package com.twitter.search.common.relevance.features;

import java.nio.ByteBuffer;

import java.util.Arrays;

import com.google.common.base.Preconditions;

/\*\*

\* A TweetIntegerShingleSignature object consists of 4 bytes, each representing the signature of

\* a status text sample. The signature bytes are sorted in ascending order and compacted to an

\* integer in big endian for serialization.

\*

\* Fuzzy matching of two TweetIntegerShingleSignature objects is met when the number of matching

\* bytes between the two is equal to or above 3.

\*/

public class TweetIntegerShingleSignature {

public static final int NUM\_SHINGLES = Integer.SIZE / Byte.SIZE;

public static final int DEFAULT\_NO\_SIGNATURE = 0;

public static final TweetIntegerShingleSignature NO\_SIGNATURE\_HANDLE =

deserialize(DEFAULT\_NO\_SIGNATURE);

public static final int DEFAULT\_MIN\_SHINGLES\_MATCH = 3;

private final int minShinglesMatch;

private final byte[] shingles;

private final int signature; // redundant information, for easier comparison.

/\*\*

\* Construct from a byte array.

\*/

public TweetIntegerShingleSignature(byte[] shingles, int minShinglesMatch) {

Preconditions.checkArgument(shingles.length == NUM\_SHINGLES);

this.shingles = shingles;

// sort to byte's natural ascending order

Arrays.sort(this.shingles);

this.minShinglesMatch = minShinglesMatch;

this.signature = serializeInternal(shingles);

}

/\*\*

\* Construct from a byte array.

\*/

public TweetIntegerShingleSignature(byte[] shingles) {

this(shingles, DEFAULT\_MIN\_SHINGLES\_MATCH);

}

/\*\*

\* Construct from a serialized integer signature.

\*/

public TweetIntegerShingleSignature(int signature, int minShinglesMatch) {

this.shingles = deserializeInternal(signature);

// sort to byte's natural ascending order

Arrays.sort(this.shingles);

this.minShinglesMatch = minShinglesMatch;

// now store the sorted shingles into signature field, may be different from what passed in.

this.signature = serializeInternal(shingles);

}

/\*\*

\* Construct from a serialized integer signature.

\*/

public TweetIntegerShingleSignature(int signature) {

this(signature, DEFAULT\_MIN\_SHINGLES\_MATCH);

}

/\*\*

\* Used by ingester to generate signature.

\* Raw signatures are in byte arrays per sample, and can be more or less

\* than what is asked for.

\*

\* @param rawSignature

\*/

public TweetIntegerShingleSignature(Iterable<byte[]> rawSignature) {

byte[] condensedSignature = new byte[NUM\_SHINGLES];

int i = 0;

for (byte[] signatureItem : rawSignature) {

condensedSignature[i++] = signatureItem[0];

if (i == NUM\_SHINGLES) {

break;

}

}

this.shingles = condensedSignature;

Arrays.sort(this.shingles);

this.minShinglesMatch = DEFAULT\_MIN\_SHINGLES\_MATCH;

this.signature = serializeInternal(shingles);

}

/\*\*

\* When used in a hashtable for dup detection, take the first byte of each signature for fast

\* pass for majority case of no fuzzy matching. For top queries, this optimization losses about

\* only 4% of all fuzzy matches.

\*

\* @return most significant byte of this signature as its hashcode.

\*/

@Override

public int hashCode() {

return shingles[0] & 0xFF;

}

/\*\*

\* Perform fuzzy matching between two TweetIntegerShingleSignature objects.

\*

\* @param other TweetIntegerShingleSignature object to perform fuzzy match against

\* @return true if at least minMatch number of bytes match

\*/

@Override

public boolean equals(Object other) {

if (this == other) {

return true;

}

if (other == null) {

return false;

}

if (getClass() != other.getClass()) {

return false;

}

final TweetIntegerShingleSignature otherSignatureInteger = (TweetIntegerShingleSignature) other;

int otherSignature = otherSignatureInteger.serialize();

if (signature == otherSignature) {

// Both serialized signature is the same

return true;

} else if (signature != DEFAULT\_NO\_SIGNATURE && otherSignature != DEFAULT\_NO\_SIGNATURE) {

// Neither is NO\_SIGNATURE, need to compare shingles.

byte[] otherShingles = otherSignatureInteger.getShingles();

int numberMatchesNeeded = minShinglesMatch;

// expect bytes are in ascending sorted order

int i = 0;

int j = 0;

while (((numberMatchesNeeded <= (NUM\_SHINGLES - i)) // early termination for i

|| (numberMatchesNeeded <= (NUM\_SHINGLES - j))) // early termination j

&& (i < NUM\_SHINGLES) && (j < NUM\_SHINGLES)) {

if (shingles[i] == otherShingles[j]) {

if (shingles[i] != 0) { // we only consider two shingles equal if they are non zero

numberMatchesNeeded--;

if (numberMatchesNeeded == 0) {

return true;

}

}

i++;

j++;

} else if (shingles[i] < otherShingles[j]) {

i++;

} else if (shingles[i] > otherShingles[j]) {

j++;

}

}

}

// One is NO\_SIGNATURE and one is not.

return false;

}

/\*\*

\* Returns the sorted array of signature bytes.

\*/

public byte[] getShingles() {

return shingles;

}

/\*\*

\* Serialize 4 sorted signature bytes into an integer in big endian order.

\*

\* @return compacted int signature

\*/

private static int serializeInternal(byte[] shingles) {

ByteBuffer byteBuffer = ByteBuffer.allocate(NUM\_SHINGLES);

byteBuffer.put(shingles, 0, NUM\_SHINGLES);

return byteBuffer.getInt(0);

}

/\*\*

\* Deserialize an integer into a 4-byte array.

\* @param signature The signature integer.

\* @return A byte array with 4 elements.

\*/

private static byte[] deserializeInternal(int signature) {

return ByteBuffer.allocate(NUM\_SHINGLES).putInt(signature).array();

}

public int serialize() {

return signature;

}

public static boolean isFuzzyMatch(int signature1, int signature2) {

return TweetIntegerShingleSignature.deserialize(signature1).equals(

TweetIntegerShingleSignature.deserialize(signature2));

}

public static TweetIntegerShingleSignature deserialize(int signature) {

return new TweetIntegerShingleSignature(signature);

}

public static TweetIntegerShingleSignature deserialize(int signature, int minMatchSingles) {

return new TweetIntegerShingleSignature(signature, minMatchSingles);

}

@Override

public String toString() {

return String.format("%d %d %d %d", shingles[0], shingles[1], shingles[2], shingles[3]);

}

}