Understanding Apache Lucene - more than full text search

Alexander Reelsen

Agenda

- Lucene Introduction
- The inverted index
- Other data structures: columnar storage, FSTs
- Inside of queries: Smarter skipping/pruning
- Memory Mapping & giving the OS hints
- Use-case: Storing & querying hotel room reservations
- Q & A

About me

- Doing search engines since 2010
- Elasticsearch User & Developer
- Lucene novice

Lucene Introduction

- Full text search
- Analysis (stemming, phonetic, ICU)
- Language support (nori, kuromoji, stempel, smartcn)
- Faceting & Grouping
- Highlighting
- Geospatial & number searches using BKD tree
- Suggest & Spellchecking

The inverted index

- Mapping: `term -> docId`
- Search: Looks up term in list, retrieves list of docIds
- Lists are easy to combine with AND/OR
- No relevance information
- Term frequencies, positions, offsets

IndexWriter

```
    IndexWriterConfig iwc = new IndexWriterConfig();
    iwc.setUseCompoundFile(false);

    try (Directory dir = FSDirectory.open(indexPath);IndexWriter writer = new IndexWriter(dir, iwc)) {
        Document doc = new Document();

        doc.add(new StringField("title", "Understanding Lucene", Field.Store.YES));
        doc.add(new TextField("title_text", "Understanding Lucene", Field.Store.YES));
        doc.add(new SortedDocValuesField("status", new BytesRef("ongoing")));

        doc.add(new NumericDocValuesField("length", 4299));

        double lat = 51.9615, lon = 7.6262;
        doc.add(new LatLonPoint("location", lat, lon));
        doc.add(new LatLonDocValuesField("location", lat, lon));

        double[] polyLats = { 51.9499, 51.9510, 51.9514, 51.9513, 51.9514, 51.9520, 51.9518, 51.9513, 51.9510,
51.9499 };
        double[] polyLons = { 7.6480, 7.6455, 7.6413, 7.6389, 7.6388, 7.6411, 7.6428, 7.6473, 7.6496, 7.6480 };
        Polygon polygon = new Polygon(polyLats, polyLons);
        for (Field f : LatLonShape.createIndexableFields("geo_shape", polygon)) {
            doc.add(f);
        }
        doc.add(LatLonShape.createDocValueField("geo_shape", polygon));

        writer.addDocument(doc);
        writer.commit();
    }
```

[finished]

```
Writing index...✅
```

IndexWriter

```
lsd -1l --size=short --date=relative --color=always /tmp/lucene/001-index-writer
```

[finished]

```
.rw-r--r-- alr wheel 157B 14 hours ago _0.fdm
.rw-r--r-- alr wheel 140B 14 hours ago _0.fdt
.rw-r--r-- alr wheel  64B 14 hours ago _0.fdx
.rw-r--r-- alr wheel 659B 14 hours ago _0.fnm
.rw-r--r-- alr wheel 260B 14 hours ago _0.kdd
.rw-r--r-- alr wheel  70B 14 hours ago _0.kdi
.rw-r--r-- alr wheel 216B 14 hours ago _0.kdm
.rw-r--r-- alr wheel  59B 14 hours ago _0.nvd
.rw-r--r-- alr wheel 103B 14 hours ago _0.nvm
.rw-r--r-- alr wheel 517B 14 hours ago _0.si
.rw-r--r-- alr wheel  79B 14 hours ago _0_Lucene103_0.doc
.rw-r--r-- alr wheel  81B 14 hours ago _0_Lucene103_0.pos
.rw-r--r-- alr wheel 112B 14 hours ago _0_Lucene103_0.psm
.rw-r--r-- alr wheel 134B 14 hours ago _0_Lucene103_0.tim
.rw-r--r-- alr wheel  92B 14 hours ago _0_Lucene103_0.tip
.rw-r--r-- alr wheel 233B 14 hours ago _0_Lucene103_0.tmd
.rw-r--r-- alr wheel 286B 14 hours ago _0_Lucene90_0.dvd
.rw-r--r-- alr wheel 491B 14 hours ago _0_Lucene90_0.dvm
.rw-r--r-- alr wheel 155B 14 hours ago segments_1
.rw-r--r-- alr wheel   0B 14 hours ago write.lock
```

Querying data

```
void main() throws IOException {
    System.setErr(new PrintStream(OutputStream.nullOutputStream()));
    Path indexPath = Paths.get("/tmp/lucene/001-index-writer/");

    try (Directory dir = FSDirectory.open(indexPath); DirectoryReader reader = StandardDirectoryReader.open
(dir)) {
        var searcher = new IndexSearcher(reader);

        TopDocs topDocs = searcher.search(new TermQuery(new Term("title", "Lucene")), 10);
        System.out.printf("Found %s hits%n", topDocs.totalHits.value());

        topDocs = searcher.search(new TermQuery(new Term("title", "Understanding Lucene")), 10);
        System.out.printf("Found %s hits%n", topDocs.totalHits.value());

        topDocs = searcher.search(new TermQuery(new Term("title_text", "lucene")), 10);
        System.out.printf("Found %s hits%n", topDocs.totalHits.value());
    }
}
```

