Homework #1

Due on 11:59pm Wednesday April, 24th 2019

Problem 1

In this problem you will explore the concept of entropy, and write a C++ program that can compress and decompress images using the Huffman Coding algorithm. Your program will read the image on `stdin` and outputs the result on `stdout`, and get in the required operation on the command line.

a. [5 points] Implement the function `compute_entropy` in the given file to compute the entropy for each of the included PGM files. The symbols are the grayscale values from 0 to 255 (one byte each). Estimate the probability by the relative frequency of occurrence of each value i.e. $P(v) = count\ (v)/N$ where *count*(*v*) is the number of pixels with value *v* and *N* is the total number of pixels in the image and

   [Note: any grayscale value that is not in the image is ignored when computing the entropy since its probability will be zero and $\log_2 0$ is infinity.]

   [Hint: Pay special attention to the files `msgx.pgm`. You can compute the entropy by hand to verify your code is working correctly.]

b. [5 points] Implement the function `build_tree` to build a Huffman Tree that takes in symbol probabilities computed in the previous step. Print the computed Huffman Tree and the codes for the symbols for the images `msg1.pgm`, `msg2.pgm` and `msg3.pgm`.

c. [5 points] Implement the function `encode` to encode an input message and output an array of codewords. You don't need to worry about bit compaction for this problem, but you need to keep track of how many bits are in the encoded image.

   Encode the four images that were shown in class (sena, sensin, earth, omaha) and compare the numbers of bytes you get for the compressed image with the numbers given in class. Comment on your results in the report.

d. [5 points] Implement a simple *decorrelation* method to try to reduce the number of bytes needed for the compression process. In particular, instead of encoding the given grayscale values directly, encode the difference between each pixel and the one on its left. So, given an input image, compute the *difference* image by subtracting from each pixel the value of its *left* neighbor, and then use that image to build the tree and encode. Note that for the first column there is no left neighbor, so just use the pixel values.

   Encode the four images that were shown in class (sena, sensin, earth, omaha) and compare the numbers of bytes you get for the compressed image with the numbers given in class. Comment on your results in the report.

Command Line

You need to modify the main file `hw1_p1.cpp` to include the required functionality. Your program should be named hw1_p1, and should be called as follows:

- To compute entropy for an image:

  ```
  ./hw1_p1 -entropy < input.pgm
  ```

  this will output the entropy of the image `input.pgm`, for example

  ```
  ./hw1_p1 -entropy < msg1.pgm
  ```

- To build a Huffman Tree from an image and prints it:

  ```
  ./hw1_p1 -tree < input.pgm
  ```

  where the input image is called `input.pgm` and the output is written to `stdout`. For example, to compute the Huffman Tree on `msg1.pgm`, you could run:

  ```
  ./hw1_p1 -tree < msg1.pgm
  ```

- To encode an image:

  ```
  ./hw1_p1 -encode < input.pgm
  ```

  where the input image is called `input.ppm` and the number of bytes used in the encoding is written to `stdout`. For example, to encode `sena.pgm`, you could run:

  ```
  ./hw1_p1 -encode < sena.pgm
  ```

- To encode an image by computing the difference first:

  ```
  ./hw1_p1 -encode -diff < input.pgm
  ```

  where the input image is called `input.pgm` and the number of bytes used in the encoding is written to `stdout`. For example, to encode `sena.pgm`, you could run:

  ```
  ./hw1_p1 -encode -diff < sena.pgm
  ```

PGM File

The PGM file is a text file representation for images. It has several formats depending on whether the image is binary, grayscale, or color. For more information, check https://en.wikipedia.org/wiki/Netpbm_format.

The color PGM format is defined as:

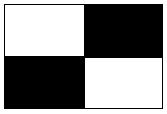
```
P2
[xres] [yres]
[max intensity]
[g0] [g1] [g2] ...
```

where `g0` is the grayscale value of the top-left pixel, and pixels go from the top-left towards the bottom right

For example:

```
P2
2 2
255
255 0
0 255
```

represents a 2x2 image (with 4 pixels) where the top-left pixel is white, the top-right is black, the bottom-left is black, and the bottom-right is white.



To view PGM files, you can use the `display` utility (part of `ImageMagick`) as follows:

```
display image.PGM
```

Problem 2

In this problem, you will write a C++ program that can compress and decompress images using the Arithmetic Coding algorithm. Your program will read the specified image file and outputs the resulting image as a file, and get in the required operation on the command line.

a. [5 points] Implement the functions `binary_to_decimal()` and `decimal_to_binary()` to convert to/from decimal fractions and their binary representations. Test your functions using simple values to make sure it's working.

   You can use the `bitset` container from the STL if you want, but it's not necessary. For more information, check this page http://www.cplusplus.com/reference/bitset/bitset/.

b. [5 points] Implement the function `encode()` to encode an image using arithmetic coding and output the total number of bits required on the first line, followed by the tags for every block of symbols, one per line. Try the function on the small images `msg1.pgm`, `msg2.pgm` and `msg3.pgm` to make sure it's working correctly.

c. [5 points] Implement the function `decode()` to decode an image encoded using arithmetic coding and output the corresponding PPM image. Note that you will need to pass in the image metadata i.e. the number of rows, columns, and maximum value on the command line, since these are not included in the encoded message. Try the function on the output of encoding the small images `msg1.pgm`, `msg2.pgm` and `msg3.pgm` to make sure it's working correctly.

d. [5 points] Try your encoder on the four 256x256 images shown in class (sena, sensin, earth, omaha) and compute the compression ratio and number of bits required when using different block sizes (number of symbols per tag) e.g. values for m = 1, 2, 3, 5, 10. Do you get similar numbers to the numbers given in the book? Comment on your answer.

