**Algorithmique / Travaux pratiques**

S12 – Ensemble d’entiers et BitSet





07.05.2015

Adriano De Almeida Silva – T-1f

Alex Travasso – T-1f

1. **Voici ci-dessous le type abstrait « ensemble d’entiers »**

*BitSetOfShorts*

package s12;

import java.util.BitSet;

public class BitSetOfShorts {

BitSet bs;

static final short LOW = Short.MIN\_VALUE;

static final short HIGH = Short.MAX\_VALUE;

// ------------------------------------------------------------

static int indexFromElt(short e) {

if (e < 0) {

int val = Math.abs(e);

return Math.abs((val \* 2) - 1);

} else {

return e \* 2;

}

}

static short eltFromIndex(int i) {

if (i % 2 == 0) {

return (short) (i / 2);

} else {

return (short) (((i + 1) / 2) \* -1);

}

}

// ------------------------------------------------------------

public BitSetOfShorts() {

bs = new BitSet(); // or: new BitSet(1 + HIGH - LOW);

}

// ------------------------------------------------------------

public void add(short e) {

bs.set(indexFromElt(e), true);

}

public void remove(short e) {

bs.set(indexFromElt(e), false);

}

public boolean contains(short e) {

return bs.get(indexFromElt(e));

}

// ------------------------------------------------------------

public void union(BitSetOfShorts s) {

BitSetOfShortsItr itr = new BitSetOfShortsItr(s);

while (itr.hasMoreElements()) {

add(itr.nextElement());

}

}

public void intersection(BitSetOfShorts s) {

BitSetOfShortsItr itr = new BitSetOfShortsItr(this);

while (itr.hasMoreElements()) {

short e = itr.nextElement();

if (!s.contains(e)) {

remove(e);

}

}

}

// ------------------------------------------------------------

public int size() {

return bs.cardinality();

}

// ------------------------------------------------------------

public boolean isEmpty() {

return bs.length() == 0;

}

// ------------------------------------------------------------

public String toString() {

String r = "{";

BitSetOfShortsItr itr = new BitSetOfShortsItr(this);

if (isEmpty())

return "{}";

r += itr.nextElement();

while (itr.hasMoreElements()) {

r += ", " + itr.nextElement();

}

return r + "}";

}

// ------------------------------------------------------------

public static void main(String[] args) {

BitSetOfShorts a = new BitSetOfShorts();

BitSetOfShorts b = new BitSetOfShorts();

short[] ta = { -3, 5, 6, -3, 9, 9 };

short[] tb = { 6, 7, -2, -3 };

int i;

for (i = 0; i < ta.length; i++) {

a.add(ta[i]);

System.out.println("" + a + a.size());

}

for (i = 0; i < tb.length; i++) {

b.add(tb[i]);

System.out.println("" + b + b.size());

}

a.union(b);

System.out.println("" + a + a.size());

}

}

*BitSetOfShortsItr :*

package s12;

public class BitSetOfShortsItr {

BitSetOfShorts b;

int index = 0;

int cpt = 0;

// ------------------------------------------------------------

public BitSetOfShortsItr(BitSetOfShorts theSet) {

b = theSet;

cpt = theSet.size();

}

public boolean hasMoreElements() {

return cpt > 0;

}

public short nextElement() {

while (!b.contains((short) index)) {

index++;

}

short nextEl = (short) index;

cpt--;

index++;

return nextEl;

}

}

# Voici ci-dessous la classe BitSet[[1]](#endnote-1)

package s12;

import java.nio.ByteBuffer;

public class MyBitSet {

// ------------------------------------------------------------

private int[] buffer;

private static final short NB\_OF\_BITS = 32;

// ------------------------------------------------------------

public MyBitSet() {

this(100);

}

// ------------------------------------------------------------

public MyBitSet(int capacity) {

buffer = new int[1 + (capacity / NB\_OF\_BITS)];

}

// ------------------------------------------------------------

public void set(int bitIndex, boolean value) { // bitIndex >= 0

int tabNbr = bitIndex / NB\_OF\_BITS;

int bitIndexCel = bitIndex % NB\_OF\_BITS;

int tmp = 1 << bitIndexCel;

checkSize(tabNbr);

if (value) {

buffer[tabNbr] = buffer[tabNbr] | tmp;

