package com.twitter.recos.user\_user\_graph

import com.twitter.graphjet.bipartite.api.EdgeTypeMask

import com.twitter.recos.recos\_common.thriftscala.UserSocialProofType

/\*\*

\* The bit mask is used to encode edge types in the top bits of an integer,

\* e.g. Follow, Mention, and Mediatag. Under current segment configuration, each segment

\* stores up to 128M edges. Assuming that each node on one side is unique, each segment

\* stores up to 128M unique nodes on one side, which occupies the lower 27 bits of an integer.

\* This leaves five bits to encode the edge types, which at max can store 32 edge types.

\* The following implementation utilizes the top four bits and leaves one free bit out.

\*/

class UserEdgeTypeMask extends EdgeTypeMask {

import UserEdgeTypeMask.\_

override def encode(node: Int, edgeType: Byte): Int = {

require(

edgeType == FOLLOW || edgeType == MENTION || edgeType == MEDIATAG,

s"encode: Illegal edge type argument $edgeType")

node | EDGEARRAY(edgeType)

}

override def edgeType(node: Int): Byte = {

(node >> 28).toByte

}

override def restore(node: Int): Int = {

node & MASK

}

}

object UserEdgeTypeMask {

/\*\*

\* Reserve the top four bits of each integer to encode the edge type information.

\*/

val MASK: Int =

Integer.parseInt("00001111111111111111111111111111", 2)

val FOLLOW: Byte = 0

val MENTION: Byte = 1

val MEDIATAG: Byte = 2

val SIZE: Byte = 3

val UNUSED3: Byte = 3

val UNUSED4: Byte = 4

val UNUSED5: Byte = 5

val UNUSED6: Byte = 6

val UNUSED7: Byte = 7

val UNUSED8: Byte = 8

val UNUSED9: Byte = 9

val UNUSED10: Byte = 10

val UNUSED11: Byte = 11

val UNUSED12: Byte = 12

val UNUSED13: Byte = 13

val UNUSED14: Byte = 14

val UNUSED15: Byte = 15

val EDGEARRAY: Array[Int] = Array(

0,

1 << 28,

2 << 28,

3 << 28,

4 << 28,

5 << 28,

6 << 28,

7 << 28,

8 << 28,

9 << 28,

10 << 28,

11 << 28,

12 << 28,

13 << 28,

14 << 28,

15 << 28

)

/\*\*

\* Map valid social proof types specified by clients to an array of bytes. If clients do not

\* specify any social proof types in thrift, it will return all available social types by

\* default.

\*

\* @param socialProofTypes are the valid socialProofTypes specified by clients

\* @return an array of bytes representing valid social proof types

\*/

def getUserUserGraphSocialProofTypes(

socialProofTypes: Option[Seq[UserSocialProofType]]

): Array[Byte] = {

socialProofTypes

.map { \_.map { \_.getValue }.toArray }

.getOrElse((0 until SIZE).toArray)

.map { \_.toByte }

}

}