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

import javax.annotation.Nullable;

/\*\*

\* A packed ints reader reading packed values (int/long) written in {@link IntBlockPool}.

\* @see IntBlockPoolPackedLongsWriter

\*

\* A standard usage would be :

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

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

\* - read: {@link #readPackedLong()}

\*

\* Example usage:

\* @see HighDFPackedIntsDocsEnum

\* @see HighDFPackedIntsDocsAndPositionsEnum

\*/

public final class IntBlockPoolPackedLongsReader {

/\*\*

\* 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 from which packed ints will be read. \*/

private final IntBlockPool intBlockPool;

/\*\* Pre-computed shifts, masks, and start int indices used to decode packed ints. \*/

private final PackedLongsReaderPreComputedValues preComputedValues;

/\*\*

\* The underlying {@link #intBlockPool} will be read block by blocks. The current read

\* block will be identified by {@link #startPointerForCurrentBlock} and assigned to

\* {@link #currentBlock}. {@link #indexInCurrentBlock} will be used access values from the

\* {@link #currentBlock}.

\*/

private int[] currentBlock;

private int indexInCurrentBlock;

private int startPointerForCurrentBlock = -1;

/\*\*

\* Whether the decoded packed values are spanning more than 1 int.

\* @see #readPackedLong()

\*/

private boolean packedValueNeedsLong;

/\*\*

\* Masks used to extract packed values.

\* @see #readPackedLong()

\*/

private long packedValueMask;

/\*\* PRE-COMPUTED: The index of the first int that has a specific packed values. \*/

private int[] packedValueStartIndices;

/\*\* PRE-COMPUTED: The shifts and masks used to decode packed values. \*/

private int[] packedValueLowBitsRightShift;

private int[] packedValueMiddleBitsLeftShift;

private int[] packedValueMiddleBitsMask;

private int[] packedValueHighBitsLeftShift;

private int[] packedValueHighBitsMask;

/\*\* Index of packed values. \*/

private int packedValueIndex;

/\*\*

\* The {@link #indexInCurrentBlock} and {@link #startPointerForCurrentBlock} of the first int

\* that holds packed values. This two values together uniquely form a int block pool pointer

\* --- {@link #packedValueStartBlockStart} + {@link #packedValueStartBlockIndex} --- that points

\* to the first int that has pointer.

\*

\* @see #jumpToInt(int, int)

\*/

private int packedValueStartBlockIndex;

private int packedValueStartBlockStart;

/\*\* Current int read from {@link #currentBlock}. \*/

private int currentInt;

/\*\*

\* If given, query cost will be tracked every time a int block is loaded.

\* @see #loadNextBlock()

\*/

private final QueryCostTracker queryCostTracker;

private final QueryCostTracker.CostType queryCostType;

/\*\*

\* Default constructor.

\*

\* @param intBlockPool from which packed ints will be read

\* @param preComputedValues pre-computed shifts, masks, and start int

\* @param queryCostTracker optional, query cost tracker used while loading a new block

\* @param queryCostType optional, query cost type will be tracked while loading a new block

\*/

public IntBlockPoolPackedLongsReader(

IntBlockPool intBlockPool,

PackedLongsReaderPreComputedValues preComputedValues,

@Nullable QueryCostTracker queryCostTracker,

@Nullable QueryCostTracker.CostType queryCostType) {

this.intBlockPool = intBlockPool;

this.preComputedValues = preComputedValues;

// For query cost tracking.

this.queryCostTracker = queryCostTracker;

this.queryCostType = queryCostType;

}

/\*\*

\* Constructor with {@link #queryCostTracker} and {@link #queryCostType} set to null.

\*

\* @param intBlockPool from which packed ints will be read

\* @param preComputedValues pre-computed shifts, masks, and start int

\*/

public IntBlockPoolPackedLongsReader(

IntBlockPool intBlockPool,

PackedLongsReaderPreComputedValues preComputedValues) {

this(intBlockPool, preComputedValues, null, null);

}

/\*\*

\* 1. Set the reader to starting reading at the given int block pool pointer. Correct block will

\* be loaded if the given pointer points to the different block than {@link #currentBlock}.

\* 2. Update shifts, masks, and start int indices based on given number of bits per packed value.

\* 3. Reset packed value sequence start data.

\*

\* @param intBlockPoolPointer points to the int from which this reader will start reading

\* @param bitsPerPackedValue number of bits per packed value.