} else {

buffer[tabNbr] = buffer[tabNbr] & (~tmp);

}

System.out.println(buffer[tabNbr]);

}

// ------------------------------------------------------------

public void set(int bitIndex) {

set(bitIndex, true);

}

// ------------------------------------------------------------

public void clear(int bitIndex) {

set(bitIndex, false);

}

// ------------------------------------------------------------

// ------------------------------------------------------------

// ensures that that array cell exists

// (re-dimensions the array if necessary)

private void checkSize(int arrayIndex) {

if (arrayIndex < buffer.length)

return;

int f = 1 + arrayIndex / buffer.length;

int[] aux = new int[f \* buffer.length]; // or new int[arrayIndex+1] if

for (int j = 0; j < buffer.length; j++)

// we choose the minimal size

aux[j] = buffer[j];

buffer = aux;

assert arrayIndex < buffer.length;

}

// ------------------------------------------------------------

public boolean get(int bitIndex) {

int tabNbr = bitIndex / NB\_OF\_BITS;

int bitIndexCel = bitIndex % NB\_OF\_BITS;

int tmp = 1 << bitIndexCel;

checkSize(tabNbr);

if ((buffer[tabNbr] & tmp) != 0) {

return true;

}

return false; }

// ------------------------------------------------------------

public void and(MyBitSet o) {

o.checkSize(buffer.length - 1);

for (int i = 0; i < buffer.length; i++) {

buffer[i] = buffer[i] & o.buffer[i];

}

}

// ------------------------------------------------------------

public void or(MyBitSet o) {

o.checkSize(buffer.length - 1);

for (int i = 0; i < buffer.length; i++) {

buffer[i] = buffer[i] | o.buffer[i];

}

}

// ------------------------------------------------------------

public void xor(MyBitSet o) {

o.checkSize(buffer.length - 1);

for (int i = 0; i < buffer.length; i++) {

buffer[i] = buffer[i] ^ o.buffer[i];

}

}

// ------------------------------------------------------------

public int size() { // crt capacity, total nb of bits

return buffer.length \* NB\_OF\_BITS;

}

// ------------------------------------------------------------

public int length() { // highest bit "on" + 1

int result = 0;

for (int i = 0; i < buffer.length \* NB\_OF\_BITS; i++)

if (get(i))

result = i + 1;

return result;

}

// ------------------------------------------------------------

public int nextSetBit(int fromIndex) { // -1 if none

for (int i = fromIndex; i < buffer.length \* NB\_OF\_BITS; i++) {

if (get(i))

return i;

}

return -1;

}

// ------------------------------------------------------------

public int cardinality() { // nb of bits set to true

int compteur = 0;

for (int i = 0; i < buffer.length \* NB\_OF\_BITS; i++) {

if (get(i)) {

compteur++;

}

}

return compteur;

}

// ------------------------------------------------------------

public String toString() {

String r = "{";

for (int i = 0; i < buffer.length \* NB\_OF\_BITS; i++)

if (get(i))

if (r.length() == 1)

r += i;

else

r += "," + i;

return r + "}";

}

// ------------------------------------------------------------

public static void main(String[] args) {

MyBitSet a = new MyBitSet(100);

ok(a.length() == 0);

System.out.println(a);

a.set(4);

ok(a.get(4));

ok(!a.get(3));

a.clear(4);

a.clear(5);

a.set(6);

ok(!a.get(4));

ok(a.get(6));

ok(!a.get(5));

System.out.println(a);

}

// ------------------------------------------------------------

static void ok(boolean b) {

if (b)

return;

throw new RuntimeException("property not verified: ");

}

}

On pourrait réaliser l’opération cardinality en O(1) en stockant les positions des valeurs à true. Ce n’est pas une bonne idée car cela prendra beaucoup de mémoire pour une petite fonction.

1. Possibles similitudes avec le travail de Micael Domingues Alves dû à son aide pour la réalisation de cet exercice [↑](#endnote-ref-1)