GED : Compression Huffman

Mustapha SAHLI, 2 ING GLSI 1

Tests des différentes fonctionnalités de compression en utilisant l'algorithme de Huffman

Institut Supérieur d'Informatique – A.U. : 2016 / 2017

Arbre et Code de Huffman

La théorie de l'information fournit une solution optimale pour construire un tel code. Cette solution utilise un arbre binaire complet pour représenter le code. Dans cet arbre, les chemins de la racine vers les feuilles fournissent le codage des lettres de l'alphabet placées dans les feuilles.

```java
/**
 * test d'affichage des arbres [ Question-4 ]
 */
public static void testAffichageArbres() {
    System.out.println( "Arbre 1" );
    Noeud arbre1 = new Interne(
            new Interne(
                    new Interne(
                            new Feuille( 2, 'd' ),
                            new Feuille( 2, 'c' ) ),
                    new Feuille( 4, 'b' ) ),
            new Feuille( 8, 'a' ) );
    arbre1.afficher();
    System.out.println( arbre1 );

    System.out.println();

    System.out.println( "Arbre 2" );
    Noeud arbre2 = new Interne(
            new Interne(
                    new Feuille( 2, 'd' ),
                    new Feuille( 2, 'c' ) ),
            new Interne(
                    new Feuille( 4, 'b' ),
                    new Feuille( 8, 'a' ) ) );
    arbre2.afficher();
    System.out.println( arbre2 );
}
```

```
Arbre 1
 0: 'a'
 0
  0: 'b'
  1
   0: 'c'
   1
   1: 'd'
[[[<d, 2>,<c, 2>],<b, 4>],<a, 8>]

Arbre 2
  0: 'a'
  0
  1: 'b'
 0
  0: 'c'
  1
  1: 'd'
[[<d, 2>,<c, 2>],[<b, 4>,<a, 8>]]
```

```java
/**
 * test de génération des codes [ Question-5 ]
 */
public static void testGenerationCodes() {
    Noeud arbre1 = new Interne(
            new Interne(
                    new Interne(
```

```
                                new Feuille( 2, 'd' ),
                                new Feuille( 2, 'c' ) ),
                        new Feuille( 4, 'b' ) ),
                new Feuille( 8, 'a' ) );
        HashMap<Character, String> codes = new HashMap<>();
        System.out.println( codes );
        arbre1.remplirTable( codes );
        System.out.println( codes );

        codes.clear();

        Noeud arbre2 = new Interne(
                new Interne(
                        new Feuille( 2, 'd' ),
                        new Feuille( 2, 'c' ) ),
                new Interne(
                        new Feuille( 4, 'b' ),
                        new Feuille( 8, 'a' ) ) );
        System.out.println( codes );
        arbre2.remplirTable( codes );
        System.out.println( codes );
    }
```

```
{}
{a=0, b=10, c=110, d=111}
{}
{a=00, b=01, c=10, d=11}
```

Codage et Décodage

Dans cette partie nous allons voir comment utiliser les arbres et les tables de codes que nous venons de définir pour coder et décoder une chaîne de caractères.

```
    /**
     * test de codage [ Question-7 ]
     */
    public static void testCodage() {
        Noeud arbre1 = new Interne(
                new Interne(
                        new Interne(
                                new Feuille( 2, 'd' ),
                                new Feuille( 2, 'c' ) ),
                        new Feuille( 4, 'b' ) ),
                new Feuille( 8, 'a' ) );
        HashMap<Character, String> codes = new HashMap<>();
        System.out.println( codes );
        arbre1.remplirTable( codes );
        System.out.println( codes );

        String texte = "aaaabbcdaaaabbcd";
        String resultatAttendu = "00001010110111000010101101111";
        StringBuffer resultat = new StringBuffer( "" );
        Huffman.encodeTexte( texte, codes, resultat );
        System.out.println( "R  = " + resultat );
        System.out.println( "RA = " + resultatAttendu );

        if ( resultat.toString().equals( resultatAttendu ) ) {
            System.out.println( "Correct!" );
        } else {
            System.out.println( "Erreur!" );
        }
    }
```

```
{}
{a=0, b=10, c=110, d=111}
R  = 00001010110111100001010110111
RA = 00001010110111100001010110111
Correct!
```

