package com.twitter.search.core.earlybird.index.inverted;

import java.io.IOException;

import java.util.Collections;

import java.util.List;

import java.util.Map;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import org.apache.lucene.index.PostingsEnum;

import org.apache.lucene.util.BytesRef;

import com.twitter.search.core.earlybird.index.DocIDToTweetIDMapper;

/\*\*

\* A PostingsEnum that maps doc IDs in one DocIDToTweetIDMapper instance to doc IDs in another

\* DocIDToTweetIDMapper.

\*

\* Unoptimized segments can use any DocIDToTweetIDMapper they want, which means that there are no

\* guarantees on the distribution of the doc IDs in this mapper. However, optimized segments must

\* use an OptimizedTweetIDMapper: we want to assign sequential doc IDs and use delta encondings in

\* order to save space. So when an Earlybird segment needs to be optimized, we might need to convert

\* the doc ID space of the unoptimized tweet ID mapper to the doc ID space of the optimized mapper.

\* However, once we do this, the doc IDs stored in the posting lists in that segment will no longer

\* be valid, unless we remap them too. So the goal of this class is to provide a way to do that.

\*

\* When we want to optimize a posting list, we need to traverse it and pack it. This class provides

\* a wrapper around the original posting list that does the doc ID remapping at traversal time.

\*/

public class OptimizingPostingsEnumWrapper extends PostingsEnum {

private final List<Integer> docIds = Lists.newArrayList();

private final Map<Integer, List<Integer>> positions = Maps.newHashMap();

private int docIdIndex = -1;

private int positionIndex = -1;

public OptimizingPostingsEnumWrapper(PostingsEnum source,

DocIDToTweetIDMapper originalTweetIdMapper,

DocIDToTweetIDMapper newTweetIdMapper) throws IOException {

int docId;

while ((docId = source.nextDoc()) != NO\_MORE\_DOCS) {

long tweetId = originalTweetIdMapper.getTweetID(docId);

int newDocId = newTweetIdMapper.getDocID(tweetId);

Preconditions.checkState(newDocId != DocIDToTweetIDMapper.ID\_NOT\_FOUND,

"Did not find a mapping in the new tweet ID mapper for tweet ID %s, doc ID %s",

tweetId, docId);

docIds.add(newDocId);

List<Integer> docPositions = Lists.newArrayListWithCapacity(source.freq());

positions.put(newDocId, docPositions);

for (int i = 0; i < source.freq(); ++i) {

docPositions.add(source.nextPosition());

}

}

Collections.sort(docIds);

}

@Override

public int nextDoc() {

++docIdIndex;

if (docIdIndex >= docIds.size()) {

return NO\_MORE\_DOCS;

}

positionIndex = -1;

return docIds.get(docIdIndex);

}

@Override

public int freq() {

Preconditions.checkState(docIdIndex >= 0, "freq() called before nextDoc().");

Preconditions.checkState(docIdIndex < docIds.size(),

"freq() called after nextDoc() returned NO\_MORE\_DOCS.");

return positions.get(docIds.get(docIdIndex)).size();

}

@Override

public int nextPosition() {

Preconditions.checkState(docIdIndex >= 0, "nextPosition() called before nextDoc().");

Preconditions.checkState(docIdIndex < docIds.size(),

"nextPosition() called after nextDoc() returned NO\_MORE\_DOCS.");

++positionIndex;

Preconditions.checkState(positionIndex < positions.get(docIds.get(docIdIndex)).size(),

"nextPosition() called more than freq() times.");

return positions.get(docIds.get(docIdIndex)).get(positionIndex);

}

// All other methods are not supported.

@Override

public int advance(int target) {

throw new UnsupportedOperationException(

"OptimizingPostingsEnumWrapper.advance() is not supported.");

}

@Override

public long cost() {

throw new UnsupportedOperationException(

"OptimizingPostingsEnumWrapper.cost() is not supported.");

}

@Override

public int docID() {

throw new UnsupportedOperationException(

"OptimizingPostingsEnumWrapper.docID() is not supported.");

}

@Override

public int endOffset() {

throw new UnsupportedOperationException(

"OptimizingPostingsEnumWrapper.endOffset() is not supported.");

}

@Override

public BytesRef getPayload() {

throw new UnsupportedOperationException(

"OptimizingPostingsEnumWrapper.getPayload() is not supported.");

}

@Override

public int startOffset() {

throw new UnsupportedOperationException(

"OptimizingPostingsEnumWrapper.startOffset() is not supported.");

}

}