[finished with error]

Analysis

```java
void main() throws Exception {
    String german  = "Die Füchse laufen schnell und die Hunde schlafen";
    header("German", german);
    try (var a = new GermanAnalyzer()) { tokens(a, german); }

    String english = "the foxes are quickly running and the dogs are sleeping";
    header("Stop words", english);
    try (var a = new StopAnalyzer(EnglishAnalyzer.ENGLISH_STOP_WORDS_SET)) { tokens(a, english); }

    header("English stemmer", english);
    try (var a = new EnglishAnalyzer()) { tokens(a, english); }

    String sounds  = "Meier Meyer Maier Meyr";
    header("Phonetic (Cologne)", sounds);
    try (var a = phoneticAnalyzer()) { tokens(a, sounds); }
}

void header(String name, String input) {
    System.out.printf("%n=== %s ===%n", name);
    System.out.printf("input:  \"%s\"%n", input);
    System.out.print("tokens: ");
}
```

[finished]

Analysis

```
=== German ===
input:  "Die Füchse laufen schnell und die Hunde schlafen"
tokens: [fuchs, lauf, schnell, hund, schlaf]

=== Stop words ===
input:  "the foxes are quickly running and the dogs are
sleeping"
tokens: [foxes, quickly, running, dogs, sleeping]

=== English stemmer ===
input:  "the foxes are quickly running and the dogs are
sleeping"
tokens: [fox, quickli, run, dog, sleep]

=== Phonetic (Cologne) ===
input:  "Meier Meyer Maier Meyr"
tokens: [67, 67, 67, 67]
```

Facets

```
FacetsConfig config = new FacetsConfig();
try (Directory dir = FSDirectory.open(indexPath);
     IndexWriter writer = new IndexWriter(dir, new IndexWriterConfig())) {

    addDoc(writer, config, "Lucene in Action",                     "Technology");
    addDoc(writer, config, "Clean Code",                           "Technology");
    addDoc(writer, config, "The Pragmatic Programmer",             "Technology");
    addDoc(writer, config, "Elasticsearch: The Definitive Guide", "Technology");
    addDoc(writer, config, "A Brief History of Time",              "Science");
    addDoc(writer, config, "The Selfish Gene",                     "Science");
    addDoc(writer, config, "Cosmos",                               "Science");
    addDoc(writer, config, "Sapiens",                              "History");
    addDoc(writer, config, "Guns, Germs, and Steel",               "History");
    addDoc(writer, config, "1984",                                 "Fiction");
    addDoc(writer, config, "The Hitchhiker's Guide to the Galaxy","Fiction");
    addDoc(writer, config, "Dune",                                 "Fiction");
    writer.commit();
}
```

[finished]

```
Indexing documents...✅
```

Facets

```
FacetsConfig config = new FacetsConfig();
try (Directory dir = FSDirectory.open(indexPath);
     DirectoryReader reader = DirectoryReader.open(dir)) {
    IndexSearcher searcher = new IndexSearcher(reader);
    DefaultSortedSetDocValuesReaderState state = new DefaultSortedSetDocValuesReaderState(reader, config);
    FacetsCollectorManager fcm = new FacetsCollectorManager();
    FacetsCollectorManager.FacetsResult fr =
            FacetsCollectorManager.search(searcher, new MatchAllDocsQuery(), 10, fcm);

    Facets facets = new SortedSetDocValuesFacetCounts(state, fr.facetsCollector());
    FacetResult result = facets.getTopChildren(10, "category");

    System.out.println("Category counts (all documents):");
    for (LabelAndValue lv : result.labelValues) {
        System.out.printf("  %-12s  %d%n", lv.label, lv.value.intValue());
    }
```

```
[finished with error]
```

Highlighting

```
try (Directory dir = FSDirectory.open(indexPath);
     var analyzer = new StandardAnalyzer();
     IndexWriter writer = new IndexWriter(dir, new IndexWriterConfig(analyzer))) {

    addDoc(writer,
            "Lucene in Action",
            "Lucene is a high-performance, full-featured text search engine library. " +
            "It supports full-text search with relevance scoring, faceting, and highlighting. " +
            "You can index millions of documents and search them in milliseconds.");

    addDoc(writer,
            "Introduction to Algorithms",
            "This book covers a broad range of algorithms in depth. " +
            "Binary search and sorting algorithms are explained with clear pseudocode " +
            "and rigorous proofs of correctness.");
```