\*/

public void jumpToInt(int intBlockPoolPointer, int bitsPerPackedValue) {

assert bitsPerPackedValue <= Long.SIZE;

// Update indexInCurrentBlock and load a different index if needed.

int newBlockStart = IntBlockPool.getBlockStart(intBlockPoolPointer);

indexInCurrentBlock = IntBlockPool.getOffsetInBlock(intBlockPoolPointer);

if (startPointerForCurrentBlock != newBlockStart) {

startPointerForCurrentBlock = newBlockStart;

loadNextBlock();

}

// Re-set shifts, masks, and start int indices for the given number bits per packed value.

packedValueNeedsLong = bitsPerPackedValue > Integer.SIZE;

packedValueMask =

bitsPerPackedValue == Long.SIZE ? 0xFFFFFFFFFFFFFFFFL : (1L << bitsPerPackedValue) - 1;

packedValueStartIndices = preComputedValues.getStartIntIndices(bitsPerPackedValue);

packedValueLowBitsRightShift = preComputedValues.getLowBitsRightShift(bitsPerPackedValue);

packedValueMiddleBitsLeftShift = preComputedValues.getMiddleBitsLeftShift(bitsPerPackedValue);

packedValueMiddleBitsMask = preComputedValues.getMiddleBitsMask(bitsPerPackedValue);

packedValueHighBitsLeftShift = preComputedValues.getHighBitsLeftShift(bitsPerPackedValue);

packedValueHighBitsMask = preComputedValues.getHighBitsMask(bitsPerPackedValue);

// Update packed values sequence start data.

packedValueIndex = 0;

packedValueStartBlockIndex = indexInCurrentBlock;

packedValueStartBlockStart = startPointerForCurrentBlock;

// Load an int to prepare for readPackedLong.

loadInt();

}

/\*\*

\* Read next packed value as a long.

\*

\* Caller could cast the returned long to an int if needed.

\* NOTICE! Be careful of overflow while casting a long to an int.

\*

\* @return next packed value in a long.

\*/

public long readPackedLong() {

long packedValue;

if (packedValueNeedsLong) {

packedValue =

(LONG\_MASK & currentInt)

>>> packedValueLowBitsRightShift[packedValueIndex] & packedValueMask;

packedValue |=

(LONG\_MASK & loadInt()

& packedValueMiddleBitsMask[packedValueIndex])

<< packedValueMiddleBitsLeftShift[packedValueIndex];

if (packedValueHighBitsLeftShift[packedValueIndex] != 0) {

packedValue |=

(LONG\_MASK & loadInt()

& packedValueHighBitsMask[packedValueIndex])

<< packedValueHighBitsLeftShift[packedValueIndex];

}

} else {

packedValue =

currentInt >>> packedValueLowBitsRightShift[packedValueIndex] & packedValueMask;

if (packedValueMiddleBitsLeftShift[packedValueIndex] != 0) {

packedValue |=

(loadInt()

& packedValueMiddleBitsMask[packedValueIndex])

<< packedValueMiddleBitsLeftShift[packedValueIndex];

}

}

packedValueIndex++;

return packedValue;

}

/\*\*

\* A simple getter of {@link #packedValueIndex}.

\*/

public int getPackedValueIndex() {

return packedValueIndex;

}

/\*\*

\* A setter of {@link #packedValueIndex}. This setter will also set the correct

\* {@link #indexInCurrentBlock} based on {@link #packedValueStartIndices}.

\*/

public void setPackedValueIndex(int packedValueIndex) {

this.packedValueIndex = packedValueIndex;

this.indexInCurrentBlock =

packedValueStartBlockIndex + packedValueStartIndices[packedValueIndex];

this.startPointerForCurrentBlock = packedValueStartBlockStart;

loadInt();

}

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

\* Private Helper Methods \*

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

/\*\*

\* Load a new int block, specified by {@link #startPointerForCurrentBlock}, from

\* {@link #intBlockPool}. If {@link #queryCostTracker} is given, query cost with type

\* {@link #queryCostType} will be tracked as well.

\*/

private void loadNextBlock() {

if (queryCostTracker != null) {

assert queryCostType != null;

queryCostTracker.track(queryCostType);

}

currentBlock = intBlockPool.getBlock(startPointerForCurrentBlock);

}

/\*\*

\* Load an int from {@link #currentBlock}. The loaded int will be returned as well.

\* If the {@link #currentBlock} is used up, next block will be automatically loaded.

\*/

private int loadInt() {

while (indexInCurrentBlock >= IntBlockPool.BLOCK\_SIZE) {

startPointerForCurrentBlock += IntBlockPool.BLOCK\_SIZE;

loadNextBlock();

indexInCurrentBlock = Math.max(indexInCurrentBlock - IntBlockPool.BLOCK\_SIZE, 0);

}

currentInt = currentBlock[indexInCurrentBlock++];

return currentInt;

}

}