The University of Western Ontario
London, Ontario, Canada
Department of Computer Science
CS 4481b/9628b - Image Compression
Assignment 3
Due Thursday March 15, 2018 at 11:55 PM

INDEPENDENT WORK is required on each assignment.

After finishing the assignment, you have to do the following:

- Type your report and convert it to *PDF format*, which must include:
  - Answers to all questions/requirements in the assignment, if any
  - A copy of all programs that you have written
- Prepare a soft-copy submission, including:
  - A copy of your *typed PDF* report
  - All programs that you wrote: `lz77_encoding_function.c`, `lz77_encoding_function.h`, `lz77_decoding_function.c`, `lz77_decoding_function.h`, `lz77_encoding.c`, `lz77_decoding.c`, `mean_absolute_error.c`, `mean_absolute_error.h`, and `compare_pgm_images.c`.
- Upload the soft-copy submission file-by-file, or as an archived directory.

Late assignments are strongly discouraged.

- 10% will be deducted from a late assignment (up to 24 hours after the due date/time)
- After 24 hours from the due date/time, late assignments will not be accepted.

N.B: When marking your assignment, your program will be tested on the GAUL network. If you will develop your program in any other platform, make sure that your program is error free and produces the required outputs when compiling and running it on the GAUL network.

In this assignment, a read and write PBM/PGM/PPM library is provided (included in the attachment as **pnm_library.tar**). Carefully read and understand the code of this library. *Do NOT change anything in the provided library. Just use it.*

Before starting this assignment, you may want to review some C programming topics, including:

- Arrays
- Pointers & strings
- Dynamic memory allocations
- File input/output

1. (40 marks) Develop the following function and save it in a file named `lz77_encoding_function.c`

   void Encode_Using_LZ77(char *in_PGM_filename_Ptr, unsigned int searching_buffer_size, float *avg_offset_Ptr, float *std_offset_Ptr, float *avg_length_Ptr, float *std_length_Ptr)

   This function should

   - Read from file *in_PGM_filename_Ptr a PGM image
   - Generate an LZ77 compressed file, where the maximum size of the searching buffer is searching_buffer_size. During encoding, if you find more than one match with the same length, *select the one that has the smaller offset*. ***Do not*** apply codeword encoding.
   - Write the compressed image to a file with the same name as the input file name but with the *searching buffer size* and `".lz"` concatenated to it. This compressed file should include:
     - A header that includes ***enough information*** to allow the decoder to decompress the compressed image.
     - An array for all offsets generated from the LZ77 compression
     - An array for all matching_lengths generated from the LZ77 compression
     - An array for all next_symbols generated from the LZ77 compression

- Write the offset histogram data (in comma separated values format, `.csv`) to a file with the same name as the input file name but with the *searching buffer size* and `".offsets.csv"` concatenated to it. This file will have lines of "offset value, frequency" where offset values are recorded in an increasing order. You need to omit any line with zero frequency.
- Write the match length histogram data (in comma separated values format, `.csv`) to a file with the same name as the input file name but with the *searching buffer size* and `".lengths.csv"` concatenated to it. This file will have lines of "length value, frequency" where length values are recorded in an increasing order. You need to omit any line with zero frequency.
- Finally, the function will calculate the average and the standard deviation of the offsets and the match lengths and store them in *avg_offset_Ptr, *std_offset_Ptr, * avg_length_Ptr, and *std_length_Ptr, respectively.

For example, if you call the function as follow: `Encode_Using_LZ77("image.pgm", 1024, &avg1, &std1, &avg2, &std2)`, the function will generate the following files: `image.pgm.1024.lz`, `image.pgm.1024.offset`, and `image.pgm.1024.length`.

It will also store 4 numbers (the average of the offsets, the standard deviation of the offsets, the average of the match lengths, and the standard deviation of the match lengths) in `avg1`, `std1`, `avg2`, and `std1`, respectively.

IMPORTANT: Since you were not asked to do codeword encoding, it is expected that you will not achieve compression.

---

2. (20 marks) Develop the following function and save it in a file named `lz77_decoding_function.c`

   void Decode_Using_LZ77(char *in_compressed_filename_Ptr)

   This function should

   - Read from a file called *in_compressed_filename_Ptr an LZ77 encoded image, which is generated using the Encode_Using_LZ77 function.
   - Interpret the header that you saved at the beginning of the file
   - Decode all pixel values.
   - Write the reconstructed version of the image to a file with the same name as the input file name but with `".pgm"` concatenated to it.

   For example, if you call the function as follow `Decode_Using_LZ77("image.pgm.1024.lz")`, the function will generate a decompressed image called `image.pgm.1024.lz.pgm`.

   IMPORTANT:Since you are doing a lossless compression, you have to make sure that the reconstructed version of the image is EXACTLY the same (bit-by-bit) as the original image.