[finished]

```
Indexing documents...✅
```

Highlighting

```java
try (Directory dir = FSDirectory.open(indexPath);
     DirectoryReader reader = DirectoryReader.open(dir);
     var analyzer = new StandardAnalyzer()) {

    IndexSearcher searcher = new IndexSearcher(reader);
    Query query = new TermQuery(new Term("body", "search"));

    TopDocs topDocs = searcher.search(query, 10);
    System.out.printf("Found %d document(s) matching \"%s\":%n%n",
            topDocs.totalHits.value(), "search");

    UnifiedHighlighter highlighter = UnifiedHighlighter.builder(searcher, analyzer).build();
    String[] snippets = highlighter.highlight("body", query, topDocs);

    for (int i = 0; i < topDocs.scoreDocs.length; i++) {
        var doc = reader.storedFields().document(topDocs.scoreDocs[i].doc);
        System.out.printf("  %s, score %s%n", doc.get("title"), topDocs.scoreDocs[i].score);
        System.out.printf("  %s%n%n", snippets[i]);
    }
}
```

```
[finished with error]
```

Highlighting

Querying

- Parsing
- Finding
- Scoring
- Sorting

Parsing

- Query language is important
- SQL is the most well known one in the world, but no meaning of relevance
- Lucene has a more than one, `classic` is the most well known one
- Terms, Ranges, Wildcards, Fuzzy, Proximity, Boosting, AND/OR/NOT
- Syntax creates tree of queries
- `title:"The Right Way" AND text:go`
- `/[mb]oat/`
- `"jakarta apache"~10`
- `mod_date:[20020101 TO 20030101]`
- `"jakarta apache"^4 "Apache Lucene"`
- `(jakarta OR apache) AND website`

See https://lucene.apache.org/core/10_3_2/queryparser/org/apache/lucene/queryparser/classic/package-summary.html

Scoring

- This is where the difference to SQL is at
- A document is not just within our outside of a result set
- A document is scored against the query
- Core question: What is relevance?

See https://www.elastic.co/blog/practical-bm25-part-1-how-shards-affect-relevance-scoring-in-elasticsearch

https://www.elastic.co/blog/practical-bm25-part-2-the-bm25-algorithm-and-its-variables

https://www.elastic.co/blog/practical-bm25-part-3-considerations-for-picking-b-and-k1-in-elasticsearch

Scoring via BM 25

- Term frequency, ensuring a term frequency saturation
- Inverse document frequency
- Field length vs. average field length

Scoring example

```
void main() throws Exception {
    System.setErr(new PrintStream(OutputStream.nullOutputStream()));
    Path indexPath = Paths.get("/tmp/lucene/002-scoring-example/");
    FileUtils.deleteDirectory(indexPath.toFile());
    System.out.print("Writing index...");

    IndexWriterConfig iwc = new IndexWriterConfig();

    try (Directory dir = FSDirectory.open(indexPath);IndexWriter writer = new IndexWriter(dir, iwc)) {
        writer.addDocument(doc("Elasticsearch - the definitive guide", "en"));
        writer.addDocument(doc("Elasticsearch in Action", "en"));
        writer.addDocument(doc("Elastic Stack 8.x Cookbook", "en"));
        writer.addDocument(doc("Elasticsearch: Ein praktischer Einstieg", "de"));
        writer.addDocument(doc("Designing data-intensive Applications", "en"));
        writer.commit();
    }

    System.out.println("✅");
}
```

[finished]

```
Writing index...✅
```

Scoring query

```
void main() throws Exception {
    System.setErr(new PrintStream(OutputStream.nullOutputStream()));
    Path indexPath = Paths.get("/tmp/lucene/002-scoring-example/");

    try (Directory dir = FSDirectory.open(indexPath); DirectoryReader reader = StandardDirectoryReader.open
(dir)) {
        var searcher = new IndexSearcher(reader);

        search("elasticsearch", searcher);

        search("elasticsearch lang:de^2", searcher);
    }
}

void search(String q, IndexSearcher searcher) throws Exception {
    var parser = new QueryParser("title", new StandardAnalyzer());
    Query query = parser.parse(q);

    TopDocs docs = searcher.search(query, 10);
    System.out.printf("___________________________________________________%n");
    System.out.printf("Found %s hits%n", docs.totalHits.value());

    StoredFields storedFields = searcher.storedFields();
    for (ScoreDoc scoreDoc : docs.scoreDocs) {
        Document doc = storedFields.document(scoreDoc.doc);
        System.out.printf("%-40s%f%n", doc.get("title"), scoreDoc.score);
    }
```

```
[finished with error]
```

Scoring query

Persistence & Disk storage

- Data is immutable & append-only, never gets overwritten
- Result: Lock free reading, multithreaded
- Writes create segments, mini inverted indices
- Segments can contain different files

Codecs

- A `Codec` defines how data is persisted to the disk
- Each data structure gets its own format
- Allows for BWC - and for BWC simplification
- Configured with the `IndexWriter`
- Groups all the formats together: `TermVectorsFormat`, `FieldInfosFormat`, `SegmentInfoFormat`, `LiveDocsFormat`, `CompoundFormat`, `NormsFormat`, `PostingsFormat`, `DocValuesFormat`, `KnnVectorsFormat`, `StoredFieldsFormat`