Construction du Code

```java
/**
 * test de compression / décompression [ Question-19 ]
 */
public static void testCompressionDecompression() throws IOException {
    // Encodage
    String texte = Copy.readFile( "in.txt" );
    System.out.println( "texte : " + texte );
    StringBuffer texteApresCodage = new StringBuffer( "" );
    Huffman.encode( texte, texteApresCodage );
    String texteApresCodageStr = texteApresCodage.toString();
    Copy.writeInFile( "out.txt", texteApresCodageStr ); // Sauvegarde du texte codé dans un fichier
    System.out.println( "texteApresCodage : " + texteApresCodageStr );
    System.out.println( "arbre : " + Huffman.arbreCodes );

    // Sauvegarde de l'arbre de codage dans un fichier
    StringBuffer buffer = new StringBuffer();
    Huffman.arbreCodes.encode( buffer );
    System.out.println( "arbre codé : " + buffer.toString() );
    Copy.writeInFile( "arbre.txt", buffer.toString() );

    // Décodage
    String chaineCode = Copy.readFile( "out.txt" );
    Etat e = new Etat( chaineCode );
    Noeud a = Huffman.decodeArbre( new Etat( Copy.readFile( "arbre.txt" ) ) );
    System.out.println( "arbre décodé : " + a );
    String chaineDecode = Huffman.decodeTexte( a, e );
    System.out.println( "chaîne décodée : " + chaineDecode );
    Copy.writeInFile( "out2.txt", chaineDecode );

    double nbBitsAvantCodage = chaineDecode.length() * 8; // chaque caractére est codé sur 8 bits

    double nbBitsApresCodage = chaineCode.length();
    double tauxCompression = nbBitsApresCodage / nbBitsAvantCodage * 100;
    System.out.println( "taux de compression : " + tauxCompression + "%" );
}
```

```
texte : lorem ipsum dolor sit amet
texteApresCodage : 0100101000101011000010000011001111100100001110110101001010001001111000001110010110110001010111
arbre : [[[<s, 2>,[<d, 1>,<u, 1>]],[<o, 3>,<m, 3>]],[[[<t, 2>,[<a, 1>,<p, 1>]],[<e, 2>,<l, 2>]],[< , 4>,[<r, 2>,<i, 2>]]]]
arbre codé : 000101110011010110010010111010101011011111101101100010111010001011000011011100000101100101101101100010010000001011100101011010001
arbre décodé : [[[<s, 0>,[<d, 0>,<u, 0>]],[<o, 0>,<m, 0>]],[[[<t, 0>,[<a, 0>,<p, 0>]],[<e, 0>,<l, 0>]],[< , 0>,[<r, 0>,<i, 0>]]]]
chaîne décodée : lorem ipsum dolor sit amet
taux de compression : 45.192307692307695%
```

