University of London
Computing and Information Systems/Creative Computing
CO3325 Data compression
Coursework assignment 1 2018–2019

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 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 7. Note: program source code may be checked using plagiarism detection programs and you should use Harvard short-form for in-text citation and referencing.

Develop a prototype Java program to demonstrate how the LZW (encoding and decoding) algorithms work on an English text file.

Your program should demonstrate the step-by-step progress of running both encoding and decoding algorithms. For each step, demonstrate the main characteristics of the algorithms by tracing the values of the main variables such as `next character x, word+x, output, new entry of the dictionary` and `(updated) word`.

For example, let the input string be `AGGAGAGAG`.

Your program would display the texts below upon execution and it would demonstrate how the same version of a dictionary is maintained independently for the encoder and decoder.

```
Encoder:
 1. Symbols to be read: AGGAGAGAG
 read next_character x:  A
 word+x: A  (in dictionary)
 word  : A

 2. Symbols to be read: GGAGAGAG
 read next_character x:  G
 word+x: AG  (not in dictionary)
 output: 33
 new entry of the dictionary: 96:AG
 word  : G

 3. Symbols to be read: GAGAGAG
 read next_character x:  G
 word+x: GG  (not in dictionary)
 output: 39
 new entry of the dictionary: 97:GG
 word  : G

 4. Symbols to be read: AGAGAG
 read next_character x:  A
 word+x: GA  (not in dictionary)
 output: 39
 new entry of the dictionary: 98:GA
 word  : A
```

```
5. Symbols to be read: GAGAG
read next_character x:  G
word+x: AG  (in dictionary)
word  : AG

6. Symbols to be read: AGAG
read next_character x:  A
word+x: AGA  (not in dictionary)
output: 96
new entry of the dictionary: 99:AGA
word  : A

7. Symbols to be read: GAG
read next_character x:  G
word+x: AG  (in dictionary)
word  : AG

8. Symbols to be read: AG
read next_character x:  A
word+x: AGA  (in dictionary)
word  : AGA

9. Symbols to be read: G
read next_character x:  G
word+x: AGAG  (not in dictionary)
output: 99
new entry of the dictionary: 100:AGAG
word  : G

10 Symbol to be read: none
read next_character x:  EOF
output: 39

So the total output is :  33 39 39 96 99 39
The new entries of the dictionary are:
96:AG, 97:GG, 98:GA, 99:AGA, 100:AGAG,
Decoder:
 1. Tokens to be read: 33 39 39 96 99 39
 Read next_token (x): 33
 output:  A
 word  : A

 2. Tokens to be read: 39 39 96 99 39
 Read next_token (x): 39
 Look up the dictionary and output element:  G
 Dictionary (new entries so far):
 96:AG
 word   : G

 3. Tokens to be read: 39 96 99 39
 Read next_token (x): 39
 Look up the dictionary and output element:  G
 Dictionary (new entries so far):
 96:AG 97:GG
 word   : G

 4. Tokens to be read: 96 99 39
 Read next_token (x): 96
 Look up the dictionary and output element:  AG
 Dictionary (new entries so far):
 96:AG 97:GG 98:GA
 word   : AG

 5. Tokens to be read: 99 39
 Read next_token (x): 99
 Look up the dictionary and find there is no entry yet for 99
 element: AGA/* element=word+first_char_of_word */
 output element: AGA
 Dictionary (new entries so far):
 96:AG 97:GG 98:GA 99:AGA
 word   : AGA

 6. Tokens to be read: 39
 Read next_token (x): 39
 Look up the dictionary and output element:  G
 Dictionary (new entries so far):
 96:AG 97:GG 98:GA 99:AGA 100:AGAG
 word   : G

 7. Tokens to be read: (none)
 Read next_token (x): EOF

 The original Text is AGGAGAGAG
 The new entries of the dictionary are:
 96:AG, 97:GG, 98:GA, 99:AGA, 100:AGAG
```

Note: Your answer does not have to be identical to the example due to design or implementation details.

Hints: You would really need a proper design of your compressor/decompressor in diagrams *before* any implementation. There are many decisions to be made and many questions to be answered. For example, you may ask, what would be the main functions of my prototype program? What would constitute a basic compressor or decompressor that would simply return an encoded or decoded message? What extra information might my prototype offer to the user? Would I also allow the user to run my prototype step by step? Would my prototype also report to the user about the compression ratio or any other useful information? What data structures would I use to make my prototype program more efficient?

There are often many ways to implement a data structure. For example, you may use a linked list, array, or tree. Therefore, you may need to review your work in the Data Structure and Algorithms module (see Volume 2 of the Subject Guide CO2226 Software engineering, algorithm design and analysis).

Assume that the source is stored in `source.txt`, and the compressed file is in `code.txt`. You may add more assumptions to ease your programming tasks.

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

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

[END OF COURSEWORK ASSIGNMENT 1]

University of London
Computing and Information Systems/Creative Computing
CO3325 Data compression
Coursework assignment 2 2018–2019

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 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 7. Note: program source code may be checked using plagiarism detection programs and you should use Harvard short-form for in-text citation and referencing.

Develop a Java prototype program to help you explore differences in compression performances.

I. Develop a program that converts a matrix of decimal integers to a matrix of binary code (codewords) and to a matrix of Gray code.

   The input and output of your program should be read from and sent to a text file, namely *input.txt* and *output.txt*, respectively. The program should be able to read at least the following data as the input:

   (a) A one dimensional array of integers (decimal values ranging from 0 to 255, *i.e.* [0,255], inclusive 0 and 255)
   (b) Two dimensional array of integers [0,255]
   (c) Three dimensional array of integers [0,255].

   The output of the program should contain the following:

   (a) A one dimensional array of binary codewords and their Gray codes
   (b) Two dimensional array of binary codewords and their Gray codes
   (c) Three dimensional array of binary codewords and their Gray codes.

II. You should apply ONE of the lossless compression algorithms below to the sources of the binary and Gray code, and you may *reuse* the program developed in your previous coursework assignment:

   (1) Run-length
   (2) Huffman coding
   (3) Shannon-Fano coding
   (4) Arithmetic coding
   (5) Dictionary compression

III. Your program should report statistics of changes in the process. For example, what are the changes (if any) in the entropy of the source of binary code and that of Gray code? What are the changes (if any) in the average lengths of the codewords? Are there any other changes of the source, or anything else interesting to report?

   The compression efficiency should also be reported using the analytical techniques learnt from the course. For example, what are the changes (if any) in the gap between the entropies of the two coding sources?

   While being encouraged to design good user interfaces, you are welcome to use essential user interfaces, such as tables (or diagrams), that include at least the following information:

Parts	Prompt for input	Display for output
Encoding	text source	encoded compression result
Decoding	encoded compression result	decoded original source
Computing	any required input	compression ratio entropy average length of the code

Note: If you use others' program code (including part of others' code), you must identify this code and its source both in your document (Discussion section) and in your code (on the Main Menu).

[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.* `YourName_SRN_COxxxxcw#.pdf`
   (*e.g.* `MarkZuckerberg_920000000_CO3325cw2.pdf`), where
   - `YourName` is your name as it appears on 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 (must be in .pdf format, and should be uploaded as a single, separate file, *i.e.* not in a folder, zipped, *etc.*) 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 Reference section in Harvard format)

   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 8),
   - 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 Reference 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]