TextCodec

```java
IndexWriterConfig iwc = new IndexWriterConfig();
iwc.setUseCompoundFile(false);
iwc.setCodec(new SimpleTextCodec());

try (Directory dir = FSDirectory.open(indexPath);IndexWriter writer = new IndexWriter(dir, iwc)) {
    writer.addDocument(doc("Elasticsearch - the definitive guide", "en", 721));
    writer.addDocument(doc("Elasticsearch in Action", "en", 475));
    writer.addDocument(doc("Elastic Stack 8.x Cookbook", "en", 688));
    writer.addDocument(doc("Elasticsearch: Ein praktischer Einstieg", "de", 262));
    writer.addDocument(doc("Designing data-intensive Applications", "en", 614));
    writer.commit();
}
```

[finished]

```
Writing index...✅
```

TextCodec

```
lsd -1l --size=short --date=relative --color=always /tmp/lucene/003-codec-example/
```

[finished]

```
.rw-r--r-- alr wheel  253B 14 hours ago _0.dat
.rw-r--r-- alr wheel   81B 14 hours ago _0.dii
.rw-r--r-- alr wheel  496B 14 hours ago _0.dim
.rw-r--r-- alr wheel 1015B 14 hours ago _0.fld
.rw-r--r-- alr wheel  1.2K 14 hours ago _0.inf
.rw-r--r-- alr wheel  142B 14 hours ago _0.len
.rw-r--r-- alr wheel  1.1K 14 hours ago _0.pst
.rw-r--r-- alr wheel  739B 14 hours ago _0.si
.rw-r--r-- alr wheel  156B 14 hours ago segments_1
.rw-r--r-- alr wheel    0B 14 hours ago write.lock
```

TextCodec

- `write.lock`: Lock file to prevent concurrent writing
- `segments_1`: commit point for live segments
- `_0.si`: Segment info: metadata like name, document count and codec
- `_0.fld`: Field info: names, types, index options, and attributes
- `_0.inf`: Term dictionary and postings metadata
- `_0.pst`: Postings (inverted index): document IDs, term frequencies, and positions for each term
- `_0.len`: Norms (field length)
- `_0.dii`: Point values index; contains the index into the `.dim`
- `_0.dim`: Point values (BKD tree) data
- `_0.dat`: Stored fields data, original data would be lost otherwise

Codec - Compound files

- Too many little files hurt performance
- Small segments

Codec - Compound files

```java
IndexWriterConfig iwc = new IndexWriterConfig();
iwc.setUseCompoundFile(true);
iwc.setCodec(new SimpleTextCodec());

try (Directory dir = FSDirectory.open(indexPath);IndexWriter writer = new IndexWriter(dir, iwc)) {
    writer.addDocument(doc("Elasticsearch - the definitive guide", "en", 721));
    writer.addDocument(doc("Elasticsearch in Action", "en", 475));
    writer.addDocument(doc("Elastic Stack 8.x Cookbook", "en", 688));
    writer.addDocument(doc("Elasticsearch: Ein praktischer Einstieg", "de", 262));
    writer.addDocument(doc("Designing data-intensive Applications", "en", 614));
    writer.commit();
    writer.maybeMerge();
}
```

[finished]

```
Writing index...✅
```

Codec - Compound files

```
lsd -1l --size=short --date=relative --color=always /tmp/lucene/005-codec-example/
```

[finished]

```
.rw-r--r-- alr wheel 4.8K 14 hours ago _0.scf
.rw-r--r-- alr wheel 632B 14 hours ago _0.si
.rw-r--r-- alr wheel 156B 14 hours ago segments_1
.rw-r--r-- alr wheel   0B 14 hours ago write.lock
```

Deletions

- Deletion: Create new document, mark old document as deleted
- Query: Tombstone files
- Cleanup & Performance: Merging existing segments

Deletions

```java
IndexWriterConfig iwc = new IndexWriterConfig();
iwc.setUseCompoundFile(false);
iwc.setCodec(new SimpleTextCodec());

try (Directory dir = FSDirectory.open(indexPath);IndexWriter writer = new IndexWriter(dir, iwc)) {
    writer.addDocument(doc("Elasticsearch - the definitive guide", "en", 721));
    writer.addDocument(doc("Elasticsearch in Action", "en", 475));
    writer.addDocument(doc("Elastic Stack 8.x Cookbook", "en", 688));
    writer.addDocument(doc("Elasticsearch: Ein praktischer Einstieg", "de", 262));
    writer.addDocument(doc("Designing data-intensive Applications", "en", 614));
    writer.commit();
    writer.deleteDocuments(new Term("lang", "de"));
    writer.commit();
}
```

