University of London International Programmes
Computing coursework assignments 2016–2017

CO3325 Data compression
Coursework assignment 1

This coursework assignment is designed to help you enrich your learning experience and to encourage self-study and creativity. You may write down, if necessary, additional assumptions in your coursework answers that you employed to simplify your implementation tasks or to help you understand issues. You should, however, attempt the exercises at the end of each chapter in the textbook or subject guide before doing the coursework, otherwise you may find the tasks in the coursework assignment difficult and the experience less rewarding. You should read the coursework assignments or questions carefully and pay particular attention to the Submission requirements on page 5. Note: programme source code may be checked using plagiarism detection programs.

Develop a prototype Java program to demonstrate how the Adaptive Huffman Coding (encoding and decoding) algorithms work.

The Adaptive Huffman Coding algorithms have a practical impact, and they are the basis of the UNIX `compact` program. The compressor (encoding program) and the decompressor (decoding program) 'mirror' each other in steps, both starting with an empty Huffman tree and updating the tree when an input symbol is read and processed. Here 'process' means *compress* for a compressor, and *decompress* for a decompressor.

Your program should demonstrate step by step the progress of running both encoding and decoding algorithms. For each step, trace the values of the *input*, *output*, *alphabet* and the *tree structure*.

For example, the sequence of symbols to be encoded is `abcbbdaaddd` initially. Your encoding program should display the following steps (the left branch of each tree node is labelled as `0` and the right `1`):

```
(0) Initialise
A={DAG(0)}

Tree (with a single node):
             0
            /
          DAG(0)

(1)
Read-input:   a
Output    :   0 ASCII("a")

A={a(1), DAG(0)}

Tree:       1
         /     \
       a(1)   DAG(0)

(2)
Read-input:   b
Output    :   1 ASCII("b")

A={b(1),a(1),DAG(0)}

Tree:       2
         /     \
        1      b(1)
      /   \
    a(1)  DAG(0)
```

```
(3)
Read-input:   c
Output    :   10 ASCII("c")

A={c(1),b(1),a(1),DAG(0)}

Tree:         3
           /     \
         1         2
      /    \     /    \
   a(1)  DAG(0) c(1) b(1)

(4)
Read-input:   b
Output    :   11

A={b(2),c(1),a(1),DAG(0)}

Tree:         4
           /    \
          2      b(2)
       /   \
      1     c(1)
   /    \
a(1)  DAG(0)
```

```
(5)
Read-input:   b
Output    :   1

A={b(3),c(1),a(1),DAG(0)}

Tree:         5
           /    \
         b(3)    2
               /   \
             1     c(1)
          /    \
        a(1) DAG(0)

(6)
Read-input:   d
Output    :   101 ASCII("d")

A={b(3),d(1),c(1),a(1),DAG(0)}

Tree:        6
           /   \
          3    b(3)
        /   \
      1        2
    /   \     /   \
 a(1) DAG(0)  d(1) c(1)
```

```
(7)
Read-input:   a
Output    :   000

A={b(3),a(2),d(1),c(1),DAG(0)}

Tree:           7
              /    \
             4      b(3)
          /    \
        2      a(2)
      /   \
    1    d(1)
  /    \
c(1) DAG(0)
```

```
(8)
Read-input:   a
Output    :   01

A={b(3),a(3),d(1),c(1),DAG(0)}

Tree:             8
                /   \
               5     b(3)
             /   \
          a(3)    2
                /   \
               1    d(1)
             /    \
          c(1)   DAG(0)
```

```
(9)
Read-input:   d
Output    :   011

A={b(3),a(3),d(2),c(1),DAG(0)}

Tree:          9
            /     \
           3       6
         /  \     /  \
     d(2)   1   b(3) a(3)
          /   \
       c(1)  DAG(0)
```

```
(10)
Read-input:   d
Output    :   00

A={b(3),a(3),d(3),c(1),DAG(0)}

Tree:   10
      /    \
    6       4
   / \     /  \
b(3) a(3) d(3)  1
              /  \
           c(1) DAG(0)
```

```
(11)
Read-input:   d
Output    :   10

A={d(4),b(3),a(3),c(1),DAG(0)}

Tree:       11
          /    \
         7       4
       /  \     /  \
    d(4) b(3)  a(3)  1
                   /  \
                c(1) DAG(0)
```

So, the encode for sequence `abcbbdaaddd` is `0 ASCII("a") 1 ASCII("b") 10 ASCII("c") 11 1 101 ASCII("d") 000 01 011 00 10`.