Command Line

You need to modify the main file `hw1_p2.cpp` to include the required functionality. Your program should be named hw1_p2, and should be called as follows:

- To convert from binary to decimal fractions:

  ```
  ./hw1_p2 -bin_to_dec 0x80000000
  ```

  this will output the decimal representation of the binary fraction 0.11 which should be 0.5. Note that the MSB of the binary fraction is the first bit on the input i.e. bit 31, followed by bit 30, … and any bit not specified in the input is assumed to be 0.

- To convert from decimal to binary fractions:

  ```
  ./hw1_p2 -dec_to_bin 0.5
  ```

  this will output the binary representation of the decimal fraction 0.5 which should be 0x80000000.

- To encode an input image and print out the total number of bits required on the first line followed by the binary representation of the tags, one per line:

  ```
  ./hw1_p2 -encode m < input.pgm
  ```

  where *m* is the block size (number of symbols per block), the input image is called `input.pgm` and the output is written to `stdout`. For example, to encode `msg1.pgm` using 3 symbols per block, you could run:

  ```
  ./hw1_p2 -encode 3 < msg1.pgm
  ```

- To decode an image encoded above:

  ```
  ./hw1_p2 -decode m xres yres max < image.enc
  ```

  where *m* is the block size (number of symbols per block), the encoded image is called `image.enc` and the original image had xres columns, yres rows, and maximum intensity max. For example, to decode the encoded version of `sena.pgm`, using 3 symbols per block, you could run:

  ```
  ./hw1_p2 -encode 3 < sena.pgm | ./hw1_p2 -decode 3 256 256 255
  ```

Problem 3

In this problem, you will write a C++ program that can compress and decompress images using dictionary coding. Your program will read the image on `stdin` and outputs the resulting image on `stdout`, and get in the required operation on the command line.

a. [5 points] Implement the function `encode()` to encode an image using the LZ77 algorithm and output the total number of bits required on the first line, followed by the triplets **(*j*, *k*, *x*)** one per line. The triplets represent the match position *j*, the match length *k*, and the code of the symbol following the match i.e. three integers separated by a space. Try the function on the small images `msg1.pgm`, `msg2.pgm` and `msg3.pgm` to make sure it's working correctly.

   Note: we will only use *fixed-length* coding for the triplets, so the number of bits required depend on the size of the search buffer *S*, the size of the lookahead buffer *T*, and the size of the alphabet.

b. [5 points] Implement the function `decode()` to decode an image encoded using the LZ77 algorithm and output the corresponding PGM image. Note that you will need to pass in the image metadata i.e. the number of rows, columns, and maximum value on the command line, since these are not included in the encoded message. Try the function on the output of encoding the small images `msg1.pgm`, `msg2.pgm` and `msg3.pgm` to make sure it's working correctly.

c. [5 points] Try your encoder on the four 256x256 images shown in class (sena, sensin, earth, omaha) and compute the compression ratio and number of bits required when using different values for the parameters *S* and *T*.

d. [5 points] Using the best parameter value from above, try encoding the *difference* values instead of the pixel values. In particular, for every pixel, instead of encoding the pixel value, encode the difference between that pixel and the pixel on the left (except the first column which are encoded as is).

   Do you get similar numbers to the numbers given in the book? Comment on your answer.

Command Line

You need to modify the main file `hw1_p3.cpp` to include the required functionality. Your program should be named `hw1_p3`, and should be called as follows:

- To encode an input image and print out the total number of bits required on the first line followed by the triplets, one per line:

  ```
  ./hw1_p3 -encode S T A < input.ppm
  ```

  where *S* is legnth of the search buffer, *T* is the size of the lookahead buffer, and *A* is the number of symbols in the alphabet, the input image is called `input.ppm` and the output is written to `stdout`. For example, to encode `msg1.pgm` using a search buffer of 256, a lookahead buffer of 16, an alphabet of 3 symbols, you could run:

  ```
  ./hw1_p3 -encode 256 16 3 < msg1.pgm
  ```

- To decode an image encoded above:

  ```
  ./hw1_p3 -decode xres yres max < image.enc
  ```

  where the encoded image is called `image.enc` and the original image had *xres* columns, *yres* rows, and maximum intensity *max*. For example, to decode the encoded version of `sena.pgm`, using a search buffer of 1024, a lookahead buffer of 256, and 256 symbols in the alphabet, you could run:

  ```
  ./hw1_p3 -encode 1024 256 256 < sena.pgm | ./hw1_p3 -decode 256 256 255
  ```

Instructions

- All code should be implemented in C++ under Linux.
- Please present a report containing your answers as well as a zip file containing all your code.
- Please submit your homework in one zip file named as follows: *CMP206.HW##.FirstName.LastName.zip*.
- Please include all your code and sample output in the zip file, with a README file to explain what you did.
- Failure to follow these instructions will cause deductions from your grade.
- You are allowed to discuss the problems among yourselves. However, **copying** any part of the code will result a grade of **ZERO**. No exceptions.

Grading Criteria:

- 80% for satisfying functionality
- 20% for code organization, having enough documentation and following a naming convention consistently.

Submission:

- Submit your zip file in the submission section in the course page on E-learn website with your E-learn account.
- In case of technical issues during submission on E-learn and you are close to missing the deadline, email the solution to the TAs.
- Late submissions will incur a 20% penalty/late day and no submission will be accepted past two days from the due date.