[finished]

```
Writing index...✅
```

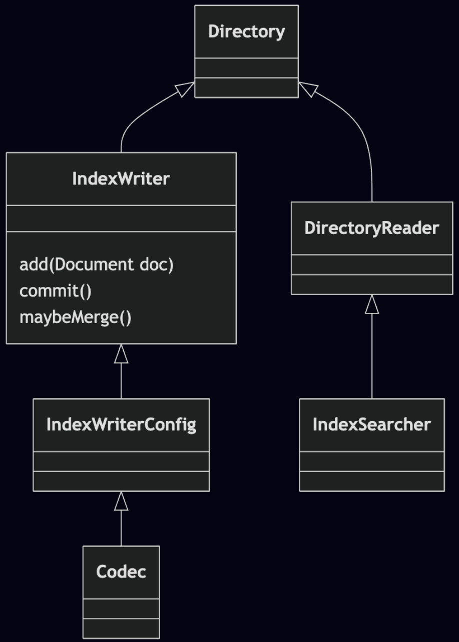
Deletions

```
cat /tmp/lucene/004-deletions/_0_1.liv
```

[finished]

```
size 5
  doc 0
  doc 1
  doc 2
  doc 4
END
checksum 00000000003447506980
```

Class Summary



- Reading & Writing separate
- ... but fight for the same resources
- `IndexSearcher` can be created from `IndexWriter`
- Merges can be limited, can also stop indexing
- Overwhelming indexing causes backpressure
- Serverless: Seperating read & write

Performance

- Two-phase iterators
- Block max WAND
- FSTs

Two phase iterators

- Approximation phase (cheap)
- Match phase (expensive)
- Phrase search `quick brown fox`: approximation all docs that match terms, match all docs right next to each other
- Geo search: approximate within lat/lon bounds, match doing complex geometry calculations
- Combination allows for heavy reduction in approximation phase, `title:"quick brown fox" AND pages:[300 TO *]`

Optimize while running a query

- Search for `quick OR brown OR fox`, get top 10 docs
- First top 10 is collected, min score to get into list is set
- WAND pruning: Faster skipping due to min score
- Block-max pruning: On a per block figure out max possible score for each term. If that sum is lower, skip the whole block without doing any doc comparison
- The higher the score of the last document is, the more documents get skipped, speeding up the search

FST - Finite state transducers

- Used for fast lookups
- A weighted graph (weights allow for fast pruning)
- Generation expensive, usage very fast
- Used for synonyms, Kuromoji / Nori analyzers, autocomplete
- Can go off-heap as they are just byte arrays

Skipping - being lazy makes you fast

- Goal: From `O(n)` linear scans `O(log n)` or `O(1)`
- Most simple way of skipping: index sorting
- BKD tree pruning for range/geo queries
- Example: `DocValuesSkipper` uses a multi level skip index for fast inclusion/exclusion
    - Meta file: Per field with skip index: offset, length, maxValue, minValue, docCount, maxDocID
    - Data file: At `offset`, for `length` bytes: a sequence of intervals. Each interval: 1 byte level count, then 28 bytes × levels (maxDocID, min DocID, maxValue, minValue, docCount per level), levels written from highest to 0, max of 4 levels

Memory Mapping - Let the OS do the work

- OS maps file contents into memory
- Reads can become fast as RAM access
- OS takes care of caching, no implementation needed
- `MMapDirectory` creates `MemorySegment` (Foreign Function & Memory API, since Java 22)
- Eliminates per-read system call overhead
- Developers can give hints to the operating system

ReadAdvice - Helping the OS

```java
private int mapReadAdvice(ReadAdvice readAdvice) {
  return switch (readAdvice) {
    case NORMAL -> POSIX_MADV_NORMAL;
    case RANDOM -> POSIX_MADV_RANDOM;
    case SEQUENTIAL -> POSIX_MADV_SEQUENTIAL;
  };
}
```

Tell the operating system if we're planning to read more data, so it can eagerly page in.

madvise - Helping the OS

- `madvise - give advice about use of memory` https://man7.org/linux/man-pages/man2/madvise.2.html
- Tell the operating system, what Lucene needs
- `MADV_RANDOM`: expect random access
- `MADV_SEQUENTIAL`: expect sequential access, read ahead is useful, dropping after reading as well
- `MADV_WILLNEED`: will need in the future, please page
- Lucene uses `MemorySegment` to map segments into pages

Prefetching

- `IndexInput.prefetch(offset, length)` calls `madvise(MADV_WILLNEED)`
- Tracks if segments have been loaded
- Lucene API: `IndexInput.prefetch()`
- For StoredFields you have a `StoredFields.prefetch(docId)`
- Thanks to the index structure Lucene knows where a docId is on disk and can tell the Linux kernel what part of a file to memory map

etcd

- `LRUQueryCache` - Caches `Query -> DocIdSet` after a certain number of usages, also requires segments to be a certain size. Cache is bound to segment lifecycle - which is immutable.
- `RoaringBitSets`: Reduce memory foot print compared to fixed bitsets
- `Accountable`: Try to keep everything accountable, so that you can trigger flushes, observe your code and even prevent OOMs
- Data can be loaded on-heap or off-heap, i.e. FSTs

