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

/\*\*

\* A packed ints writer writing packed values (int/long) into {@link IntBlockPool}.

\* @see IntBlockPoolPackedLongsReader

\*

\* A standard useage would be:

\* - set writer at an int block pool pointer and number of bits per packed value:

\* {@link #jumpToInt(int, int)}

\* - write: {@link #writePackedInt(int)} or {@link #writePackedLong(long)}

\*

\* Example usage:

\* @see HighDFPackedIntsPostingLists

\*/

public final class IntBlockPoolPackedLongsWriter {

/\*\*

\* Mask used to convert an int to a long. We cannot just cast because it will fill in the higher

\* 32 bits with the sign bit, but we need the higher 32 bits to be 0 instead.

\*/

private static final long LONG\_MASK = 0xFFFFFFFFL;

/\*\* The int block pool into which packed ints will be written. \*/

private final IntBlockPool intBlockPool;

/\*\* The value in the current position in the int block pool. \*/

private int currentIntValue = 0;

/\*\* Starting bit index of unused bits in {@link #currentIntValue}. \*/

private int currentIntBitIndex = 0;

/\*\* Pointer of {@link #currentIntValue} in {@link #intBlockPool}. \*/

private int currentIntPointer = -1;

/\*\*

\* Number of bits per packed value that will be written with

\* {@link #writePackedInt(int)} or {@link #writePackedLong(long)}.

\*/

private int numBitsPerPackedValue = -1;

/\*\*

\* Mask used to extract the lower {@link #numBitsPerPackedValue} in a given value.

\*/

private long packedValueBitsMask = 0;

/\*\*

\* Sole constructor.

\*

\* @param intBlockPool into which packed ints will be written

\*/

public IntBlockPoolPackedLongsWriter(IntBlockPool intBlockPool) {

this.intBlockPool = intBlockPool;

}

/\*\*

\* 1. Set this writer to start writing at the given int block pool pointer.

\* 2. Set number of bits per packed value that will be write.

\* 3. Re-set {@link #currentIntValue} and {@link #currentIntBitIndex} to 0.

\*

\* @param intBlockPoolPointer the position this writer should start writing packed values. This

\* pointer must be less then or equal to he length of the block pool.

\* Subsequent writes will {@link IntBlockPool#add(int)} to the

\* end of the int block pool if the given pointer equals to the length.

\* @param bitsPerPackedValue must be non-negative.

\*/

public void jumpToInt(int intBlockPoolPointer, int bitsPerPackedValue) {

assert intBlockPoolPointer <= intBlockPool.length();

assert bitsPerPackedValue >= 0;

// Set the writer to start writing at the given int block pool pointer.

this.currentIntPointer = intBlockPoolPointer;

// Set number of bits that will be write per packed value.

this.numBitsPerPackedValue = bitsPerPackedValue;

// Compute the mask used to extract lower number of bitsPerPackedValue.

this.packedValueBitsMask =

bitsPerPackedValue == Long.SIZE ? -1L : (1L << bitsPerPackedValue) - 1;

// Reset current int data to 0.

this.currentIntValue = 0;

this.currentIntBitIndex = 0;

}

/\*\*

\* The given int value will be ZERO extended to a long and written using

\* {@link #writePackedValueInternal(long)} (long)}.

\*

\* @see #LONG\_MASK

\*/

public void writePackedInt(final int value) {

assert numBitsPerPackedValue <= Integer.SIZE;

writePackedValueInternal(LONG\_MASK & value);

}

/\*\*

\* Write a long value.

\* The given long value must bu UNABLE to fit in an int.

\*/

public void writePackedLong(final long value) {

assert numBitsPerPackedValue <= Long.SIZE;

writePackedValueInternal(value);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Private Helper Method \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* Write the given number of bits of the given value into this int pool as a packed int.

\*

\* @param value value will be written

\*/

private void writePackedValueInternal(final long value) {

// Extract lower 'numBitsPerPackedValue' from the given value.

long val = value & packedValueBitsMask;

assert val == value : String.format(

"given value %d needs more bits than specified %d", value, numBitsPerPackedValue);

int numBitsWrittenCurIter;

int numBitsRemaining = numBitsPerPackedValue;

// Each iteration of this while loop is writing part of the given value.

while (numBitsRemaining > 0) {

// Write into 'currentIntValue' int.

currentIntValue |= val << currentIntBitIndex;

// Calculate number of bits have been written in this iteration,

// we either used up all the remaining bits in 'currentIntValue' or

// finished up writing the value, whichever is smaller.

numBitsWrittenCurIter = Math.min(Integer.SIZE - currentIntBitIndex, numBitsRemaining);

// Number of bits remaining should be decremented.

numBitsRemaining -= numBitsWrittenCurIter;

// Right shift the value to remove the bits have been written.

val >>>= numBitsWrittenCurIter;

// Update bit index in current int.

currentIntBitIndex += numBitsWrittenCurIter;

assert currentIntBitIndex <= Integer.SIZE;

flush();

// if 'currentIntValue' int is used up.

if (currentIntBitIndex == Integer.SIZE) {

currentIntPointer++;

currentIntValue = 0;

currentIntBitIndex = 0;

}

}

}

/\*\*

\* Flush the {@link #currentIntValue} int into the int pool if the any bits of the int are used.

\*/

private void flush() {

if (currentIntPointer == intBlockPool.length()) {

intBlockPool.add(currentIntValue);

assert currentIntPointer + 1 == intBlockPool.length();

} else {

intBlockPool.set(currentIntPointer, currentIntValue);

}

}

}