Source Coding - Encoding

Information Theory Lab 4

Objective

Understand linear block codes by implementing a basic encoding application.

Theoretical notions

See lecture notes.

Exercises

1. Write a C program to perform a linear block encoding of every byte from a given data file. The program shall be called as follows:

   `Encode.exe code.dat input.txt output.txt`

- The arguments are:
  - `code.dat`: a file containing the code to be used (known as the "codebook" file)
  - `input.txt`: the file to encode
  - `output.txt`: the output (encoded) file

The codebook file contains a vector of 256 elements of the following structure type:

```
typedef struct
{
    int len;                /* length of code, in bits */
    unsigned long code;     /* the first "len" bits are the codeword */
} CODE32BIT;
```

- The program will follow the following steps:
  - Read the full vector from the codebook file;

  - Allocate an array named `out` of `unsigned char` of max size 10MB (i.e. 10000000 bytes);
  - The, open the input file and read every byte in a loop. For each byte do the following:
  - Write the codeword for the byte, bit by bit, in the `out` vector. You need to keep track of the number of bits written, in order to continue writing from where the previous codeword stopped.
  - Write the output data to the output file, as follows:
    * Open the second file for writing
    * Write first the total number of bits
    * Write afterwards the vector `out`, but not more than the number of bytes actually used for coding
    * *Note: when decoding the file, we will read back the data in the same order.*

2. Encode the file `textro.txt` with the provided codebook `codero.dat`. Check the size of the output file and compute the compression ratio.

3. Repeat 2. for `texten.txt` with codebook `codeen.dat`.

4. Encode a file with the codebook from the other language. Check the size of the output file and compute the compression ratio. Compare with the one using the same language codebook. Which case is better?

Implementation hints

- The following C functions may be used for file-based operations. Look up their documentation on the Internet (e.g. *cplusplus.com*, or Google search).
  - `fopen(...)`, to open a file for reading;
  - `fread(...)`, to read byte data from the file;
  - `fclose()`, to close the file when finished.
- The following macros implement the basic bit operations:
  - reading a single bit i from a number x;
  - set bit i from a number x to 1;
  - clear bit i from a number x (i.e. set to value 0);
  - change the value of bit i from a number x (i.e. if 0 make 1, if 1 make 0).

```
#define READ_BIT(x,i)       (((x) & (1U << (i))) != 0)
#define SET_BIT(x,i)        ((x) = (x) | (1U << (i)))
#define CLEAR_BIT(x,i)      ((x) = (x) & ~(1U << (i)))
#define TOGGLE_BIT(x,i)     ((x) = (x) ^ (1U << (i)))
#define WRITE_BIT(x,i,val)  ((val) ? SET_BIT((x),(i)) : CLEAR_BIT((x),(i)))

#define VECREAD_BIT(v,i)       (READ_BIT((v[(i)/8]),(i)%8))
#define VECWRITE_BIT(v,i,val)  (WRITE_BIT((v[(i)/8]),)
```

- When writing the i-th bit in a large vector of bytes, `i/8` is the index of the byte and `i%8` is the bit inside that byte.
  - Example: the 20th bit in vector `out` = bit number 4 from byte `out[2]`.

Final questions

1. TBD
2. TBD