Vector Search

```java
void main() throws Exception {
    System.setErr(new PrintStream(OutputStream.nullOutputStream()));
    Path indexPath = Paths.get("/tmp/lucene/006-vector-example/");
    FileUtils.deleteDirectory(indexPath.toFile());
    System.out.print("Writing index...");

    try (Directory dir = FSDirectory.open(indexPath);
         IndexWriter writer = new IndexWriter(dir, new IndexWriterConfig())) {

        writer.addDocument(createDocument("1", "electronics", new float[]{0.1f, 0.9f, 0.3f}));
        writer.addDocument(createDocument("2", "electronics", new float[]{0.8f, 0.2f, 0.5f}));
        writer.addDocument(createDocument("3", "other", new float[]{0.1f, 0.9f, 0.3f}));
    }

    System.out.println("✅");
}

private Document createDocument(String id, String category, float[] embedding) {
    Document doc = new Document();
    doc.add(new StringField("id", id, Field.Store.YES));
    doc.add(new StringField("category", category, Field.Store.YES));
    doc.add(new KnnFloatVectorField("embedding", embedding, VectorSimilarityFunction.COSINE));
    return doc;
}
```

[finished]

```
Writing index...✅
```

Vector Search

```
try (IndexReader reader = DirectoryReader.open(FSDirectory.open(indexPath))) {
    IndexSearcher searcher = new IndexSearcher(reader);

    float[] queryVector = {0.15f, 0.85f, 0.35f};

    Query filter = new TermQuery(new Term("category", "electronics"));
    Query query = new KnnFloatVectorQuery("embedding", queryVector, 5, filter);
    TopDocs results = searcher.search(query, 5);

    for (ScoreDoc sd : results.scoreDocs) {
        String id = reader.storedFields().document(sd.doc).get("id");
        System.out.printf("id=%-8s  score=%.4f%n", id, sd.score);
    }
}
```

```
──────────────────────────────── [finished with error] ────────────────────────────────
```

Use-case: Hotel night reservation

- Problem: Hotel has rooms, each room has an availability
- Query: Is there a room for 3 nights from X to Y
- Data structure: `BitSet`, one year fits in 384 bits per room, 48 bytes
- Lucene: Store BitSet in `BinaryDocValues`

Use-case: Hotel night reservation

```
// Room 101 — Single: unavailable first week of February (days 31-37)
BitSet room101 = allAvailable();
room101.clear(31, 38);
writer.addDocument(hotelRoom("Room 101", "Single", room101));
```

```
private BitSet allAvailable() {
    BitSet bs = new BitSet(384);
    bs.set(0, 365); // bits 0-364 = Jan 1 - Dec 31
    return bs;
}

private Document hotelRoom(String name, String type, BitSet availability) {
    Document doc = new Document();
    doc.add(new StringField("name", name, Field.Store.YES));
    doc.add(new StringField("type", type, Field.Store.YES));
    // Serialize the BitSet into exactly 48 bytes (384 bits)
    byte[] bytes = new byte[48];
    byte[] bsBytes = availability.toByteArray();
    System.arraycopy(bsBytes, 0, bytes, 0, Math.min(bsBytes.length, 48));
    doc.add(new BinaryDocValuesField("availability", new BytesRef(bytes)));
    return doc;
}
```

[finished]

```
Writing index...✅
```

Queries

```
    try (Directory dir = FSDirectory.open(indexPath);
         DirectoryReader reader = DirectoryReader.open(dir)) {

        // Search for rooms available for 3 nights: March 15-17
        // day 0 = Jan 1, so March 15 = day 31+28+14 = 73
        BitSet requestedNights = new BitSet(384);
        requestedNights.set(73, 76); // bits 73, 74, 75 → March 15, 16, 17

        IndexSearcher searcher = new IndexSearcher(reader);
        var query = new HotelAvailabilityQuery("availability", requestedNights);
        TopDocs topDocs = searcher.search(query, 10);

        System.out.printf("Found %d available room(s):%n", topDocs.totalHits.value());
        for (ScoreDoc sd : topDocs.scoreDocs) {
            Document doc = reader.storedFields().document(sd.doc);
            System.out.printf("  - %s (%s)%n", doc.get("name"), doc.get("type"));
        }
    }
```

[finished with error]

```
Searching for rooms available March 15-17 (bits 73-75)...
```

Rethink into a search problem

- BitSet approach: scan all documents, O(n) regardless of matches
- Keyword approach: index each available night as a date term `"2025-03-15"`
- Query for March 15–17 → `BooleanQuery` with 3 MUST `TermQuery` clauses
- Lucene intersects posting lists — never visits non-matching rooms
- No custom Query class needed; naturally extends to any year