Note: Your answer may not be identical to the one here due to implementation details.

Hints:

The most challenging part of the programming task may be centred at the Huffman tree. You would really need a proper design of your compressor/decompressor in diagrams *before* any implementation. There are many decisions to be made. For example, a basic compressor/decompressor would simply run the static Huffman algorithms to construct the Huffman tree based on the alphabet and frequency table for each step. An advanced compressor would just efficiently update the Huffman tree for each step.

The Huffman tree has a so-called *sibling property* that needs to be maintained. That is, if you scan them level by level from the bottom (left most leaf) up to the top (root), you will find that the frequency values of the tree nodes are actually sorted in ascending order, from the smallest value to the biggest value.

There are also many ways to implement the Huffman tree, for example, you may use the linked tree structure or array. Hence you may need to review your work about implementation of trees in the Data Structure and Algorithms module (see Subject guide CO2226 Software engineering, algorithm design and analysis – Volume 2) in your second academic year to decide whether you would use the linked tree structure or an array to represent the Huffman tree.

You are encouraged to explore and highlight (in the Discussion section of your report) various implementation details.

The correctness of your program carries [60%] of the marks for program Code and any advanced features including efficiency [40%].

[END OF COURSEWORK ASSIGNMENT 1]

CO3325 Data compression Coursework assignment 2

This coursework assignment is designed to help you enrich your learning experience and to encourage self-study and creativity. You may write down, if necessary, additional assumptions in your coursework answers that you employed to simplify your implementation tasks or to help you understand issues. You should, however, attempt the exercises at the end of each chapter in the textbook or subject guide before doing the coursework, otherwise you may find the tasks in the coursework assignment difficult and the experience less rewarding. You should read the coursework assignments or questions carefully and pay particular attention to the Submission requirements on page 5. Note: programme source code may be checked using plagiarism detection programs.

This assignment consists of a number of tasks as follows:

I. Develop a prototype Java program that helps explore the effect of preprocessing techniques on a grey-scale image. [50%]

Given integers (m, n, a, b) as the input, your program should:

(a) generate a random matrix $\mathbf{M}_{m\times n} = (M_{ij})_{m\times n}$ of real values $\in [a, b]$ (i.e. ranging from a to b inclusive), where $i = 1, \cdots, m, j = 1, \cdots, n$ and $M_{ij} \in [a, b]$
(b) compute the residual matrix $\mathbf{R}_{m\times n} = (R_{ij})_{m\times n}$ of a given matrix
(c) apply quantisation to a given matrix, i.e. round (or truncate) each M_{ij} and return the integer matrix $\mathbf{R}_{m\times n} = (Q_{ij})_{m\times n}$.

For example, the program can take the input $(3, 4, -256, 255)$ as the input, generate a matrix of random values within the range $[-256, 255]$, for example,

$$\mathbf{M} = \begin{pmatrix} 123.45 & -112.30 & 224.50 & 34.30 \\ 25.03 & -100.01 & 25.00 & 255.00 \\ 224.02 & 225.40 & 33.00 & 48.45 \end{pmatrix}.$$

The program should perform quantisation on $\mathbf{M}$ to get:

$$\mathbf{Q} = \begin{pmatrix} 123 & -112 & 225 & 34 \\ 25 & -100 & 25 & 255 \\ 224 & 225 & 33 & 48 \end{pmatrix}.$$

Following JPEG rule (d) at the user's choice, the program should return the residual matrix:

$$\mathbf{R} = \begin{pmatrix} 123 & -112 & 225 & 34 \\ 25 & 123 & -112 & 225 \\ 224 & 25 & -100 & 25 \end{pmatrix}.$$

The JPEG rules can be highlighted as follows, where x is the estimate:

T	S
Q	x

(a) No prediction
(b) $x = Q$
(c) $x = S$
(d) $x = T$
(e) $x = Q + S - T$
(f) $x = Q + (S - T)/2$
(g) $x = S + (Q - T)/2$
(h) $x = (Q + S)/2$

II. You should use your program in part I as preprocessing and then compress the given image source using the Adaptive Huffman program that you developed in the previous assignment and investigate the differences that the preprocess makes. [20%]

You may use another lossless compression program if you did not do the Huffman Coding Assignment previously.

