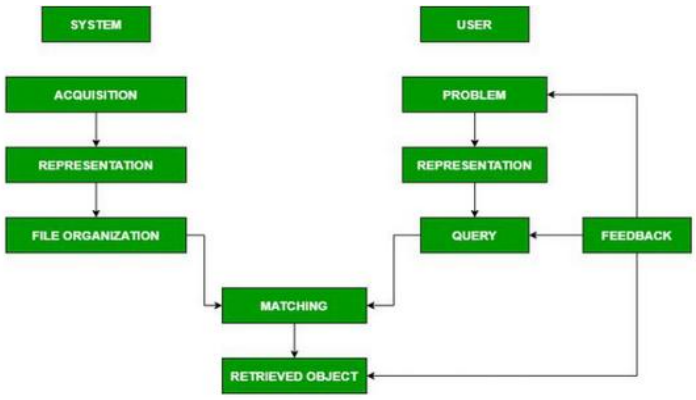
IR UNIT 1 – PYQ Answers

➢ **OCT 2022**

Q1

a) Draw the software architecture of the IR System and explain its components in detail. [7]



1. Acquisition (System Side)

- Collects documents from various sources (web pages, books, databases).
- Converts them into a format suitable for processing.
- Example: Crawling web pages or uploading documents into the repository.

2. Representation (System Side)

- Prepares documents for indexing by extracting significant terms.
- Includes tokenization, stop-word removal, and stemming.
- Creates a structured form of the document for efficient retrieval.

3. File Organization

- Stores processed documents and index structures efficiently.
- Uses inverted index, hash tables, or other indexing methods.
- Ensures quick access during query matching.

4. Problem (User Side)

- User identifies an **information need**.
- This is the mental stage before forming an actual query.
- Example: Needing “latest research papers on blockchain”.

5. Representation (User Side)

- User converts the problem into a formal query.
- Keywords or natural language statements are prepared.
- Example: Typing “blockchain recent papers” in the search box.

6. Query

- Formal input given to the IR system for processing.
- May include operators (AND, OR, NOT) or search parameters.
- System processes this query to match with the stored index.

7. Matching

- Compares the processed query with the stored document index.
- Uses models such as Boolean, Vector Space, or Probabilistic models.
- Identifies documents relevant to the query terms.

8. Retrieved Object

- The output documents fetched from the index after matching.
- Results are often ranked based on relevance (e.g., TF-IDF score).
- Displayed to the user with titles, snippets, and links.

9. Feedback

- User may provide feedback on retrieved results (relevance feedback).
- System uses this to refine the search and improve ranking.
- Example: Clicking “Not relevant” on certain search results.

(b) Describe the following IR Models: [8]

i) Boolean Model

The Boolean Model is a simple Information Retrieval model where documents are represented as a set of terms, and retrieval is based on exact matching using Boolean logic operators.

Key Features:

1. **Representation:**
    - Documents and queries are represented as a collection of keywords.
    - Each term is either present (1) or absent (0) in a document.
2. **Operators Used:**
    - **AND:** Retrieves documents containing *all* query terms.
    - **OR:** Retrieves documents containing *any* query term.
    - **NOT:** Excludes documents containing specific terms.
3. **Matching Process:**
    - Exact match is required between query and document terms using Boolean expressions.
    - Example: Query (AI AND Blockchain) NOT Bitcoin.
4. **Advantages:**
    - Simple to understand and implement.
    - Precise control over retrieved results.
5. **Limitations:**
    - No ranking of documents — results are either relevant or not relevant.
    - Requires users to know Boolean query syntax.

ii) TF-IDF Model

TF-IDF (Term Frequency–Inverse Document Frequency) is a statistical weighting scheme used to rank the importance of terms in documents relative to the query and the document collection.

Key Concepts:

1. **Term Frequency (TF):**
   - Measures how often a term appears in a document.
   - Formula:

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

2. **Inverse Document Frequency (IDF):**
   - Measures how unique or rare a term is across all documents.
   - Formula:

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

3. **TF-IDF Weight:**
   - Product of TF and IDF.
   - Formula:

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

4. **Working:**
   - Common terms (e.g., "the", "is") get low weights due to low IDF.
   - Rare but important terms get higher weights, improving ranking accuracy.
5. **Advantages:**
   - Provides ranking based on relevance score.
   - Efficient for keyword-based retrieval.
6. **Limitations:**
   - Ignores semantic meaning of words.
   - Assumes terms are independent.

Q2

(a) What are stop words and suggest solution for stop words removal with suitable examples. [7]

Stop words are commonly used words in a language that carry very little meaning in information retrieval.

Examples: *is, the, a, an, of, in, and, to*.
These words occur frequently in text and queries, but they usually do not help in distinguishing relevant documents from irrelevant ones.

Problems Caused by Stop Words

1. **Increased Index Size:**
    - Adds unnecessary entries to the inverted index.
2. **Lower Retrieval Efficiency:**
    - Wastes processing on irrelevant matches.
3. **Query Noise:**
    - Common words may overshadow important keywords.

Solutions for Stop Word Removal

a) Stop Word List Method

- Maintain a predefined list of stop words.
- During preprocessing, filter out any term that matches the list.
- **Example:**
  Input: "The cat is on the mat"
  Stop word removal: "cat mat"

b) Frequency Threshold Method

- Calculate term frequency in the corpus.
- Remove words that appear in more than a certain percentage of documents.
- **Example:** Remove words that occur in more than 80% of all documents.

c) Using NLP Libraries

- Use built-in stop word removal functions from libraries like **NLTK**, **spaCy**, or **Lucene**.
- Advantage: Language-specific and easily customizable.