Keyword Availability: Indexing

```java
private Document hotelRoom(String name, String type, List<LocalDate> availableDates) {
    Document doc = new Document();
    doc.add(new StringField("name", name, Field.Store.YES));
    doc.add(new StringField("type", type, Field.Store.YES));
    for (LocalDate date : availableDates) {
        doc.add(new StringField("night", date.toString(), Field.Store.NO));
    }
    return doc;
}
```

[finished]

```
Writing index...✅
```

Keyword Availability: Querying

```java
BooleanQuery.Builder builder = new BooleanQuery.Builder();
builder.add(new TermQuery(new Term("night", "2025-03-15")), BooleanClause.Occur.MUST);
builder.add(new TermQuery(new Term("night", "2025-03-16")), BooleanClause.Occur.MUST);
builder.add(new TermQuery(new Term("night", "2025-03-17")), BooleanClause.Occur.MUST);
BooleanQuery query = builder.build();

System.out.println("Searching for rooms available March 15-17, 2025...");

try (Directory dir = FSDirectory.open(indexPath);
     DirectoryReader reader = DirectoryReader.open(dir)) {
    IndexSearcher searcher = new IndexSearcher(reader);
    TopDocs topDocs = searcher.search(query, 10);

    System.out.printf("Found %d available room(s):%n", topDocs.totalHits.value());
    for (ScoreDoc sd : topDocs.scoreDocs) {
        Document doc = reader.storedFields().document(sd.doc);
        System.out.printf("  - %s (%s)%n", doc.get("name"), doc.get("type"));
    }
}
```

[finished with error]

```
Searching for rooms available March 15-17, 2025...
```

Rethink into a numbers problem

- IntRange approach: index booked periods, not available nights
- Each booking = one `IntRange` value: days as integers (Jan 1 = 0, Dec 31 = 364)
- Query: `MUST_NOT newIntersectsQuery(...)` excludes rooms with conflicting bookings

IntRange Availability: Indexing

```
private IntRange booked(LocalDate from, LocalDate to) {
    int fromDay = from.getDayOfYear() - 1;
    int toDay   = to.getDayOfYear() - 1;
    return new IntRange("booked", new int[]{fromDay}, new int[]{toDay});
}

private Document hotelRoom(String name, String type, IntRange... bookings) {
    Document doc = new Document();
    doc.add(new StringField("name", name, Field.Store.YES));
    doc.add(new StringField("type", type, Field.Store.YES));
    for (IntRange booking : bookings) {
        doc.add(booking);
    }
    return doc;
}
```

[finished]

```
Writing index...✅
```

IntRange Availability: Querying

```
// Search for rooms available for a 3-night stay: March 15-17, 2025
LocalDate checkIn  = LocalDate.of(2025, 3, 15);
LocalDate checkOut = LocalDate.of(2025, 3, 17);
int checkInDay  = checkIn.getDayOfYear() - 1;
int checkOutDay = checkOut.getDayOfYear() - 1;

Query conflict = IntRange.newIntersectsQuery("booked",
        new int[]{checkInDay}, new int[]{checkOutDay});
Query query = new BooleanQuery.Builder()
        .add(new MatchAllDocsQuery(), BooleanClause.Occur.MUST)
        .add(conflict, BooleanClause.Occur.MUST_NOT)
        .build();
```

[finished with error]

```
Searching for rooms available March 15-17, 2025...
Conflict query: booked:<ranges:[73 : 75]>
```

Size comparison

▓▓▓ 1 million rooms, random, but same bookings

- BitSet: 46 MB, index creation took 1s, running 1k queries, 36s
- Keyword: 33 MB, index creation took 45s, running 10k queries 8.7s
- IntRange: 5 MB, index creation took 2s, running 10k queries 0.3s

P.S. Completely unscientific test

Extra: Sample SpellChecker

```
    try (Directory dir = FSDirectory.open(mainPath); IndexWriter writer = new IndexWriter(dir, iwc)) {
        addDoc(writer, "Lucene in Action",                 "lucene search searcher indexing querying scoring
ranking");
        addDoc(writer, "Elasticsearch: The Definitive Guide", "elasticsearch search indexing querying documents
");
        addDoc(writer, "Search Algorithms",                "search algorithm sorting binary data structures");
        addDoc(writer, "Database Design",                  "database query schema indexing table join");
        addDoc(writer, "Data Structures and Algorithms", "data structure sorting algorithm binary tree");
        addDoc(writer, "Introduction to Algorithms",       "algorithm data sorting searching tree graph");
        addDoc(writer, "Clean Code",                       "code refactoring method function testing");
        addDoc(writer, "The Pragmatic Programmer",         "programming code testing debugging practice");
        addDoc(writer, "Machine Learning Basics",          "machine learning data model training algorithm");
        addDoc(writer, "Database Systems",                 "database query transaction indexing schema");
        // typo sneaked into this doc
        addDoc(writer, "Sorting and Search Basics",        "sortign search comparison data");
        writer.commit();
    }
```

