001/\*  
002 \* Licensed to the Apache Software Foundation (ASF) under one or more  
003 \* contributor license agreements. See the NOTICE file distributed with  
004 \* this work for additional information regarding copyright ownership.  
005 \* The ASF licenses this file to You under the Apache License, Version 2.0  
006 \* (the "License"); you may not use this file except in compliance with  
007 \* the License. You may obtain a copy of the License at  
008 \*  
009 \* http://www.apache.org/licenses/LICENSE-2.0  
010 \*  
011 \* Unless required by applicable law or agreed to in writing, software  
012 \* distributed under the License is distributed on an "AS IS" BASIS,  
013 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
014 \* See the License for the specific language governing permissions and  
015 \* limitations under the License.  
016 \*/  
017package org.apache.commons.collections4.trie.analyzer;  
018  
019import org.apache.commons.collections4.trie.KeyAnalyzer;  
020  
021/\*\*  
022 \* An {@link KeyAnalyzer} for {@link String}s.  
023 \*  
024 \* @since 4.0  
025 \*/  
026public class StringKeyAnalyzer extends KeyAnalyzer<String> {  
027  
028 private static final long serialVersionUID = -7032449491269434877L;  
029  
030 /\*\* A singleton instance of {@link StringKeyAnalyzer}. \*/  
031 public static final StringKeyAnalyzer INSTANCE = new StringKeyAnalyzer();  
032  
033 /\*\* The number of bits per {@link Character}. \*/  
034 public static final int LENGTH = Character.SIZE;  
035  
036 /\*\* A bit mask where the first bit is 1 and the others are zero. \*/  
037 private static final int MSB = 0x8000;  
038  
039 /\*\* Returns a bit mask where the given bit is set. \*/  
040 private static int mask(final int bit) {  
041 return MSB >>> bit;  
042 }  
043  
044 @Override  
045 public int bitsPerElement() {  
046 return LENGTH;  
047 }  
048  
049 @Override  
050 public int lengthInBits(final String key) {  
051 return key != null ? key.length() \* LENGTH : 0;  
052 }  
053  
054 @Override  
055 public int bitIndex(final String key, final int offsetInBits, final int lengthInBits,  
056 final String other, final int otherOffsetInBits, final int otherLengthInBits) {  
057  
058 boolean allNull = true;  
059  
060 if (offsetInBits % LENGTH != 0 || otherOffsetInBits % LENGTH != 0  
061 || lengthInBits % LENGTH != 0 || otherLengthInBits % LENGTH != 0) {  
062 throw new IllegalArgumentException("The offsets and lengths must be at Character boundaries");  
063 }  
064  
065 final int beginIndex1 = offsetInBits / LENGTH;  
066 final int beginIndex2 = otherOffsetInBits / LENGTH;  
067  
068 final int endIndex1 = beginIndex1 + lengthInBits / LENGTH;  
069 final int endIndex2 = beginIndex2 + otherLengthInBits / LENGTH;  
070  
071 final int length = Math.max(endIndex1, endIndex2);  
072  
073 // Look at each character, and if they're different  
074 // then figure out which bit makes the difference  
075 // and return it.  
076 char k = 0, f = 0;  
077 for(int i = 0; i < length; i++) {  
078 final int index1 = beginIndex1 + i;  
079 final int index2 = beginIndex2 + i;  
080  
081 if (index1 >= endIndex1) {  
082 k = 0;  
083 } else {  
084 k = key.charAt(index1);  
085 }  
086  
087 if (other == null || index2 >= endIndex2) {  
088 f = 0;  
089 } else {  
090 f = other.charAt(index2);  
091 }  
092  
093 if (k != f) {  
094 final int x = k ^ f;  
095 return i \* LENGTH + Integer.numberOfLeadingZeros(x) - LENGTH;  
096 }  
097  
098 if (k != 0) {  
099 allNull = false;  
100 }  
101 }  
102  
103 // All bits are 0  
104 if (allNull) {  
105 return KeyAnalyzer.NULL\_BIT\_KEY;  
106 }  
107  
108 // Both keys are equal  
109 return KeyAnalyzer.EQUAL\_BIT\_KEY;  
110 }  
111  
112 @Override  
113 public boolean isBitSet(final String key, final int bitIndex, final int lengthInBits) {  
114 if (key == null || bitIndex >= lengthInBits) {  
115 return false;  
116 }  
117  
118 final int index = bitIndex / LENGTH;  
119 final int bit = bitIndex % LENGTH;  
120  
121 return (key.charAt(index) & mask(bit)) != 0;  
122 }  
123  
124 @Override  
125 public boolean isPrefix(final String prefix, final int offsetInBits,  
126 final int lengthInBits, final String key) {  
127 if (offsetInBits % LENGTH != 0 || lengthInBits % LENGTH != 0) {  
128 throw new IllegalArgumentException(  
129 "Cannot determine prefix outside of Character boundaries");  
130 }  
131  
132 final String s1 = prefix.substring(offsetInBits / LENGTH, lengthInBits / LENGTH);  
133 return key.startsWith(s1);  
134 }  
135}