Compression de Dictionnaire

```java
/**
 * test du dictionnaire [ Question-24 et Question-25 ]
 */
public static void testDictionnaire() throws IOException {
    NoeudDico dictionnaire = null;
```

```
6.
7.      String[] mots = Copy.getWords( "words.txt" );
8.      for ( String mot : mots ) {
9.          if ( dictionnaire == null ) {
10.             dictionnaire = new NoeudDico( mot );
11.         } else {
12.             dictionnaire.ajouter( mot );
13.         }
14.     }
15.     dictionnaire.afficher();
16. }
```

```
underslung(0)
undramatic(0)
uniatism(0)
untrinitarian(0)
untune(0)
user(0)
usuriously(0)
uvula(0)
vulned(0)
wastage(0)
```

```
1.  /**
2.   * test codage / décodage du dictionnaire [ Question-26 ]
3.   */
4.  public static void testCodageDecodageDictionnaire() throws IOException, TasVideException {
5.      NoeudDico dictionnaire = null;
6.
7.      String[] mots = Copy.getWords( "words.txt" );
8.      for ( String mot : mots ) {
9.          if ( dictionnaire == null ) {
10.             dictionnaire = new NoeudDico( mot );
11.         } else {
12.             dictionnaire.ajouter( mot );
13.         }
14.     }
15.
16.     System.out.println( "dictionnaire : " );
17.     dictionnaire.afficher();
18.
19.     StringBuffer buffer = new StringBuffer();
20.     dictionnaire.encode( buffer );
21.     System.out.println( "dictionnaire codé : " + buffer.toString() );
22.
23.     NoeudDico dictionnaireDecode = NoeudDico.decode( buffer.toString() );
24.     System.out.println( "dictionnaire décodé : " );
25.     dictionnaireDecode.afficher();
26. }
```

```
lettres : {a=<a, 41>, b=<b, 6>, c=<c, 18>, d=<d, 16>, e=<e, 63>, f=<f, 9>, g=<g, 16>, h=<h, 11>, i=<i, 49>, j=<j, 3>, k=<k, 4>, l=<l, 23>, m=<m, 10>,
n=<n, 40>, o=<o, 20>, p=<p, 9>, q=<q, 2>, r=<r, 35>, s=<s, 33>, t=<t, 33>, u=<u, 20>, v=<v, 7>, w=<w, 3>, x=<x, 1>, y=<y, 6>, z=<z, 1>}
arbre de codage : [[[<a, 41>,[[<h, 11>,[<b, 6>,[<j, 3>,<w, 3>]]],<l, 23>]],[<i, 49>,[[[<y, 6>,<v, 7>],<d, 16>],<s, 33>]]],[[<e, 63>,[<t, 33>,[<g,
16>,[[<k, 4>,[<q, 2>,[<x, 1>,<z, 1>]]],<p, 9>]]]],[[<r, 35>,[<c, 18>,[<f, 9>,<m, 10>]]],[<n, 40>,[<u, 20>,<o, 20>]]]]]
arbre codé :
000101100001001011010000101100010010110101010111011110110110001011010010001011110011011101101011001001011100110010110010101011101000101100111001011010
110101110001010111100010111101010111000000101110010010110001101011001101011011010101101110010111010110110111 1;
codes : {a=111, b=110101, c=00101, d=10010, e=011, f=001001, g=01001, h=11011, i=101, j=1101001, k=0100011, l=1100, m=001000, n=0001, o=00000,
p=010000, q=01000101, r=0011, s=1000, t=0101, u=00001, v=100110, w=1101000, x=010001001, y=100111, z=010001000}
dictionnaire codé :
000101100001001011010000101100010010110101010111011110110110001011010010001011110011011101101011001001011100110010110010101011101000101100111001011010
1101011100010101111000101111010101110000001011100100101100011010110011010110110101011011100101110101101101111;01101001000001000110111001011111110010001
101010001100000001001101111101100011100001010010010000000101011100100000011010010111111111101010010101101111100000101101010001101010001010111001010011 01
001011111111111000100001110001010110110101000010011011000101001000110011111111111010100010000011010010000010011010000100011111111100110000100101000000 0
001111111000000000100100100000001100010101000000010001111111110000100000000001011100010010001001010100001010100101010011111111110110101000110010111 0
001010110110101000010010011111111110110110101001100011011100101010100010101110110011111111110010000011101000010000011000010010010011000111111111110100
001110000100100100000101010000100111111111110010100110010000011100100000001100011111110101000100101010111011010101100001111111111100000000000010010101
101000001101110001101010000100100111111111101001100011000110100000010000000000000000101011011111111111110010000001100011010100100000110110001101110100
000111111111001000000110010100000101110000100010100111111111110101011111000110011010010001101000000010000101000010011010010111111101010100000100100000
101010100001010000100100111111111101010000101001100011000010010101010000000000111111111011000101011101101010101011000101001010100111111111111001000 0
001100001001100101000110111001010101000000000011111111111100000000101001100010100101000110011010000100011111111101000010011100010101101100000011100
001011101100010011101000010101000111111111111110000010001001000100101101010111011000110011111111100100010100000101010110001100000100101010100001101100 11
010001111111100001001000101000001001100010010101010010111111111110001101110000101000011100010100100011111111001101001001010100000010100000000100110001 10
001111111111001001001100001001100011011100101001101001011111111101000001100100100011001100100101110010101010000000000111111111110000010110101011010010
011000101001011111111110100001110001100010100000000010101110001100010101010000100000000010000111010001111111111111110001010001100110110100001001000011 0
101010011000110001111111111110011000101001100011000010001000001100001001011111111101100011010010000010010001111110001000010100101001100100010010010100
000100011011101100111111111111011011001100001010010111100000000000100101011011001101110100000101001100011111111111100100000101000100100010010101001101 0
000010111111111001010001100110000100100100101011011001100001010100001001001011000100111111111111111111001010111010000100000110110000110100101111111001
101100001100100100110000101010001011111111110000010000101110110001100001100100100000000011010100100010000011010010111111111110100100011000110100100000
001101000011000000001000010010011111111100011011100010000111001010101000101111111101010111001010101010000000100011111100101000110101000010101001010111 00
011010101110000111111111110000010000100111111101000001100011110000010001101010000000000001010000110001001111111111110100110000000101100011111111101001100
00001011000000100110100101111110110100001110100000101011100100100111111111
```