[finished]

```
Indexing documents...✅
```

Extra: Sample SpellChecker

```
void main() throws Exception {
    System.setErr(new PrintStream(OutputStream.nullOutputStream()));
    Path mainPath  = Paths.get("/tmp/lucene/011-spellcheck/main/");

    try (Directory mainDir     = FSDirectory.open(mainPath);
         DirectoryReader reader = DirectoryReader.open(mainDir)) {
        DirectSpellChecker spell = new DirectSpellChecker();

        suggest("algorythm", SUGGEST_WHEN_NOT_IN_INDEX, spell, reader);
        suggest("databse", SUGGEST_WHEN_NOT_IN_INDEX, spell, reader);
        suggest("sortign", SUGGEST_WHEN_NOT_IN_INDEX, spell, reader);
        suggest("sortign", SUGGEST_ALWAYS, spell, reader);
        suggest("sortign", SUGGEST_MORE_POPULAR, spell, reader);
    }
}

private void suggest(String term, SuggestMode mode, DirectSpellChecker spell, IndexReader reader) throws
IOException {
    SuggestWord[] s = spell.suggestSimilar(new Term("body", term), 3, reader, mode);
    System.out.printf("%-15s %-20s  %-22s%n", term, mode.name(), Arrays.toString(s));
}
```

[finished with error]

Extra: Built an ultra fast suggester

```java
//DEPS org.apache.lucene:lucene-core:10.3.2
//DEPS org.apache.lucene:lucene-suggest:10.3.2

import org.apache.lucene.search.suggest.FileDictionary;
import org.apache.lucene.search.suggest.Lookup.LookupResult;
import org.apache.lucene.search.suggest.fst.WFSTCompletionLookup;
import org.apache.lucene.store.ByteBuffersDirectory;

void main() throws Exception {
    System.setErr(new PrintStream(OutputStream.nullOutputStream()));
    var lookup = new WFSTCompletionLookup(new ByteBuffersDirectory(), "wfst");
    lookup.build(new FileDictionary(new StringReader(
        "Berlin" + "\t" + 3755000 + "\n" + "Hamburg" + "\t" + 1853000 + "\n" +
        "München" + "\t" + 1512000 + "\n" + "Köln" + "\t" + 1084000 + "\n" +
        "Frankfurt" + "\t" + 773000 + "\n" + "Stuttgart" + "\t" + 632000 + "\n" +
        "Düsseldorf" + "\t" + 621000 + "\n" + "Leipzig" + "\t" + 620000 + "\n" +
        "Dortmund" + "\t" + 588000 + "\n" + "Essen" + "\t" + 582000 + "\n" +
        "Münster" + "\t" + 317000 + "\n" + "Mülheim" + "\t" + 170000 + "\n" +
        "Mühlacker" + "\t" + 27000 + "\n" + "Mühldorf" + "\t" + 18000 + "\n"
    )));

    for (String prefix : List.of("M", "Mü", "Mün", "D")) {
        System.out.println("\nPrefix \"" + prefix + "\":");
        for (LookupResult r : lookup.lookup(prefix, null, false, 5))
            System.out.printf("  %-15s %,d%n", r.key, r.value);
    }
}
```

[finished]

Extra: Built an ultra fast suggester

```
Prefix "M":
  München         1.512.000
  Münster         317.000
  Mülheim         170.000
  Mühlacker       27.000
  Mühldorf        18.000

Prefix "Mü":
  München         1.512.000
  Münster         317.000
  Mülheim         170.000
  Mühlacker       27.000
  Mühldorf        18.000

Prefix "Mün":
  München         1.512.000
  Münster         317.000

Prefix "D":
  Düsseldorf      621.000
  Dortmund        588.000
```

Summary

- Lucene keeps innovating, makes use of latest Java features
- Way more than just full text search
- If you don't want to deal with it, use an existing search engine like Elasticsearch (+ distribution, HTTP, use-cases, model integrations on hosted versions)
- Hybrid search, focus on intent
- Searches are changing
    - Humans: keyword driven
    - LLMs: Natural language driven
- If you haven't, invest time in benchmark infrastructure, check out Lucene benchmarks https://benchmarks.mikemccandless.com/

Links

- Search Benchmark, the game https://tantivy-search.github.io/bench/
- File formats https://lucene.apache.org/core/10_3_2/core/org/apache/lucene/codecs/lucene103/package-summary.html#package.description

Discussion

- Questions? Answers?
  - mailto:alr@spinscale.de
  - web:spinscale.de
  - mastodon:@spinscale@mastodon.social
  - bluesky:@spinscale.bsky.social
  - linkedin:alexanderreelsen