Programming Assignment

Due: Thursday, May 11, 2017, by 11:59p

You can use any programming language you prefer (MATLAB, Python, C/C++, or Julia, for example). Write down your code as clearly as possible and add suitable comments. For the submission, please follow the instruction below.

- Summarize the answers concisely in a document of any extension (e.g. hw3-ans.doc, hw3-ans.pdf). If you cannot get an answer because your code does not run, please comment your progress in the answer file.
- Please zip your code and the answer file in one file with the exact name "hw3-Last name-First name.zip".
- Submit your zip file to `ece154ucsd@gmail.com` with the exact subject `ECE 154C (HW3)`.

1. In this problem, we want to combine Tunstall coding with (binary) Huffman coding to find a better variable-to-variable encoding scheme. For `src_pmf` $= \{0.9, 0.1\}$, using the functions `Tunstall(src_pmf, num_phrases)` and `binaryHuffman(pmf)`, find the average length of codewords per average length of symbols for `num_phrases` $= 1$ to 1000, and plot it. Does it converge? If so, what is the limit?

2. *(Difficult.)* Note that Tunstall coding scheme may not give the optimal branching of probabilities. Hence in this problem, we want to find the optimal branching for the given number of phrases. Write a code that compares the Tunstall+Huffman with the (optimal branching)+Huffman in terms of the average length of codewords per average length of symbols, for a given number of phrases.

   (a) For `src_pmf` $= \{0.9, 0.1\}$, find the smallest number of phrases such that the optimal branching is strictly better than Tunstall coding.

   (b) Repeat the experiment for `src_pmf` $= \{0.7, 0.3\}$.

3. In this problem, we will implement the sliding-window Lempel–Ziv (often abbreviated as SWLZ or LZ77) compression algorithm.

   (a) Write a program for a function `MatchLengthPosition(window, text)` that takes a "window" of string (say, English text) `window` and a string (say, a text sample) `text` as inputs. If there is a match, the function returns a flag bit 1, the starting position of the longest match in the window (with the position of the rightmost symbol in the window taken as 0), and the match length (according to the LZ77 parsing). If there is a tie for the longest match, then pick the rightmost position. If there is no match, this function returns a flag bit 0, and the first character of `text`.

   For example,

```
[1,2,6] = MatchLengthPosition(‘‘MY ’’,‘‘MY MY WHAT A HAT IS THAT’’)
```

and

```
[0,‘B’] = MatchLengthPosition(‘‘AAAA’’,‘‘BABBAA’’)
```

(b) Using the function in part (a), write a program for a function `ParseSWLZ(InputText, WindowSize)` that takes a string of characters (say, a sample of English text) `InputText` and a positive integer `WindowSize` as inputs, and returns the encoding by the sliding window Lempel–Ziv algorithm (using a window of size `WindowSize`). For example, if we input `InputText=‘‘MY MY WHAT A HAT IS THAT’’` and `WindowSize` = 16, then the function should output the following.

```
(0,‘M’), (0,‘Y’), (0,‘ ’), (1,2,3), (0,‘W’), (0,‘H’), (0,‘A’),
(0,‘T’), (1,4,1), (1,2,1), (1,1,1), (1,5,4), (0,‘I’), (0,‘S’),
                  (1,2,1), (1,4,1), (1,7,3)
```

(c) Write a program for a function `LengthEncoding(length)` that takes any positive integer `length` as input and outputs the prefix-free binary encoding of `length` according to the scheme discussed in class. For example,

```
110010 = LengthEncoding(5)
```

and

```
11110010 = LengthEncoding(13)
```

(d) Using the functions above, write a program for a function `SWLZ(InputText, WindowSize)` that takes the same input as before, but returns the binary encoding of the string according to the sliding window Lempel–Ziv algorithm (using a window of size `WindowSize`) and the compression ratio. For the same input as before, `InputText=‘‘MY MY WHAT A HAT IS THAT’’` and `WindowSize` = 16, it should output

```
0100110101011001001000001001011100101011101001000010000010101010010100
0010100101010001101010111000101001001010100111001010101001010111111 0
```

(e) Load the file `sample_text.txt` into a string `s`. For example, in MATLAB, this can be done using the following code.

```
FID = fopen(‘sample_text.txt’, ‘rb’);
  s = fread(FID, [1, inf], ‘char’);
          fclose(FID);
```

Use the function `SWLZ` to compress `s` using the sliding-window Lempel–Ziv algorithm, and find the compression ratios (in number of bits per symbol) for window sizes 256, 512, 1024, 2048, and 4096. What do you observe? (You do not have to show the full binary encoding of the text.)