CSC310 – Assignment #2

Due: Oct.20, 2006, 10am
Hand in to Prof. Roweis in Pratt 290 or send by email to `csc310@cs.toronto.edu`
Worth: 17%
Late assignments not accepted.

Lossless Compression

In this question you will write computer programs to losslessly compress and re-expand files based on Huffman and Arithmetic coding. You can use whatever programming language you like for this assignment, and you do not have to hand in your code. However, as with many of your other programming courses, you are expected to do all of your coding by yourself; sharing code with other students or copying parts of programs you did not write or do not understand is not permitted and will be considered cheating.

- Your programs will read in arbitrary files one byte at a time and thus our source alphabet will be the set $\{0, 1, \ldots, 255\}$. As the probabilities of these symbols, you will use the relative frequencies of the symbols in the file you are encoding, in other words the counts of the number of times each byte occurs divided by the total number of bytes in the file. (Notice that this means many symbol probabilities will be zero; make sure your programs deal correctly with this.) It is important that you keep the symbols (at least the ones with nonzero counts) ordered from 0 to 255 so that we can compare our encodings/decodings with yours.
- Of course, your decoder will also need to know the symbol probabilities. In a real world system these probabilities would themselves have be encoded (somehow) at the beginning of the compressed file or estimated adaptively during decompression. But for this assignment, you can "cheat" and just write the symbol counts to a separate file (different than your encoding file) which the decoder reads before it starts decoding the compressed (encoded) file.
- Your encoding alphabet will be binary bits $\{0, 1\}$. To keep things simple, when you read/write your encoded files you can represent each bit as either the ASCII character '0' or '1', rather than using single bits directly. This means your encoded file will be 8 times larger than it should be, but you can just divide by 8 when computing encoding lengths, etc. in terms of bits. Of course, if you want you can implement code to read/write bits directly, but don't waste too much time hacking this up.
- Here's what you need to code up:
  1. Write a Huffman encoder/decoder which encodes a single symbol (byte) at a time, using the frequency counts in the original file as the probabilities. You will have to build the Huffman tree, figure out the optimal code based on that, and then use that code to compress sequences of bytes into sequences of bits and decompress the other way.
  2. Write a Huffman encoder/decoder which encodes pairs of symbols (two bytes) at a time, i.e. for the second extension of your source. The symbol table you write out in this case is over pairs of symbols.
  3. Write an Arithmetic encoder/decoder which encodes and decodes streams of symbols into streams of bits. To keep things simple, you can use double precision floating point representations of the (cumulative) probabilities and of the interval width r. Since the files you have to compress and decompress have quite small sizes this will be more than sufficient precision for your probabilities which are estimated by counts. Use unsigned long integers (or the equivalent) to represent the interval endpoints u, v and the message t.

Experiments

- Here are the experiments you need to run and the results you need to report:

    1. Using your Huffman coder, your extended Huffman coder and your Arithmetic coder, compress the files `file1, file2, file3, file4` from the course website. Decompress the encoded versions of these files and verify that the decoded result exactly matches the original file. (The unix command `diff` can be useful for this.) We have also provided a `file0` which may be useful for debugging, but you don't need to report any results on this file.
    2. Report the number of bits needed to encode each file with each method (not counting the length of the separate file you used to store the symbol counts).
    3. Compute the average number of bits per symbol and the compression ratio you achieve for each file with each method.
    4. Compute the entropy of each source model (ie the entropy of the set of probabilities you estimated by counting), and state how much worse than the entropy each encoder is on each file.
    5. Report the first, second, last and second-to-last bit for each of the file encodings using each method.
    6. Create a file which is a single line containing your name in all capital letters followed by your student number, with no spaces or punctuation anywhere and no linefeed or carriage return at the end. (In other words, your file should contain only the symbols A-Z and 0-9.) If you need to, add a 0 in front of your student number to make the length of this file an even number. On your assignment type or print this line, as well as the complete encoding of it using all three programs. Also email this line to `csc310@cs.toronto.edu`. It is cheating to ask anyone else to produce this encoding for you.