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

import java.io.IOException;

import java.util.Comparator;

import org.apache.lucene.index.BaseTermsEnum;

import org.apache.lucene.index.ImpactsEnum;

import org.apache.lucene.index.PostingsEnum;

import org.apache.lucene.index.SlowImpactsEnum;

import org.apache.lucene.index.TermsEnum;

import org.apache.lucene.util.BytesRef;

import org.apache.lucene.util.InPlaceMergeSorter;

import org.apache.lucene.util.IntsRefBuilder;

import org.apache.lucene.util.fst.BytesRefFSTEnum;

import org.apache.lucene.util.fst.FST;

import org.apache.lucene.util.fst.PositiveIntOutputs;

import org.apache.lucene.util.fst.Util;

import org.apache.lucene.util.packed.PackedInts;

import com.twitter.search.common.util.io.flushable.DataDeserializer;

import com.twitter.search.common.util.io.flushable.DataSerializer;

import com.twitter.search.common.util.io.flushable.FlushInfo;

import com.twitter.search.common.util.io.flushable.Flushable;

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

public class FSTTermDictionary implements TermDictionary, Flushable {

private final FST<Long> fst;

private final PackedInts.Reader termPointers;

private final ByteBlockPool termPool;

private final TermPointerEncoding termPointerEncoding;

private int numTerms;

FSTTermDictionary(int numTerms, FST<Long> fst,

ByteBlockPool termPool, PackedInts.Reader termPointers,

TermPointerEncoding termPointerEncoding) {

this.numTerms = numTerms;

this.fst = fst;

this.termPool = termPool;

this.termPointers = termPointers;

this.termPointerEncoding = termPointerEncoding;

}

@Override

public int getNumTerms() {

return numTerms;

}

@Override

public int lookupTerm(BytesRef term) throws IOException {

if (fst == null) {

return EarlybirdIndexSegmentAtomicReader.TERM\_NOT\_FOUND;

}

final BytesRefFSTEnum<Long> fstEnum = new BytesRefFSTEnum<>(fst);

final BytesRefFSTEnum.InputOutput<Long> result = fstEnum.seekExact(term);

if (result != null && result.input.equals(term)) {

// -1 because 0 is not supported by the fst

return result.output.intValue() - 1;

} else {

return EarlybirdIndexSegmentAtomicReader.TERM\_NOT\_FOUND;

}

}

static FSTTermDictionary buildFST(

final ByteBlockPool termPool,

int[] termPointers,

int numTerms,

final Comparator<BytesRef> comp,

boolean supportTermTextLookup,

final TermPointerEncoding termPointerEncoding) throws IOException {

final IntsRefBuilder scratchIntsRef = new IntsRefBuilder();

final int[] compact = new int[numTerms];

for (int i = 0; i < numTerms; i++) {

compact[i] = i;

}

// first sort the terms

new InPlaceMergeSorter() {

private BytesRef scratch1 = new BytesRef();

private BytesRef scratch2 = new BytesRef();

@Override

protected void swap(int i, int j) {

final int o = compact[i];

compact[i] = compact[j];

compact[j] = o;

}

@Override

protected int compare(int i, int j) {

final int ord1 = compact[i];

final int ord2 = compact[j];

ByteTermUtils.setBytesRef(termPool, scratch1,

termPointerEncoding.getTextStart(termPointers[ord1]));

ByteTermUtils.setBytesRef(termPool, scratch2,

termPointerEncoding.getTextStart(termPointers[ord2]));

return comp.compare(scratch1, scratch2);

}

}.sort(0, compact.length);

final PositiveIntOutputs outputs = PositiveIntOutputs.getSingleton();

final org.apache.lucene.util.fst.Builder<Long> builder =

new org.apache.lucene.util.fst.Builder<>(FST.INPUT\_TYPE.BYTE1, outputs);

final BytesRef term = new BytesRef();

for (int termID : compact) {

ByteTermUtils.setBytesRef(termPool, term,

termPointerEncoding.getTextStart(termPointers[termID]));

// +1 because 0 is not supported by the fst

builder.add(Util.toIntsRef(term, scratchIntsRef), (long) termID + 1);

}

if (supportTermTextLookup) {

PackedInts.Reader packedTermPointers = OptimizedMemoryIndex.getPackedInts(termPointers);

return new FSTTermDictionary(

numTerms,

builder.finish(),

termPool,

packedTermPointers,

termPointerEncoding);

} else {

return new FSTTermDictionary(

numTerms,

builder.finish(),

null, // termPool

null, // termPointers

termPointerEncoding);

}

}

@Override

public boolean getTerm(int termID, BytesRef text, BytesRef termPayload) {

if (termPool == null) {

throw new UnsupportedOperationException(

"This dictionary does not support term lookup by termID");

} else {

int termPointer = (int) termPointers.get(termID);

boolean hasTermPayload = termPointerEncoding.hasPayload(termPointer);

int textStart = termPointerEncoding.getTextStart(termPointer);

// setBytesRef sets the passed in BytesRef "text" to the term in the termPool.