Huffman sur les Mots

```java
/**
 * test : numérotation du dictionnaire [ Question-27 ]
 */
public static void testNumerotationArbre() throws IOException {
    NoeudDico dictionnaire = null;

    String[] mots = Copy.getWords( "test.txt" );
    for ( String mot : mots ) {
        if ( dictionnaire == null ) {
            dictionnaire = new NoeudDico( mot );
        } else {
            dictionnaire.ajouter( mot );
        }
    }
    dictionnaire.numeroter();
    dictionnaire.afficher();
}
```

```
autem(5)
conferebamus(6)
cotidieque(7)
cum(8)
ea(9)
ego(10)
eos(11)
epicuri(12)
erat(13)
etiam(14)
inquam(15)
intellegerem(16)
inter(17)
mentitum(18)
mihi(19)
miraretur(20)
nominavi(21)
nostro(22)
omnes(23)
phaedrum(24)
praeter(25)
probarem(26)
putas(27)
quid(28)
quorum(29)
quos(30)
sane(31)
satis(32)
sedulitatem(33)
sententiae(34)
sunt(35)
tu(36)
umquam(37)
utrumque(38)
zenonem(39)
```

```java
/**
 * test de l'algorithme de Huffman appliqué sur les mots
 * @throws IOException
 * @throws TasVideException
 */
public static void testHuffmanMots() throws IOException, TasVideException {
    String[] mots = Copy.getWords( "test2.txt" );
    HashMap<String, Integer> map = new HashMap<>();
    for ( String mot : mots ) {
        if ( map.containsKey( mot ) ) { // le mot existe déja
            Integer nbOcc = map.get( mot );
            map.put( mot, nbOcc + 1 );
        } else {
            map.put( mot, new Integer( 1 ) ); // 1ére apparition
        }
    }
    System.out.println( map );
    // à ce niveau on a la liste des mots avec leur occurrences
    // on doit les ajouter dans le dictionnaire pour les numéroter
    TasMots tasMots = new TasMots();
    Iterator<Entry<String, Integer>> it = map.entrySet().iterator();
    while ( it.hasNext() ) {
        Entry<String, Integer> pair = (Entry<String, Integer>) it.next();
        FeuilleMot feuilleMot = new FeuilleMot( pair.getKey(), pair.getValue() );
        tasMots.ajouter( feuilleMot );
    }
    NoeudMot arbreCodage;
    if ( tasMots.singleton() ) { // cas des chaînes qui ne contiennent qu'une
```

```
        // seule lettre comme aaaaaaaaaa
        NoeudMot nSingleton = tasMots.retirer();
        arbreCodage = new InterneMot( nSingleton, nSingleton );
    }
    while ( !tasMots.singleton() ) {
        NoeudMot nx = tasMots.retirer();
        NoeudMot ny = tasMots.retirer();
        NoeudMot nn = new InterneMot( nx, ny );
        tasMots.ajouter( nn );
    }
    arbreCodage = tasMots.retirer();

    System.out.println( arbreCodage );

    HashMap<String, String> codes = new HashMap<>();
    arbreCodage.remplirTable( codes );

    System.out.println( codes );
}
```

```
{lorem=1, dolor=1, set=1, test=4, amet=1, ipsum=1}
[<test, 4>,[[<amet, 1>,<ipsum, 1>],[<set, 1>,[<lorem, 1>,<dolor, 1>]]]]
{dolor=0000, lorem=0001, set=001, test=1, amet=011, ipsum=010}
```

