UNIVERSITY OF LONDON

GOLDSMITHS COLLEGE

B. Sc. Examination 2004

COMPUTING AND INFORMATION SYSTEMS

IS53010A(CIS325) Data Compression

Duration: 2 hours and 15 minutes

Date and time: May 2004

Answer THREE questions only.
Full marks will be awarded for complete answers to THREE questions.
There are 75 marks available on this paper.
Electronic calculators may be used. The make and model should be specified on the script and the calculator must not be programmed prior to the examination.

THIS EXAMINATION PAPER MUST NOT BE REMOVED FROM THE EXAMINATION ROOM

Question 1

(a) Determine whether the following codes for the alphabet {a, b, c, d} are *uniquely decodable*. Give your reasons for each case. [8]

- (i) {1, 011, 000, 010}
- (ii) {1, 10, 101, 0101}
- (iii) {0, 01, 011, 0111}
- (iv) {0, 001, 10, 011}

(b) Consider a text model for a black-white source file. Under what probability distribution condition does a static Huffman encoding algorithm achieve the best performance? Under what condition on the source does the algorithm perform the worst? Give your reasons. [7]

(c) Outline a simple version of the adaptive Huffman encoding algorithm. [5]

(d) Illustrate how adaptive Huffman encoding works on a source of text CAAABB. [5]

Hint:

- *(i) You may demonstrate step by step the progress of running the Adaptive Huffman encoding algorithm in terms of the* input, output, alphabet *and the* tree structure.
- *(ii) Add sufficient comments in your algorithm if it is different from the one discussed in lecture.*

Question 2

(a) Consider part of a grayscale image with 16 shades of gray that is represented by the array A below:

```
A:  0011 1000 1000 0010
    1100 1000 1100 0110
    1000 1100 1001 1001
```

Demonstrate how the image can be represented by several bitplanes (bi-level images) [4]

(b) Explain, with the aid of an example, why a Huffman code is in general not optimal unless the probability of every symbol in the source alphabet is a negative power of 2. [5]

(c) One way to improve the efficiency of Huffman coding is to maintain two sorted lists during the encoding process. Using this approach, derive a canonical minimum variance Huffman code for the alphabet {A,B,C,D,E,F} with the probabilities (in %) 34,25,13,12,9,7 respectively. [8]

(d) Explain, with the aid of an example, each of the following terms: [8]

(i) fixed-to-variable model

(ii) gray-scale image

Question 3

(a) Explain what is used to represent the so-called colour depth in a common RGB colour model. What is the value of the colour depth in a representation where 8 bits are assigned to every pixel? [4]

(b) Comment on the *truth* of the following statement describing the absolute limit on lossless compression. [4]

"No algorithm can compress even 1% of all files, even by one byte."

(c) Explain briefly what is meant by *Canonical* and *Minimum-variance* Huffman coding, and why it is possible to derive two different Huffman codes for the same probability distribution of an alphabet. [4]

(d) Describe briefly, with the aid of an example, how Shannon-Fano encoding differs from *static* Huffman encoding. [5]

Hint: You may give an example by deriving a code for {A,B,C,D} with probability distribution 0.5, 0.3, 0.1, 0.1 using the two algorithms, and show the difference.

(e) A binary tree (0-1 tree) can be used to represent a code containing a few codewords of variable length. Consider each of the four codes ((i)-(iv) below) for alphabet {A,B,C,D} and draw the binary tree for each code.

(i) {000, 001, 110, 111}
(ii) {110,111,0,1}
(iii) {0000,0001,1,001}
(iv) {0001,0000,0001,1}

For each tree drawn, comment on whether the code being represented by the binary tree is a prefix code and give your reasons. [8]

Question 4

(a) Explain the meaning of the following terms: [8]

  (i) rate-distortion problem
  (ii) entropy
  (iii) prefix codes
  (iv) motion prediction.

(b) Given an alphabet of four symbols {A,B,C,D}, discuss the possibility of a uniquely decodable code in which the codeword for A has length 1, that for B has length 2 and for both C and D have length 3. Justify your conclusions. [4]

(c) Derive the output of the HDC algorithm on the source sequence below. Explain the meaning of each control symbol that you use. Finally, describe briefly a data structure(s) and algorithm(s) that can be used in solving *counting* sub-problems in the HDC algorithm. [5]

```
TTU␣␣␣␣␣␣␣␣␣K␣␣RR33333333333333␣␣PPPEE
```

Hint: You may simply describe the data structure(s) and algorithm(s) that you have used in your Lab Exercise when implementing the HDC algorithm.

(d) Explain the concept of bitmapped images and vector graphics. What are the differences between vector and bitmapped graphics in terms of *requirements* to the computer storage capacity, and the *size* of the image files. [8]

Question 5

(a) It has been suggested that the unique decodability is not a problem for any fixed length code. Explain why this is so with the aid of an example. [4]

(b) Describe the main idea of *predictive encoding*. Suppose the matrix below represents the pixel values (in decimal) of part of a grayscale image. Let the prediction be that each pixel is of the same value as the one to its left. Illustrate step by step how predictive encoding may be applied to the array. [6]

```
1 1 1 1
5 1 1 1
5 5 5 5
7 9 4 5
```

(c) Demonstrate step by step how the Basic LZW *encoding* and *decoding* algorithms maintain the same version of a dictionary without ever transmitting it between the compression and the decompression end, using a small source string `BBGBGH` as an example. [15]

The LZW encoding and decoding algorithms are given below.

(i) Encoding

```
1. word=‘’;
2. while not end_of_file
      x=read_next_character;
      if word+x is in the dictionary
         word=word+x
      else
         output the dictionary index for word;
         add word+x to the dictionary;
         word=x;
3. output the dictionary number for word;
```

(ii) Decoding

```
1. read a token x from compressed file;
2. look up dictionary for element at x;
   output element
   word=element;
3. while not end_of_compressed_file do
      read x;
      look up dictionary for element at x;
      if there is no entry yet for index x
         then element=word+first_char_of_word;
      output element;
      add word+first_char_of_element to the dictionary;
      word=element;
4. end
```