizes overly long documents (unless terms are truly dense).
- Naturally ranks results; often outperforms plain TF–IDF cosine.

Strengths

- Strong ranking accuracy in practice (BM25 is a standard).
- Can incorporate relevance feedback and priors.
- Accounts for document length and robust term frequency handling.

Limitations

- Requires parameter tuning and estimating statistics.
- Classical independence assumptions are simplistic.
- More complex than Boolean/VSM to explain and implement from scratch.

When to choose

- You care most about ranking quality; medium/large collections; can compute collection stats (df, avgdl). Typical choice for production retrieval.

Advantages

- Produces very accurate rankings.
- Accounts for document length.
- Can use feedback and probability estimates.

Disadvantages

- More complex than Boolean/VSM.
- Needs parameter tuning.
- Independence assumption is unrealistic.

Scatter Storage or Hash Addressing (in Information Retrieval)

Definition:

Scatter storage, also called **hash addressing**, is a method used to store and quickly retrieve index terms in an **inverted index file**. Instead of keeping terms in sequential order, a **hash function** is applied to each term, which computes an address (location) in memory/disk where the postings of that term are stored.

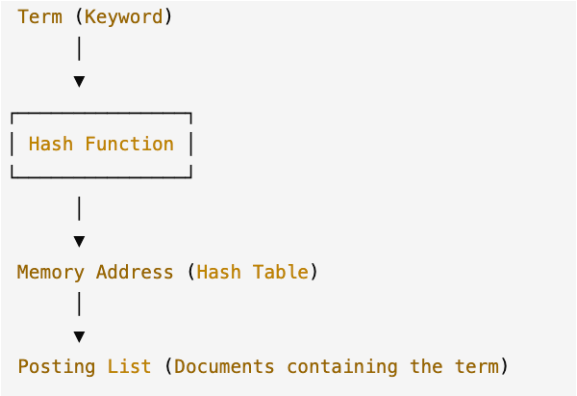
How it works:

1. Each **term** (keyword) is given to a **hash function**.
2. The hash function generates a **numeric address**.
3. That address is used to store or retrieve the **posting list** (list of documents containing that term).
4. When a query arrives, the same hash function is applied → direct access to the term's postings.

Example:

Suppose we have 3 terms in documents:

- "Cloud", "Data", "Security"
- Apply a hash function $h(x)$:
  - $h(\text{Cloud}) = 5$ → stored at address 5
  - $h(\text{Data}) = 12$ → stored at address 12
  - $h(\text{Security}) = 20$ → stored at address 20



Now, if query is "Data", system directly jumps to address **12** to get the posting list (instead of searching sequentially).

Advantages:

- Very **fast access** (constant time O(1)).
- Saves search time compared to sequential/serial search.

Disadvantages:

- **Collisions** may occur (two terms may hash to the same address).
- Needs good hash function and collision-handling methods.
- Memory usage may not be efficient if hash table is sparse.

Use in IR:

Scatter storage (hash addressing) is commonly used in **dictionary lookup** of inverted index files, making **searching faster** by directly jumping to the address of a term instead of scanning all entries.