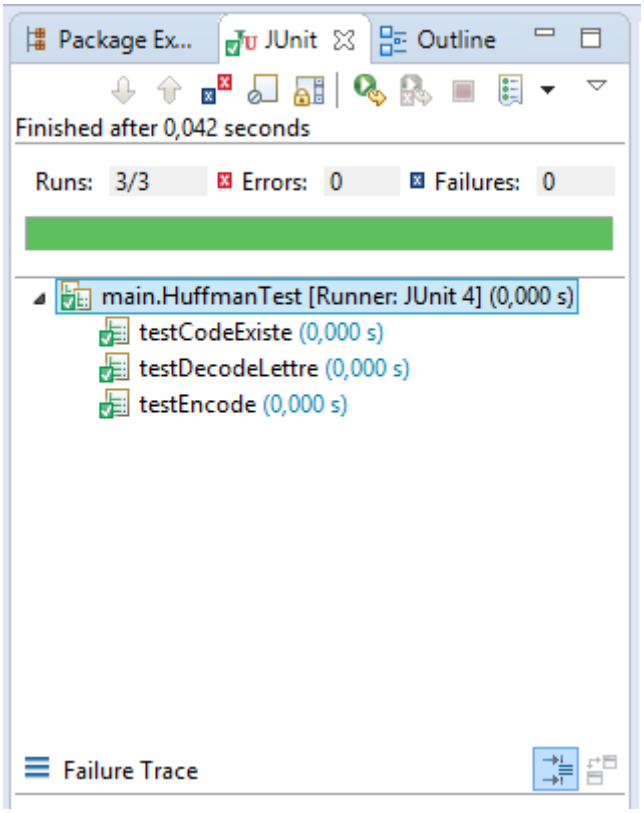
```
/**
 * test décodage de l'arbre
 */
public static void testDecodeArbre() {
    Noeud arbre = Huffman.decodeArbre( new Etat(
            "000101100101010010000010010011100101101111101110100010111000110110000100010110110110110111001011101011011001100101101001010111011001011100101011011 00" ) );
    System.out.println( arbre );
    String texteApresCodage = "000100100011111101000001100001011001001011000010111100010100011001 0101111";
    String texteOriginal = Huffman.decodeTexte( arbre, new Etat( texteApresCodage ) );
    System.out.println( "texteOriginal: " + texteOriginal );
}
```

```
[[[<e, 0>,[< , 0>,<', 0>]],[[<o, 0>,<t, 0>],[<q, 0>,<a, 0>]]],[[[<m, 0>,<n, 0>],[<u, 0>,<f, 0>]],[<i, 0>,[<v, 0>,[<r, 0>,<l, 0>]]]]]
texteOriginal: vive l'informatique
```

Tests Unitaires des méthodes de la classe Huffman

```
package main;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import java.util.HashMap;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import main.arbreDeCodes.Feuille;
import main.arbreDeCodes.Interne;
import main.arbreDeCodes.Noeud;

/**
```

```java
 * classe permettant de tester la classe Huffman
 * @author Stoufa
 *
 */
public class HuffmanTest {

    Noeud                       arbre1           = new Interne(
            new Interne(
                    new Interne(
                            new Feuille( 2, 'd' ),
                            new Feuille( 2, 'c' ) ),
                    new Feuille( 4, 'b' ) ),
            new Feuille( 8, 'a' ) );
    String                      texte            = "aaaabbcdaaaabbcd";
    String                      texteApresCodage = "00001010110111000010101101111";
    HashMap<Character, String> codes             = new HashMap<>();
    Etat                        e1, e2, e3, e4;

    boolean                     initialized      = false;

    public void init() {
        if ( !initialized ) {
            initialized = true;
            arbre1.remplirTable( codes );
            e1 = new Etat( "0" );
            e2 = new Etat( "10" );
            e3 = new Etat( "110" );
            e4 = new Etat( "111" );
        }
    }

    @Before
    public void setUp() throws Exception {
        init();
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testDecodeLettre() {
        assertEquals( "a", Huffman.decodeLettre( codes, e1 ) );
        assertEquals( "b", Huffman.decodeLettre( codes, e2 ) );
        assertEquals( "c", Huffman.decodeLettre( codes, e3 ) );
        assertEquals( "d", Huffman.decodeLettre( codes, e4 ) );
    }

    @Test
    public void testCodeExiste() {
        StringBuffer character = new StringBuffer();
        assertTrue( Huffman.codeExiste( codes, "0", character ) ); //System.out.println(character); // a
        assertTrue( Huffman.codeExiste( codes, "10", character ) ); //System.out.println(character);    // b
        assertTrue( Huffman.codeExiste( codes, "110", character ) ); //System.out.println(character);   // c
        assertTrue( Huffman.codeExiste( codes, "111", character ) ); //System.out.println(character);   // d
    }

    @Test
    public void testEncode() {
        StringBuffer codage = new StringBuffer();
        arbre1.encode( codage );
        String codageAttendu = "00010110010010110001110110001010110000 1";
        assertEquals( codageAttendu, codage.toString() );
    }
}
```



Tests passés avec succès