Problem 1

Suppose you use the RLE109 algorithm, as defined in lecture last Friday (lecture slides may be found here: http://www.cs.bu.edu/fac/snyder/cs109/Home.html). Encode the following messages into binary in uncompressed or compressed form as indicated.

- A_FED into uncompressed form: 000 111 100 011 010
- AAADEEEE into compressed form: 3A1D4E = 011 000 001 010 100 011
- AAAAAAAAAAAAAAADD into compressed form: 15A2D = 000 001 111 000 010 010

Problem 2

Again, using the RLE109 algorithm, decode the following messages:

- 010 110 011 011 = 2N3E = NNEEE
- 000 001 000 011 001 100 = 8E1F = EEEEEEEF

Problem 3

In lecture we discussed how a code such a LRE109 could make a binary “message” shorter (hopefully!) or longer (hopefully not!). It could also change the encoding but leave it at the same length.

A. Give an example of a message that is unchanged in length after encoding by the LRE109 method (hint: some part of the message will get longer and the other part will get shorter, balancing out in the end to the same number of bits as the original).

(A) There are many possibilities, among which the simplest is probably “2A” which translates to 010 000, as opposed to “AA” which translates to 000 000, the same length. If you use my hint, then you might come up with something like: “ABBB” = “1A3B” = 001 000 011 001 (12 bits) as opposed to “ABBB” = 000 001 001 001 (12 bits).
B. What is the best possible compression ratio (i.e., the smallest ratio (# bits after compression)/(#bits before compression)) you could expect to get for any message using the LRE109 compression algorithm? Give an example of a message that has this optimal compression ratio. (Hint: Consider a message that involves multiple occurrences of only a single letter.)

(B) The best compression ratio occurs when you have a small compressed message and a long uncompressed message. The best you can do is with the additional feature of RLE109 which allowed you to encode frequencies up to 6 bits. So if you encode all A’s using this code, you would have

```
000111111000
```

which is 12 bits, encoding “63A” or

```
AAAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAAAA AAAAAAAAAAAA
```

This message is $63 \times 3 = 189$ bits, giving a compression ratio of $12/189 = 0.0635$, or about 6%. (You did not have to give the compression ratio.)

Problem 4

Using the following Huffman Code from lecture on Wednesday 10/29:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
</tr>
</tbody>
</table>

give the binary encoding of the following message:

```
ABCCAAB
```

and then decode the following message:

```
101011010001011011
```

Solution: The encoding of “ABCCAAB” is $0 \ 10 \ 11 \ 11 \ 0 \ 0 \ 10$, and the encoded message $10 \ 10 \ 11 \ 0 \ 10 \ 0 \ 0 \ 10 \ 11 \ 0 \ 11$ is “B B C A B A A B C A C”

(spaces are for readability and not part of the original message).
Problem 5

Suppose our alphabet is A, B, C, D, with probabilities as follows: A = .5, B = .3, C = .1, and D = .1. Along the same lines as the presentation in lecture on Friday, construct the Huffman Tree and the resulting Huffman Code, and then encode the message “BADDAD”. Show all work. Finally, calculate the average number of bits that this code would use for messages that have these probabilities, and give the corresponding compression ratio.

Solution: Combine C and D to get a sum probability of .2, then (CD) and B to get a sum of .5, then combine with A:

A  0
B  10
C  110
D  111

(Code for C and D could be exchanged.)

“BADDAD” = 10 0 111 111 0 111 (spaces are not part of the message)

The average number of bits for a message following these probabilities would be 0.5*1 + 0.3*2 + 0.1*3 + 0.1*3 = 1.7 bits. The compression ratio, compared with the uncompressed code of 2 bits ($2^2 = 4$) would be $1.7/2 = 0.85 = 85%$. 