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

import java.util.NoSuchElementException;

import com.google.common.annotations.VisibleForTesting;

/\*\*

\* A posting buffer used by {@link HighDFPackedIntsPostingLists} while copying over posting list.

\*/

final class PostingsBufferQueue {

/\*\*

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

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

\*/

static final long LONG\_MASK = (1L << 32) - 1;

/\*\*

\* A circular FIFO long queue used internally to store posting.

\* @see #postingsQueue

\*/

@VisibleForTesting

static final class Queue {

private final long[] queue;

private int head = 0;

private int tail = 0;

private int size;

Queue(int maxSize) {

this.queue = new long[maxSize < 2 ? 2 : maxSize];

}

boolean isEmpty() {

return size() == 0;

}

boolean isFull() {

return size() == queue.length;

}

void offer(long value) {

if (size() == queue.length) {

throw new IllegalStateException("Queue is full");

}

queue[tail] = value;

tail = (tail + 1) % queue.length;

size++;

}

long poll() {

if (isEmpty()) {

throw new NoSuchElementException("Queue is empty.");

}

long value = queue[head];

head = (head + 1) % queue.length;

size--;

return value;

}

int size() {

return size;

}

}

/\*\*

\* Internal posting queue.

\*/

private final Queue postingsQueue;

/\*\*

\* Constructor with max size.

\*

\* @param maxSize max size of this buffer.

\*/

PostingsBufferQueue(int maxSize) {

this.postingsQueue = new Queue(maxSize);

}

/\*\*

\* Check if the buffer is empty.

\*

\* @return If this buffer is empty

\*/

boolean isEmpty() {

return postingsQueue.isEmpty();

}

/\*\*

\* Check if the buffer is full.

\*

\* @return If this buffer is full

\*/

boolean isFull() {

return postingsQueue.isFull();

}

/\*\*

\* Get the current size of this buffer.

\*

\* @return Current size of this buffer

\*/

int size() {

return postingsQueue.size();

}

/\*\*

\* Store a posting with docID and a second value that could be freq, position, or any additional

\* info. This method will encode the offered doc ID and second value with

\* {@link #encodePosting(int, int)}.

\*

\* @param docID doc ID of the posting

\* @param secondValue an additional value of the posting

\*/

void offer(int docID, int secondValue) {

postingsQueue.offer(encodePosting(docID, secondValue));

}

/\*\*

\* Remove and return the earliest inserted posting, this is a FIFO queue.

\*

\* @return the earliest inserted posting.

\*/

long poll() {

return postingsQueue.poll();

}

/\*\*

\* Encode a doc ID and a second value, both are ints, into a long. The higher 32 bits store the

\* doc ID and lower 32 bits store the second value.

\*

\* @param docID an int specifying doc ID of the posting

\* @param secondValue an int specifying the second value of the posting

\* @return an encoded long represent the posting

\*/

private static long encodePosting(int docID, int secondValue) {

return ((LONG\_MASK & docID) << 32) | (LONG\_MASK & secondValue);

}

/\*\*

\* Decode doc ID from the given posting.

\* @param posting a given posting encoded with {@link #encodePosting(int, int)}

\* @return the doc ID of the given posting.

\*/

static int getDocID(long posting) {

return (int) (posting >> 32);

}

/\*\*

\* Decode the second value from the given posting.

\* @param posting a given posting encoded with {@link #encodePosting(int, int)}

\* @return the second value of the given posting.

\*/

static int getSecondValue(long posting) {

return (int) posting;

}

}