---

3. (15 marks) To test your functions, write ***main programs*** and save them in files named, `lz77_encoding.c` and `lz77_decoding.c`, where *each program calls one of the two functions developed in Q1 and Q2 separately*. Your programs *must accept all arguments in the command-line*.

   a) For the main program in `lz77_encoding.c`, it will **take as an input from the command line**
   an *input pgm image name* and
   a *searching buffer size*.
   Then, it will call Encode_Using_LZ77 function, which will produce:
   an lz77 compressed file (as described in `lz77_encoding_function.c`)
   an offset histogram data file (as described in `lz77_encoding_function.c`)
   a match length histogram data file (as described in `lz77_encoding_function.c`)

   The main function needs to report the averages and the standard deviations of both the offsets, the match lengths, as well as the ***compression time***.

b) For the main program in **`lz77_decoding.c`**, it will **take as an input from the command line** an *lz compressed file name*.
Then, it will call the Decode_Using_LZ77 function, which will produce:
an lz77 decompressed file

The main function needs to report the ***decompression time***.

c) ***You will also need to develop `mean_absolute_error.c` function and `compare_pgm_images.c` program to compare the two images.***

The prototype of the mean_absolute_error function is defined as follow:
float mean_absolute_error(char *file_name_1_ptr, char *file_name_2_ptr)

- This function accepts two parameters:
  - file_name_1_ptr, a pointer to a string which represents a file name of a PGM image
  - file_name_2_ptr, a pointer to a string which represents a file name of a PGM image
- The function should return a float value that represents the mean absolute error between the two provided images.
- If the two images are not PGM of same size, the function should return -1.
- If the max_gray_values of the two images are not the same, you need to scale the image with the smaller max_gray_value to make its max_gray_value as the larger value.

The **`compare_pgm_images.c`** program must accept ***two*** arguments in the command-line, which are an input PGM file name number 1 and an input PGM file name number 2 and then call the mean_absolute_error function.

You can reuse the same **`mean_absolute_error.c`** function and **`compare_pgm_images.c`** program that you developed in assignment 2 in this assignment as well.

---

4. (25 marks) Use at least the following images to test your programs:
   - Peppers.raw.pgm, a 256x256 image: ***You can download a PGM raw version of the image from the attachment list.***

- goldhill.raw.pgm, a 360x288 image: ***You can download a PGM raw version of the image the from attachment list.***

For each image mentioned above:

- Encode and decode the above images, assuming that the
  - searching_buffer_size = 5120
  - searching_buffer_size = 1024
  - searching_buffer_size = 256
- Make sure that the reconstructed image is EXACTLY the same as the original image.
- Plot *histograms* for the generated **offset values** in *semi-log scale*, i.e., plot the offset values against the *log(frequency)*. Note that, **offset values** ∈ [0, searching_buffer_size].

  Generate one histogram per image per searching_buffer_size, i.e., 2 images × 3 searching_buffer_size values × 1 histogram per searching_buffer_size per image = 6 histograms in total.
- **Comment** on the shape of the offset histograms and **explain** why there are some sorts of ***impulses***, if any, in these histograms.
- Plot *histograms* for the generated **matching_length values** in *semi-log scale*, i.e., plot the match_length values against the *log(frequency)*.

  Generate one histogram per image per searching_buffer_size, i.e., 2 images × 3 searching_buffer_size values × 1 histogram per searching_buffer_size per image = 6 histograms in total.
- **Comment** on the shape of the matching_length histograms.
- Report the ***average*** and the ***standard deviation*** of the data that you used to generate each histogram, i.e., report 12 histograms × 2 values (i.e., the ***average*** and the ***standard deviation***) per histogram = 24 values in total.
- Report the encoding and decoding time for each image when using various searching_buffer_size values, i.e., 2 images × 3 searching_buffer_size values per image × 2 times (one for encoding and the other for decoding) = 12 values in total. **You should do so inside your code, not by using a stop watch.**
- Present all these numbers in a **table** that is **easy to read and understand**.
- In your opinion, **which** searching_buffer_size is **more suitable for each image**?
  Justify your recommendation.

In this assignment, a *makefile* file is provided (included in the attachment as makefile) to facilitate the compilation and the testing processes. Carefully read and understand this makefile. *Do Not change anything in the provided makefile. Just use it.*

N.B: In your program, if you use return 1 to denote that there was an error, the makefile will not continue the rest of the testing cases. To go around this issue, you need to use return 0 instead, but you have to provide an appropriate error message before stopping the program.

To compile Q1, execute "`make Q1`"
To compile Q2, execute "`make Q2`"
To compile Q3a, execute "`make Q3a`"
To compile Q3b, execute "`make Q3b`"
To compile Q3c, execute "`make Q3c`"
To compile all programs, execute "`make all`"

To test the input validation, execute "`make testValidation`"
To test the compression program, execute "`make testCompression`"
To test the decompression program, execute "`make testDecompression`"
To compare images, execute "`make testComparingImages`"
To perform all testing cases, execute "`make testAll`"

To delete compiled programs, execute "`make clean`"
To delete compressed images, execute "`make cleanCOMPRESSED`"
To delete decompressed images, execute "`make cleanDECOMPRESSED`"
To delete compiled programs, compressed and decompressed images, execute "`make cleanAll`"

FYI: Assignment marking scheme includes, but not limited to,

- In-line commenting
- Error free code on GAUL network (syntax)
- Correct implementation (logically)
- Efficient implementation
- Correctly acceptance of the input from the command line

- Appropriateness of the README file (*as described in page 1*)
- The required compressed/decompressed images
- The required comparison between decompressed image and the original image

- The neatness of figure captions
- The neatness of the entire report
- The neatness of the written programs

If your program is not working properly, you still encouraged to submit your work for partial mark. In such case, you should include some comments explaining why your program is doing so.