III. Using your programs, demonstrate how an analytical report may be produced to show the differences between the compression processes with and without the preprocessing in terms of compression efficiency. For example, what difference has the preprocessing made on the source in terms of the compression ratio with and without the preprocessing? What changes has the preprocessing made on the source in terms of the entropy and average code length? [30%]

Your report should be concise, so a table (or diagram) would be better than text. For example,

One JPEG rule on four image sources

JPEG rule(d), MSE and Adaptive Huffman	source (a)	source (b)	source (c)	source (d)
original (number of symbols)				
compressed (number of symbols)				
compress ratio				
distortion MSE				
entropy				
average code length				

or

Four JPEG rules on one image source

Source (a), MSE and Adaptive Huffman	JPEG rule(d)	JPEG (e)	JPEG (f)	JPEG (g)
original (number of symbols)				
compressed (number of symbols)				
compress ratio				
distortion MSE				
entropy				
average code length				

[END OF COURSEWORK ASSIGNMENT 2]

Submission requirements

These requirements apply to both Coursework assignments. The available marks are given in square brackets.

1. Naming conventions for any `.pdf` or `.zip` file submissions.
   When naming your files, please ensure that you include your full name, student number, course code and assignment number, e.g. `FamilyName_SRN_COxxxxcw#.pdf`
   (e.g. `Zuckerberg_920000000_CO3325cw2.pdf`), where
   - `FamilyName` is your family name (also known as last name or surname) as it appears in your student record (check your student portal)
   - `SRN` is your Student Reference Number, for example 920000000
   - `COXXXX` is the course number, for example CO3325, and cw# is either cw1 (coursework 1) or cw2 (coursework 2).
2. Your coursework submission must include a report Document [40%] and the program Code [60%].
   The Document (preferable in .pdf format) should include the following sections:
   (a) Algorithms (in flow-chart)
   (b) Design (in block diagram or class-diagram in UML)
   (c) Demonstration (in 5 best screen-shots)
   (d) Discussion (including answers to any questions/problems in the Coursework assignment, and your experience in attempt of the coursework and full bibliography)

   The program code should include the
   (a) Java source codes .java
   (b) executable version .class.
3. Execution of your programs:
   [Penalty] A ZERO mark may be awarded if:
   - your program(s) cannot be run from the coursework directory by a simple command '*java menu*' (this means that you should name your main class 'menu', or adopt the `menu.java` that can be found in the Appendix on page 6),
   - your source code(s) does not compile and you give no information on your program execution environment,
   - your program(s) does not do what you claim it should do,
   - your program(s) crashes within the first *three* interactive execution steps,
   - your program(s) works for the first time of execution only, or
   - there is no comment in your source code.
4. You should monitor and report the time you have spent for each part of the coursework answers. Please leave a note to the examiner if you need to raise any issue at the beginning of your coursework answers as follows:

Total number of hours spent	
Hours spent for algorithm design	
Hours spent for programming	
Hours spent for writing report	
Hours spent for testing	
Note for the examiner (if any):	

5. Show *all* your work. Any use of others' work should be declared at the point of use and referred to in the *Bibliography* section at the end of your coursework answers.

Appendix

This is an example. Please modify accordingly to suit your own purposes.

```
import java.lang.*;
import java.io.*;
// Modify the display content to suit your purposes...
class menu {
private static final String TITLE =
"\nCO3325 Data Compression coursework\n"+
"   by FAMILYNAME-firstname_SRN\n\n"+
"\t********************\n"+
"\t1. Declaration: Sorry but part of the program was copied
from the Internet! \n" +
"\t2. Question 2 \n"+
"\t3. Question 3 \n"+
"\t4. no attempt \n"+
"\t0. Exit \n"+
"\t********************\n"+
"Please input a single digit (0-4):\n";
menu() {
int selected=-1;
while (selected!=0) {
System.out.println(TITLE);
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
// selected = Integer.parseInt(in.readLine());
try {
                       selected = Integer.parseInt(in.readLine());

                           switch(selected) {
                                case 1:  q1();
                                  break;
                                case 2:  q2();
                                  break;
                                case 3:  q3();
                                  break;
                                case 4:  q4();
                                  break;}      }
   catch(Exception ex) {}  } // end while
                System.out.println("Bye!");
}
// Modify the types of the methods to suit your purposes...
private void q1() {
System.out.println("in q1");
}
private void q2() {
System.out.println("in q2");
}
private int q3() {
System.out.println("in q3");
return 1;
}
private boolean q4() {
System.out.println("in q4");
return true;
}
   public static void main(String[] args) {
new menu();
   }
}
```

[END OF SUBMISSION REQUIREMENTS]