// As a side effect it returns the offset of the next entry in the pool after the term,

// which may optionally be used if this term has a payload.

int termPayloadStart = ByteTermUtils.setBytesRef(termPool, text, textStart);

if (termPayload != null && hasTermPayload) {

ByteTermUtils.setBytesRef(termPool, termPayload, termPayloadStart);

}

return hasTermPayload;

}

}

@Override

public TermsEnum createTermsEnum(OptimizedMemoryIndex index) {

return new BaseTermsEnum() {

private final BytesRefFSTEnum<Long> fstEnum = fst != null ? new BytesRefFSTEnum<>(fst) : null;

private BytesRefFSTEnum.InputOutput<Long> current;

@Override

public SeekStatus seekCeil(BytesRef term)

throws IOException {

if (fstEnum == null) {

return SeekStatus.END;

}

current = fstEnum.seekCeil(term);

if (current != null && current.input.equals(term)) {

return SeekStatus.FOUND;

} else {

return SeekStatus.END;

}

}

@Override

public boolean seekExact(BytesRef text) throws IOException {

current = fstEnum.seekExact(text);

return current != null;

}

// In our case the ord is the termId.

@Override

public void seekExact(long ord) {

current = new BytesRefFSTEnum.InputOutput<>();

current.input = null;

// +1 because 0 is not supported by the fst

current.output = ord + 1;

if (termPool != null) {

BytesRef bytesRef = new BytesRef();

int termId = (int) ord;

assert termId == ord;

FSTTermDictionary.this.getTerm(termId, bytesRef, null);

current.input = bytesRef;

}

}

@Override

public BytesRef next() throws IOException {

current = fstEnum.next();

if (current == null) {

return null;

}

return current.input;

}

@Override

public BytesRef term() {

return current.input;

}

// In our case the ord is the termId.

@Override

public long ord() {

// -1 because 0 is not supported by the fst

return current.output - 1;

}

@Override

public int docFreq() {

return index.getDF((int) ord());

}

@Override

public long totalTermFreq() {

return docFreq();

}

@Override

public PostingsEnum postings(PostingsEnum reuse, int flags) throws IOException {

int termID = (int) ord();

int postingsPointer = index.getPostingListPointer(termID);

int numPostings = index.getNumPostings(termID);

return index.getPostingLists().postings(postingsPointer, numPostings, flags);

}

@Override

public ImpactsEnum impacts(int flags) throws IOException {

return new SlowImpactsEnum(postings(null, flags));

}

};

}

@SuppressWarnings("unchecked")

@Override

public FlushHandler getFlushHandler() {

return new FlushHandler(this);

}

public static class FlushHandler extends Flushable.Handler<FSTTermDictionary> {

private static final String NUM\_TERMS\_PROP\_NAME = "numTerms";

private static final String SUPPORT\_TERM\_TEXT\_LOOKUP\_PROP\_NAME = "supportTermTextLookup";

private final TermPointerEncoding termPointerEncoding;

public FlushHandler(TermPointerEncoding termPointerEncoding) {

super();

this.termPointerEncoding = termPointerEncoding;

}

public FlushHandler(FSTTermDictionary objectToFlush) {

super(objectToFlush);

this.termPointerEncoding = objectToFlush.termPointerEncoding;

}

@Override

protected void doFlush(FlushInfo flushInfo, DataSerializer out)

throws IOException {

FSTTermDictionary objectToFlush = getObjectToFlush();

flushInfo.addIntProperty(NUM\_TERMS\_PROP\_NAME, objectToFlush.getNumTerms());

flushInfo.addBooleanProperty(SUPPORT\_TERM\_TEXT\_LOOKUP\_PROP\_NAME,

objectToFlush.termPool != null);

if (objectToFlush.termPool != null) {

out.writePackedInts(objectToFlush.termPointers);

objectToFlush.termPool.getFlushHandler().flush(flushInfo.newSubProperties("termPool"), out);

}

objectToFlush.fst.save(out.getIndexOutput());

}

@Override

protected FSTTermDictionary doLoad(FlushInfo flushInfo,

DataDeserializer in) throws IOException {

int numTerms = flushInfo.getIntProperty(NUM\_TERMS\_PROP\_NAME);

boolean supportTermTextLookup =

flushInfo.getBooleanProperty(SUPPORT\_TERM\_TEXT\_LOOKUP\_PROP\_NAME);

PackedInts.Reader termPointers = null;

ByteBlockPool termPool = null;

if (supportTermTextLookup) {

termPointers = in.readPackedInts();

termPool = (new ByteBlockPool.FlushHandler())

.load(flushInfo.getSubProperties("termPool"), in);

}

final PositiveIntOutputs outputs = PositiveIntOutputs.getSingleton();

return new FSTTermDictionary(numTerms, new FST<>(in.getIndexInput(), outputs),

termPool, termPointers, termPointerEncoding);

}

}

}