README — Simplified Search Engine Project

Overview

This project implements a **simplified search engine** as described in **Section 23.6** of the textbook *Advanced Algorithm Design and Implementation (zyBooks)*. It supports:

- Indexing and searching of small website content
- Trie-based inverted index
- Stop word removal
- Single-word and multi-word queries
- Ranked result retrieval (optional strict or partial match)
- Boundary condition handling

1. Data Structures Used

✅ Trie (Prefix Tree)

We use a **standard trie** to store and look up words extracted from web pages. The trie supports fast, exact string matching and links each leaf node to an **external occurrence list**, following the design in Section **23.6.4**:

- Each internal node stores a character.
- Each complete word leads to a leaf node.
- Each leaf node maps to an index in the **occurrence list** array.

✅ Occurrence List (Inverted File)

Each unique word has an external list (array of URLs) where it appears. This is implemented as a `List[List[str]]`, sorted by URL for efficient set intersection.

✅ Frequency Table

A `defaultdict(dict)` maps each `(word → URL)` pair to its frequency (number of appearances on the page). This is used for ranking.

2. Algorithms Implemented

Index Building (`SearchEngine.build_index`)

1. For each input URL:

    - Extracts `<p>` text using `requests` and `BeautifulSoup`.
    - Cleans the text (punctuation removal, stopword filtering).
    - Updates:
        - The word’s occurrence list.
        - Frequency table.
        - Inserts the word into the Trie.

Caching is enabled to reduce redundant web downloads.

Single Word Search

- Looks up the word in the Trie.
- Retrieves the occurrence list index.
- Returns a sorted list of URLs by frequency.

Multi-word Strict AND Search (`strict_ranked_search`)

- Computes the **intersection** of occurrence lists for all input words.
- Filters out pages that do **not contain all** words.
- Ranks pages by:
    - Number of matched keywords.
    - Total word frequencies.
    - (Tie-breaker) Alphabetical order of URLs.

Multi-word OR Ranked Search (`ranked_search`) *(optional feature)*

- Includes any page with **at least one** matched word.
- Scores and ranks based on:
    - Number of unique matched words.
    - Total frequency.
    - Alphabetical tie-breaker.

Text Preprocessing

- Removes punctuation with regex.
- Converts words to lowercase.
- Eliminates stopwords (custom stopword list).

3. Project Structure

```
.
├── main_single.py         # CLI for single keyword search
├── main_multi.py          # CLI for multi-keyword AND search
├── engine.py              # SearchEngine logic (trie + indexing + ranking)
├── trie.py                # Trie and TrieNode classes
├── rank.py                # RankedPage data structure and scoring logic
├── parseLinks.py          # Web scraping and text processing logic
├── stopword_utils.py       # Stopword checker
├── inputLinks.txt          # Input file with target URLs
├── cached_pages/           # Local HTML cache for each page
├── Output.pdf              # Run result screenshots (provided separately)
└── README.md               # This file
```

4. Matching Textbook Requirements

Feature	Implemented?	Notes
Trie indexing	✅	Used standard trie (Section 23.6.1)
External occurrence list	✅	Each leaf node maps to URL list
Stop word removal	✅	via `stopword_utils`
Exact keyword matching	✅	Case-insensitive exact matching
Intersection of results	✅	Set intersection (23.6.4 logic)
Ranking by frequency	✅	As described in final paragraph
Alphabetical tie-breaking	✅	Stable and consistent sort
Boundary case handling	✅	Input validation included

5. Boundary Conditions Tested

See `Output.pdf`. We tested the following:

- Empty input
- Multi-word input in single-word mode
- Case insensitivity
- Stopword filtering
- Word repetition
- Nonexistent keywords
- Valid vs invalid combinations

6. How to Run

```
$ python3 main_single.py    # Single-word search
$ python3 main_multi.py     # Multi-keyword AND search
```

Make sure you have:

- Python 3.8+
- Installed packages: `requests`, `beautifulsoup4`

7. Performance Note

- Trie lookup time: **O(m)** for a word of length *m*
- Set intersection: **O(k * n)** in worst case (where *k* is number of keywords, *n* average URLs per word)
- Ranking uses a stable `sort()` on a list of custom objects

8. Acknowledgments

Project implemented under guidelines of **CPE 600 / CS 600** — *Advanced Algorithm Design and Implementation*.