Stage 4 — Ranking Precompute (Search Engine Project)

Date: 2025-11-25
Project: Custom Search Engine

Stage: 4 / Phase A — Productionize Single-Node

Overview

Stage 4 focuses on **precomputing ranking-related statistics** to make the query engine efficient and scalable. Specifically, it calculates:

1. **Document lengths (`dl`)** — number of terms per document.
2. **Average document length (`avgdl`)** — used in BM25 normalization.
3. **Inverse Document Frequency (`idf`)** — measures term importance across the corpus.

These outputs are **binary-serialized** for fast loading in the next stage (Stage 5 — Query Engine).

Inputs

- **Dataset:** `data/sample_tokens_final_clean.txt` (cleaned tokenized documents)
- **Lexicon:** Built in Stage 1 (`stage1_lexicon`) → maps terms to `term_id` and stores document frequency.
- **Forward Index:** Built in Stage 2 (`stage2_forward_index`) → stores per-document term lists and frequencies.

Process

1️⃣Loading Lexicon and Forward Index

- **Lexicon:** Provides `term_id` mapping for all tokens in the corpus.
- **Forward Index:** Provides per-document term frequencies and lengths.

2️⃣Compute Document Lengths (`dl`)

For each document in the forward index:

```
dl[doc_id] = total number of terms in document
```

This is used for BM25 normalization (`b` parameter).

3️⃣Compute Average Document Length (`avgdl`)

avgdl=∑i=0N−1dl[i]N\text{avgdl} = \frac{\sum_{i=0}^{N-1} dl[i]}{N}avgdl=N∑i=0N−1dl[i]

Where `N` is the total number of documents.

Result in project: `avgdl ≈ 318.471` for 5000 documents.

4️⃣Compute Inverse Document Frequency (`idf`)

BM25-style IDF is calculated as:

idf(t)=logN−df(t)+0.5df(t)+0.5idf(t) = \log \frac{N - df(t) + 0.5}{df(t) + 0.5}idf(t)=logdf(t)+0.5N−df(t)+0.5

Where:

- `df(t)` = number of documents containing term `t`
- `N` = total documents

This captures **term importance**: rarer terms get higher IDF.

Implementation Highlights

- Written in **C++**, integrated with existing project structure:
  - `include/stage4_ranking.h`
  - `src/stage4_ranking.cpp`
- Modular design: `Stage4Ranking` class accepts `ForwardIndex` and `Lexicon` objects.
- Binary serialization for **fast Stage 5 loading**:
  - `dl.bin` → vector of document lengths
  - `avgdl.bin` → single double value
  - `idf.bin` → map of term_id → IDF
- Files saved to: `output/ranking/`

Execution

In `main.cpp`, Stage 4 is executed after Stage 3:

```
Stage4Ranking stage4(fwd, lex);
stage4.compute();
stage4.save_binaries(output_folder + "\\ranking");
```

Console Output:

```
Stage 4: Ranking precompute done.
Total docs: 5000, avgdl: 318.471, total terms: 46763
Stage 4: Binaries saved to output\ranking
```

- Processed **5000 documents**

- **46763 unique terms**
- Computed **avgdl = 318.471**

Deliverables

- `output/ranking/dl.bin` — document lengths
- `output/ranking/avgdl.bin` — average document length
- `output/ranking/idf.bin` — term IDF
- Fully **ready for Stage 5 Query Engine**

Challenges & Solutions

Challenge	Solution
Handling large datasets efficiently	Used binary files for fast read/write
Mapping term → term_id consistently	Used `Lexicon` from Stage 1
Computing document frequency efficiently	Used per-document unique term sets to avoid double counting
Integrating with existing project structure	Designed `Stage4Ranking` class compatible with `ForwardIndex` and `Lexicon`

Next Steps

- **Stage 5 — Query Engine**
  - Load `dl.bin`, `avgdl.bin`, and `idf.bin`

    - Implement BM25 scoring
    - Top-K retrieval
    - Snippet generation
    - Serve queries via a REST API

Stage 4 ensures that **all ranking statistics are precomputed**, enabling the query engine to perform **fast, accurate searches** over 5000+ documents with minimal runtime computation.