Example of Stop Word Removal

Before: "The quick brown fox jumps over the lazy dog"
After: "quick brown fox jumps lazy dog"

(b) Explain Simple Tokenizing with example. [8]

Simple Tokenizing is the process of breaking a stream of text into smaller units called **tokens** (usually words, terms, or symbols) for further processing in an Information Retrieval system.

Purpose:

- Converts unstructured text into a list of meaningful elements.
- Forms the basis for indexing and later retrieval.

Process Steps:

1. **Input Text:** Read the raw text data.
2. **Splitting:** Break text into tokens based on delimiters (spaces, punctuation, special characters).
3. **Filtering:** Remove empty tokens or unwanted symbols.
4. **Output:** A sequence/list of tokens.

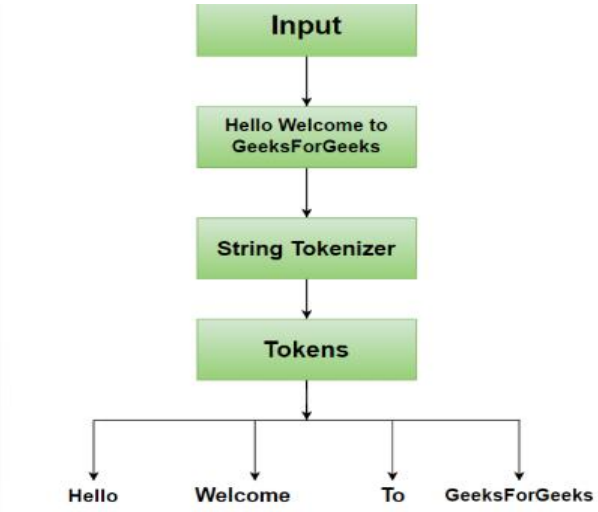
Example:
Input: "Information Retrieval is important!"
Tokens: ["Information", "Retrieval", "is", "important"]

Advantages:

- Simple and fast to implement.
- Works well for languages with clear word boundaries (like English).

Limitations:

- Cannot handle complex cases like hyphenated words, contractions, or languages without spaces (e.g., Chinese).
- May need further preprocessing like stop word removal or stemming.



➢ **2024**

Q1)

a) Draw the software architecture of the IR System and explain its components in detail.

ALREADY DONE !

b) Describe the following IR Models:

i) Boolean Model

ii) TF-IDF

ALREADY DONE

Q2)

a) What are stop words and suggest solution for stop words removal with suitable examples.

ALREADY DONE

b) Describe the following IR Models: [8]

i) Vector Model
ii) Latent Semantic Indexing Model

i) Vector Space Model (VSM)

The Vector Space Model (VSM) is an algebraic model that represents documents and queries as vectors in a high-dimensional space, where each dimension corresponds to a unique term in the corpus.

Key Concepts:

1. **Term-Document Matrix:**
   - Documents and queries are represented as vectors of term weights.
   - Example: If the vocabulary is {quick, brown, fox}, then:
     - Document: "The quick brown fox" → [1, 1, 1]
     - Query: "quick fox" → [1, 0, 1]

2. **Term Weighting (TF-IDF):**
   - **Term Frequency (TF):** Measures how often a term appears in a document.
   - **Inverse Document Frequency (IDF):** Penalizes terms that appear too frequently across documents.

- **TF-IDF = TF × IDF**, giving higher weight to rare but important terms.

3. **Similarity Measure (Cosine Similarity):**
   - Computes the angle between query and document vectors.
   - **Formula:**

$$\text{Similarity}(Q, D) = \frac{Q \cdot D}{|Q| \times |D|}$$

   - Ranges from 0 (no similarity) to 1 (exact match).

Advantages:

- Simple and intuitive.
- Handles partial matching (unlike Boolean models).
- Works well for ranking documents.

Limitations:

- Assumes term independence (ignores semantic relationships).
- Suffers from high dimensionality (sparse vectors).

Example:

- **Query**: "information retrieval"
- **Documents:**
  - D1: "information theory and retrieval models"
  - D2: "data retrieval in databases"
- VSM ranks D1 higher due to higher term overlap.

ii) Latent Semantic Indexing (LSI) Model

Latent Semantic Indexing (LSI), also called **Latent Semantic Analysis (LSA)**, is an advanced IR model that uses **Singular Value Decomposition (SVD)** to capture hidden semantic relationships between terms and documents.

Key Concepts:

1. **Term-Document Matrix Reduction:**

- Constructs a term-document matrix (like VSM).
- Applies **SVD** to decompose it into three matrices:

$$A = U\Sigma V^T$$

  - **U:** Term-concept matrix
  - **Σ:** Singular values (importance of concepts)
  - **V:** Document-concept matrix

2. **Dimensionality Reduction:**
   - Retains only the top **k** singular values (reduces noise).
   - Projects documents and queries into a **lower-dimensional "concept space."**
3. **Semantic Matching:**
   - Finds documents even if they don't share exact terms but are conceptually related.
   - Example:
     - "Car" and "Automobile" may map to the same latent concept.

Advantages:

- Handles **synonymy** (different words with similar meanings).
- Reduces noise and improves relevance.
- Better than VSM for **semantic search**.

Limitations:

- Computationally expensive (SVD on large matrices).
- Requires tuning of **k** (number of concepts).

Example:

- **Query:** "vehicle"
- **Documents:**
  - D1: "car engine specifications"
  - D2: "truck manufacturing process"
- **LSI matches both D1 and D2** because "car" and "truck" are semantically related to "vehicle."