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

import java.util.List;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

import com.twitter.search.common.indexing.thriftjava.PackedFeatures;

import com.twitter.search.common.schema.base.FeatureConfiguration;

/\*\*

\* Class used to read/write integers encoded according to

\* {@link com.twitter.search.common.schema.base.FeatureConfiguration}

\*

\* Implementations must override {@link #getInt(int pos)} and {@link #setInt(int pos, int value)}.

\*/

public abstract class IntegerEncodedFeatures {

/\*\*

\* Returns the value at the given position.

\*/

public abstract int getInt(int pos);

/\*\*

\* Sets the given value at the given position.

\*/

public abstract void setInt(int pos, int value);

/\*\*

\* Get the maximum number of integers to hold features.

\* @return the number of integers to represent all features.

\*/

public abstract int getNumInts();

/\*\*

\* Test to see if the given feature is true or non-zero. Useful for one bit features.

\* @param feature feature to examine

\* @return true if feature is non-zero

\*/

public boolean isFlagSet(FeatureConfiguration feature) {

return (getInt(feature.getValueIndex()) & feature.getBitMask()) != 0;

}

public IntegerEncodedFeatures setFlag(FeatureConfiguration feature) {

setInt(feature.getValueIndex(), getInt(feature.getValueIndex()) | feature.getBitMask());

return this;

}

public IntegerEncodedFeatures clearFlag(FeatureConfiguration feature) {

setInt(feature.getValueIndex(), getInt(feature.getValueIndex()) & feature.getInverseBitMask());

return this;

}

/\*\*

\* Sets a boolean flag.

\*/

public IntegerEncodedFeatures setFlagValue(FeatureConfiguration feature, boolean value) {

if (value) {

setFlag(feature);

} else {

clearFlag(feature);

}

return this;

}

/\*\*

\* Get feature value

\* @param feature feature to get

\* @return the value of the feature

\*/

public int getFeatureValue(FeatureConfiguration feature) {

return (getInt(feature.getValueIndex()) & feature.getBitMask())

>>> feature.getBitStartPosition();

}

/\*\*

\* Set feature value

\* @param feature feature to modify

\* @param value value to set.

\*/

public IntegerEncodedFeatures setFeatureValue(FeatureConfiguration feature, int value) {

Preconditions.checkState(

value <= feature.getMaxValue(),

"Feature value, %s, is greater than the max value allowed for this feature. "

+ "Feature: %s, Max value: %s",

value, feature.getName(), feature.getMaxValue());

// Clear the value of the given feature in its int.

int temp = getInt(feature.getValueIndex()) & feature.getInverseBitMask();

// Set the new feature value. Applying the bit mask here ensures that other features in the

// same int are not modified by mistake.

temp |= (value << feature.getBitStartPosition()) & feature.getBitMask();

setInt(feature.getValueIndex(), temp);

return this;

}

/\*\*

\* Sets feature value if greater than current value

\* @param feature feature to modify

\* @param value new value

\*/

public IntegerEncodedFeatures setFeatureValueIfGreater(FeatureConfiguration feature, int value) {

if (value > getFeatureValue(feature)) {

setFeatureValue(feature, value);

}

return this;

}

/\*\*

\* Increment a feature if its not at its maximum value.

\* @return whether the feature is incremented.

\*/

public boolean incrementIfNotMaximum(FeatureConfiguration feature) {

int newValue = getFeatureValue(feature) + 1;

if (newValue <= feature.getMaxValue()) {

setFeatureValue(feature, newValue);

return true;

} else {

return false;

}

}

/\*\*

\* Copy these encoded features to a new PackedFeatures thrift struct.

\*/

public PackedFeatures copyToPackedFeatures() {

return copyToPackedFeatures(new PackedFeatures());

}

/\*\*

\* Copy these encoded features to a PackedFeatures thrift struct.

\*/

public PackedFeatures copyToPackedFeatures(PackedFeatures packedFeatures) {

Preconditions.checkNotNull(packedFeatures);

final List<Integer> integers = Lists.newArrayListWithCapacity(getNumInts());

for (int i = 0; i < getNumInts(); i++) {

integers.add(getInt(i));

}

packedFeatures.setDeprecated\_featureConfigurationVersion(0);

packedFeatures.setFeatures(integers);

return packedFeatures;

}

/\*\*

\* Copy features from a packed features struct.

\*/

public void readFromPackedFeatures(PackedFeatures packedFeatures) {

Preconditions.checkNotNull(packedFeatures);

List<Integer> ints = packedFeatures.getFeatures();

for (int i = 0; i < getNumInts(); i++) {

if (i < ints.size()) {

setInt(i, ints.get(i));

} else {

setInt(i, 0);

}

}

}

}