The University of Western Ontario
London, Ontario, Canada
Department of Computer Science
CS 4481b/9628b - Image Compression
Assignment 3
Due Thursday March 16, 2017 at 11:55 PM

INDEPENDENT WORK is required on each assignment.

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

- Type your report, which should include:
  - Answers to all questions/requirements in the assignment
  - A copy of all programs that you have written
- Prepare a soft-copy submission, including:
  - A copy of your *typed* report
  - All programs that you wrote (use meaningful program names)
  - *A README file to explain*
    - *which-is-which*
    - *the compilation command on GAUL and*
    - *how your programs can be used*
- 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. (35 marks) Develop the following function:

   void Encode_Using_LZ77(char *in_PGM_filename_Ptr, unsigned int searching_buffer_size, char *compressed_filename_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 *compressed_filename_Ptr file. 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

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

2. (35 marks) Develop the following function:

   void Decode_Using_LZ77(char *in_compressed_filename_Ptr, char *out_PGM_filename_Ptr)

   This function should

   - Read from file an LZ77 encoded image *in_compressed_filename_Ptr, 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 *out_PGM_filename_Ptr file.

   Since you are doing a lossless compression, you have to make sure that the reconstructed version of the image is EXACTLY the same as the original image. I.e., **make sure that the reconstructed image is bit-by-bit exactly the same as the original image.**

3. To test your modules, write ***two programs***, *each one to call one of the functions separately*. Your program *must accept all arguments in the command-line*. You should use at least the following test images:
   - camera.raw.pgm, a 256x256 image: ***You can download a PGM raw version of the image from the attachment list.***

   - boats.raw.pgm, a 512x480 image: ***You can download a PGM raw version of the image the from attachment list.***

4. (30 marks) 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.
   - Calculate the ***average*** and the ***standard deviation*** of the data that you used to generate each histogram, i.e., calculate 12 histograms × 2 values (i.e., the ***average*** and the ***standard deviation***) per histogram = 24 values in total.
   - Measure 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.

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.