## **1. Overall Design of the System**

The project involves the creation of a web-based search engine that crawls web pages, indexes their content, and supports a ranked search functionality based on user queries. The system consists of the following components:

1. **Crawler (Spider)**: Fetches web pages recursively using BFS, extracts keywords, and stores the parent-child relationships in the database.
2. **Indexer**: Processes the content of crawled pages to build an inverted index for efficient search.
3. **Search Engine**: Retrieves and ranks documents based on user queries using the vector space model (TF-IDF and cosine similarity).
4. **Web Interface**: Provides a user-friendly UI for submitting queries and viewing ranked results.

## **2. File Structures in the Index Database**

### **Database Schema**

The system uses an SQLite database (**scraper.db**) for storing crawled data and indexes. The schema is defined as follows:

1. **links**
   * **Purpose**: Stores metadata for each crawled page.
   * **Columns**:
     + **id**:Unique identifier for the page.
     + **title**: The title of the page.
     + **stem\_title**:The stemmed version of the title.
     + **url**: the URL of the page.
     + **last\_mod\_date**: The last modification date of the page.
     + **size**: The size of the page content in bytes.
     + **parent\_links**: A semicolon-separated list of parent URLs.
2. **keywords\_freq**
   * **Purpose**: Stores keywords and their frequency for each page.
   * **Columns**:
     + **freq\_id**: Primary key.
     + **keyword**: Indexed keyword.
     + **parent\_group**: Reference to the page ID.
     + **frequency**: Frequency of the keyword in the page.
3. **child\_links**
   * **Purpose**: Stores child links of each page.
   * **Columns**:
     + **id**: Primary key.
     + **child\_id**: Reference to the child page ID.
     + **parent\_group**: Reference to the parent page ID.
     + **url**: Absolute URL of the child link.
4. **body\_positions**
   * **Purpose**: Stores positions of keywords in the page body.
   * **Columns**:
     + **word**: Indexed word.
     + **parent\_group**: Reference to the page ID.
     + **positions**: Comma-separated list of positions.
5. **title\_positions**
   * **Purpose**: Stores positions of keywords in the page title.
   * **Columns**:
     + **word**: Indexed word.
     + **parent\_group**: Reference to the page ID.
     + **positions**: Comma-separated list of positions.

## **3. Algorithms Used**

### **3.1 Crawler (Spider)**

The crawler fetches web pages recursively using a breadth-first strategy:

* **URL Validation**: Avoids cyclic links and checks modification dates before fetching a page.
* **Hyperlink Extraction**: Extracts and fixes relative links to absolute URLs.
* **Parent-Child Relationship**: Maintains parent-child relationships for links.

### **3.2 Indexing**

* **Stop Word Removal**: Uses a predefined list of stop words to remove common, irrelevant words.
* **Stemming**: Applies Porter’s stemming algorithm to normalize words.
* **Inverted Index**:
  + One inverted index for page bodies (**body\_positions**).
  + Another inverted index for page titles (**title\_positions**).
* **Position Tracking**: Tracks positions of keywords for phrase search.

### **3.3 Ranking (Search Engine)**

* **TF-IDF Calculation**:
  + **Term Frequency (TF)**:TF = frequency of term in document / total terms in document.
  + **Inverse Document Frequency (IDF)**: IDF = log(total\_documents / documents\_containing\_term).
  + **TF-IDF**: TF-IDF = TF \* IDF.
* **Cosine Similarity**: Measures similarity between the query vector and document vector.
* **Title Match Boosting**: Increases the rank of documents where the query matches the title.

## **4. Installation Procedure**

### **Prerequisites**

* **Python**: Version 3.10 or higher.
* **Flask**: Install Flask using **pip install flask.**
* **NLTK**: Install NLTK and its dependencies using **pip install nltk.**
* **SQLite**: Pre-installed with Python.

### **Setup Instructions**

1. Clone the repository and navigate to the project directory.
2. Run the database setup script (**main.py**) to initialize the database and index pages.
3. Start the Flask server: **python Flask(modify).py**
4. Access the web interface at **http://127.0.0.1:5000/index.html.**

## **5. Web Interface**

### **Functionality**

* **Search Bar**: Allows users to input queries (with or without phrases in quotes).
* **Spelling Correction**: Automatically suggests corrections for misspelled queries using **TextBlob**.
* **Result Ranking**: Displays up to 50 ranked results with the following details:
  1. **Score**: Calculated using TF-IDF and cosine similarity.
  2. **Page Title**: Hyperlinked to the URL.
  3. **URL**: Hyperlinked to the actual page.
  4. **Last Modified Date**: Shows the date when the page was last modified.
  5. **Size**: Displays the size of the page in bytes.
  6. **Keywords**: Lists up to 5 most frequent keywords.
  7. **Parent Links**: Lists up to 3 parent links.
  8. **Child Links**: Lists up to 3 child links.

### **User Interface**

* **Clarity**: Simple, clean design with a responsive layout.
* **Features**:
  + Spelling correction toggle.
  + Results formatted with keyword highlights, parent/child links, and cosine similarity details.

## **6. Testing and Results**

### **Test Cases**

1. **Spider Functionality**:  
   * Starting URL: https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm.
   * Indexed Pages: 300.
   * Output: spider\_result.txt, containing page titles, URLs, keywords, and links.
2. **Search Engine**:  
   * Query: "hong kong" universities.
   * Results: Up to 50 ranked pages with detailed TF-IDF and cosine similarity calculations.

### **Screenshots**

* Search Interface: (Include a screenshot of the search bar and results.)
* Ranked Results: (Include a screenshot showing result details like score, title, URL, and keywords.)

## **7. Strengths and Weaknesses**

### **Strengths**

* Efficient crawling with BFS and cyclic link handling.
* Accurate keyword indexing with position tracking for phrase search.
* Clear and user-friendly web interface.
* Robust ranking mechanism using TF-IDF and cosine similarity.

### **Weaknesses**

* Crawling speed could be optimized further.
* Limited support for advanced queries like Boolean operators.
* The system relies on the correctness of the stopword list.

## **8. Future Improvements**

* Implement a relevance feedback feature (e.g., "Get Similar Pages").
* Add support for query history and search result merging.
* Integrate a PageRank algorithm to improve ranking.
* Develop a mobile-friendly interface using AJAX and DHTML.
* Optimize indexing and search speed with caching techniques.

## **9. Contribution**

William Chen: 40% (mainly responsible for database and scraper)

Po Wa Ho: 40% (mainly responsible for html and flask.py)

Fung Ming Sze: 20% (mainly responsible for bonus part)