Homework 05

Assigned: Tue Oct 27 2015
Due: Fri Nov 06 2015

Instructions:

1. This assignment is a mix of programming and written work. Feel free to work with others on this assignment. However, you must acknowledge with whom you worked, and (1) you must write up your own solutions for the written parts and (2) you must develop your own code for the programming parts.

Problem 1 [10 pts]: Lempel-Ziv, by hand.

i. Consider your three initials, lower case. (If you do not have a middle name, use the first two letters from your first name and the first letter from your last name.) For my name, Javed A. Aslam, the three lower case initials are "jaa".

Consult an ASCII table and convert your three initials to their associated bits. At eights bits per ASCII code, you should obtain 24 total bits. For example, given my initials, you would obtain:

```
011010100110000101100001.
```

Parse the 24 bits corresponding to your initials using the Lempel-Ziv parsing algorithm described in class and in the text. For each unique substring found, generate a pair corresponding to the prefix back-reference (i.e., how many substrings back is the prefix?) and suffix bit. For my initials, you would obtain:

```
0, 1, 10, 101, 00, 11, 000, 01, 011, 0000, 1
```

and

```
(0,0), (0,1), (1,0), (1,1), (4,0), (4,1), (2,0), (7,1), (1,1), (3,0), (0,1).
```

Now encode these pairs as described in class. To encode the prefix back-references, use $\lceil \lg n \rceil$ bits to encode the back-reference corresponding to substring n. In other words, use zero bits to encode the first back reference (i.e., don't encode it—you don't need it), use one bit to encode the second back-reference, use two bits to encode the third and fourth back-references, use three bits to encode the fifth through eighth back references, and so on. For the example given above, we obtain:

```
(_,0), (0,1), (01,0), (01,1), (100,0), (100,1), (010,0), (111,1), (0001,1), (0011,0), (0000,1)
```

for a complete encoding of

```
00101001110001001010011110001100110000001.
```

(Note that the encoded length is longer than the unencoded length. Is this a failure of the Lempel-Ziv compression algorithm? No. Lempel-Ziv is guaranteed to compress at nearly the entropy rate for sufficiently long strings; our example is simply too short. Try running `gzip` on a file containing only three characters and see what happens.)

ii. Consider the string

```
00110001101100011011100100111101110001010010011001100100011100100000000
```

This bit string is the result of encoding a piece of text using the Lempel-Ziv algorithm described above (and in class). Parse this bit string to obtain blocks corresponding to (back-reference, suffix bit) pairs. Decode the back-references to obtain these pairs. For example, given the string

```
00101011101001001011101000001101001
```

we would obtain

```
0, 01, 010, 111, 0100, 1001, 0111, 0100, 00011, 01001
```

and

```
(0,0), (0,1), (1,0), (3,1), (2,0), (4,1), (3,1), (2,0), (1,1), (4,1)
```

Reconstruct the original bit string using these (back-reference, suffix bit) pairs. For the example given above, we would obtain:

```
011001100110111101101111.
```

Consult an ASCII table to convert your bit string to a sequence of characters. What string do you obtain? For the above example, we get "foo"; in your case, you should obtain a string of length six which is somehow related to this course.

Problem 2 [60 pts]**:** Lempel-Ziv, in code.

In the language of your choice, implement the one-pass version of Lempel-Ziv described above and in class. You should create two programs, one for compression and one for decompression. Notes:

- Ultimately, your goal is to be able to compress arbitrary binary files; however, it will probably be much simpler for you to deal with character strings of 0s and 1s rather than actual bits. (Packing, unpacking, and appropriately padding streams of bits can be tricky.) Below are links to four executable Perl scripts which can be used to convert bits to strings (and vice versa) before and after `0/1` character compression and decompression.

  preCompress.pl postCompress.pl preDecompress.pl postDecompress.pl

  Assuming that you use a Unix system and implement your `0/1` character `compress` and `decompress` routines in such a way that they read from STDIN and write to STDOUT, you could then compress and decompress arbitray binary files as follows:

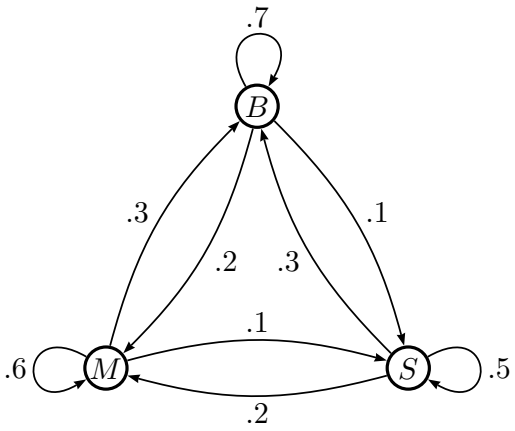
```
   preCompress.pl < file.in | compress | postCompress.pl > file.out
preDecompress.pl < file.out | decompress | postDecompress.pl > file.in
```

- I would suggest using a programming language which has convenient access to a hash data structure—using a hash to keep track of prefix strings will make coding Lempel-Ziv quite simple. (In Perl, for example, I have coded Lempel-Ziv in about a dozen lines.)

You must submit your code together with some example runs (for example, compressing and decompressing character strings).

Problem 3 [30 pts]**:** Markov Sources and Compression.

When I take my two children out to dinner, they often get to choose the restaurant. They have three favorite restaurants: Bertucci's (Italian), Margaritas (Mexican), and Sato (Chinese/Japanese). Over the years, I have noticed that they are much more likely to pick a restaurant that they have eaten at recently than to choose a different restaurant, and I have also noticed that they have clear favorites among these restaurants. The following Markov Chain roughly describes their restaurant choice behavior, where B, M, and S represent Bertucci's, Margaritas, and Sato, respectively:



i. Calculate the stationary distribution associated with this Markov Chain. What fraction of the time (on average) do we eat at Bertucci's, Margaritas, and Sato?

ii. Calculate the entropy of this stationary distribution, and calculate the entropy of the Markov Chain itself.

iii. Generate a "long" i.i.d. sequence of restaurant choices according to the stationary distribution, and generate a "long" sequence of restaurant choices according to the Markov Chain (represent the restaurants using the characters `B`, `M`, and `S`).

Compress these two sequences using the Lempel-Ziv compressor you developed above. What are your compression rates? How do they compare to the entropy rates you calculated above? Explain.

iv. Compress the two sequences above using a "real" compressor such as `gzip`, `bzip2`, etc. What are your compression rates, and how do they compare to the entropy rates and the compression rates you obtained above? Explain.

v. Go to the Project Gutenberg website and obtain a plain text copy of Alice in Wonderland by Lewis Carroll. Strip off the lengthy header text, retaining just the story itself.

Compress this text using the Lempel-Ziv compressor you developed above. What is your compression rate? Now compress this text with a "real" compressor (`gzip`, `bzip2`, etc.); what is your compression rate? Compare and contrast this to the results you obtained in parts iii and iv above. Explain.