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

import org.apache.lucene.util.ByteBlockPool;

import org.apache.lucene.util.BytesRef;

import org.apache.lucene.util.StringHelper;

/\*\*

\* Utility class for BytePools which have each term's length encoded before the contents in the

\* ByteBlockPool

\* Another solution is to have a class that encapsulates both textStarts and the byteBlockPool and

\* knows how the byteBlockPool is used to store the strings

\*\*/

public abstract class ByteTermUtils {

/\*\*

\* Fill in a BytesRef from term's length & bytes encoded in byte block

\*/

public static int setBytesRef(final BaseByteBlockPool byteBlockPool,

BytesRef term,

final int textStart) {

final byte[] block = term.bytes =

byteBlockPool.pool.buffers[textStart >>> ByteBlockPool.BYTE\_BLOCK\_SHIFT];

final int start = textStart & ByteBlockPool.BYTE\_BLOCK\_MASK;

int pos = start;

byte b = block[pos++];

term.length = b & 0x7F;

for (int shift = 7; (b & 0x80) != 0; shift += 7) {

b = block[pos++];

term.length |= (b & 0x7F) << shift;

}

term.offset = pos;

assert term.length >= 0;

return textStart + (pos - start) + term.length;

}

/\*\*

\* Test whether the text for current RawPostingList p equals

\* current tokenText in utf8.

\*/

public static boolean postingEquals(final BaseByteBlockPool termPool,

final int textStart, final BytesRef other) {

final byte[] block = termPool.pool.getBlocks()[textStart >>> ByteBlockPool.BYTE\_BLOCK\_SHIFT];

assert block != null;

int pos = textStart & ByteBlockPool.BYTE\_BLOCK\_MASK;

byte b = block[pos++];

int len = b & 0x7F;

for (int shift = 7; (b & 0x80) != 0; shift += 7) {

b = block[pos++];

len |= (b & 0x7F) << shift;

}

if (len == other.length) {

final byte[] utf8Bytes = other.bytes;

for (int tokenPos = other.offset;

tokenPos < other.length + other.offset; pos++, tokenPos++) {

if (utf8Bytes[tokenPos] != block[pos]) {

return false;

}

}

return true;

} else {

return false;

}

}

/\*\*

\* Returns the hashCode of the term stored at the given position in the block pool.

\*/

public static int hashCode(

final BaseByteBlockPool termPool, final int textStart) {

final byte[] block = termPool.pool.getBlocks()[textStart >>> ByteBlockPool.BYTE\_BLOCK\_SHIFT];

final int start = textStart & ByteBlockPool.BYTE\_BLOCK\_MASK;

int pos = start;

byte b = block[pos++];

int len = b & 0x7F;

for (int shift = 7; (b & 0x80) != 0; shift += 7) {

b = block[pos++];

len |= (b & 0x7F) << shift;

}

// Hash code returned here must be consistent with the one used in TermHashTable.lookupItem, so

// use the fixed hash seed. See TermHashTable.lookupItem for explanation of fixed hash seed.

return StringHelper.murmurhash3\_x86\_32(block, pos, len, InvertedRealtimeIndex.FIXED\_HASH\_SEED);

}

/\*\*

\* Copies the utf8 encoded byte ref to the termPool.

\* @param termPool

\* @param utf8

\* @return The text's start position in the termPool

\*/

public static int copyToTermPool(BaseByteBlockPool termPool, BytesRef bytes) {

// Maybe grow the termPool before we write. Assume we need 5 bytes in

// the worst case to store the VInt.

if (bytes.length + 5 + termPool.byteUpto > ByteBlockPool.BYTE\_BLOCK\_SIZE) {

// Not enough room in current block

termPool.nextBuffer();

}

final int textStart = termPool.byteUpto + termPool.byteOffset;

writeVInt(termPool, bytes.length);

System.arraycopy(bytes.bytes, bytes.offset, termPool.buffer, termPool.byteUpto, bytes.length);

termPool.byteUpto += bytes.length;

return textStart;

}

private static void writeVInt(final BaseByteBlockPool termPool, final int v) {

int value = v;

final byte[] block = termPool.buffer;

int blockUpto = termPool.byteUpto;

while ((value & ~0x7F) != 0) {

block[blockUpto++] = (byte) ((value & 0x7f) | 0x80);

value >>>= 7;

}

block[blockUpto++] = (byte) value;

termPool.byteUpto = blockUpto;

}

}