Weekly Programming Quiz 9: Huffman Encoding: The lie, and the truth

CS 5002
Released: December 2, 2017
Due: December 8, 2017; Midnight

1 Quiz Objectives

The objectives of this quiz are to:

- Reflect on Huffman
- Make a data-driven business argument
- Develop your intuition about type and struct sizes
- Practice instrumenting your code so you can reason about it

2 Reflecting on Huffman

With Quiz 8, we made the argument that by encoding characters with a variable number of bits that depend on which characters (and how frequent those characters) are in the original message, we could minimize the number of bits it takes to send that message "over the wire" (that is, to a recipient). In this Quiz, you're going to investigate and quantify those claims.

Specifically, we noted that with Huffman encoding, in addition to sending the encoded message, you also need to somehow provide the recipient with the HTree you used to encode the message, so they can decode the message. In this quiz, we're going to explore the efficiency savings we can realize.

At the end of this writeup are tables to collect data for 3 different strings. Run your `encode` function on these inputs, and fill in the tables with the indicated values. To collect the data, you can attempt to calculate it by hand, or modify your code (described later) to help with your measurements.

Note: When you call `sizeof` in C, it returns the number of *bytes* to store that type. There are 8 bits in each byte. We are working primarily with *bits* below.

Here is a brief explanation of each data point you are asked to collect:

1. Number of distinct characters:
   The number of unique characters in the input message. This is the number of entries in your frequency table that are not 0.
2. Length of message (in characters):
   The number of characters in the original message passed to `encode`.
3. **Ratio 1:** Length of message / number of distinct characters:
   This ratio provides an indication of how many times each character occurs.

4. Number of bits (Length of message * 8 bits/character):
   This is the number of bits it takes to encode a message with our traditional 8 bits/character approach.
5. Length of encoded bits (length of final `BitSeq`):
   This is the number of bits of the final Huffman-encoded message.
6. **Ratio 2:** Encoded bits / number of bits:
   This ratio indicates the efficiency improvement in terms of number of bits needed to transmit a message.
7. Number of nodes in HTree:
   The number of nodes in your final HTree used to encode your message.
8. Size of HTreeNode (in bits):
   The size of each HTreeNode in your implementation.
9. Size of HTree (in bits):
   This number of nodes * the size of HTreeNode, to calculate the overall number of bits needed to store an HTree.
10. **Ratio 3:** (Size of HTree + encoded bits) / number of bits:
    This is the number of bits to send both the HTree *AND* the encoded message to the recipient, divided by the number of bit required to send the original message.

2.1 How to do this

To generate the data for the above questions, you will likely want to modify your original Huffman code from Quiz 8. To do this:

- Copy all of your code from the `quiz8` folder and put it in a `quiz9` folder (in your repo).
- Change the relevant file names and code to `quiz9`. This includes inside the `Makefile` and making sure the .c files now include the correct .h files.
- Run your code to make sure nothing is broken.
- Make your modifications.
- Continue using the tests until you're confident that you have not broken your code by adding the instrumentation. After that, you can stop running the tests and instead run your `encode` function to generate the data for the strings specified below. Ask Instructors if you're uncertain about how to do this.

It might be helpful to create a `Report` struct that has fields for every measurement you want to capture. Fill in the appropriate fields during the encode process, and when it's complete, print the report to the console.

2.2 What to turn in

The following items must be in a folder named '`quiz9`' in your Github repo:

- Your modified quiz8 code (which is now quiz9):
  - `quiz9.c`
  - `quiz9.h`
  - `Quiz9.pdf`

  - `quiz9_test.c`
  - `Makefile`

- A modified copy of this .pdf file, named `quiz9_solutions.pdf`, with the answers filled in.
- A `README.md` file with a description of the code changes you made.

Input Message: Silly Sarah was super surprised.	
1. Number of distinct characters:	15
2. Length of message (in characters):	32
3. **Ratio 1**: Length of message / number of distinct characters:	2.13
4. Number of bits (Length of message * 8 bits/character):	256
5. Length of encoded bits (length of final `BitSeq`):	120
6. **Ratio 2** Encoded bits / number of bits:	0.47
7. Number of nodes in HTree:	29
8. Size of HTreeNode (in bits):	192
9. Size of HTree (in bits):	5568
10. **Ratio 3** (Size of HTree + encoded bits) / number of bits:	22.22

Input Message: The quick brown fox jumped over the lazy dog. The quick brown fox jumped over the lazy dog. The quick brown fox jumped over the lazy dog. The quick brown fox jumped over the lazy dog. The quick brown fox jumped over the lazy dog.	
1. Number of distinct characters:	28
2. Length of message (in characters):	229
3. **Ratio 1**: Length of message / number of distinct characters:	8.18
4. Number of bits (Length of message * 8 bits/character):	1832
5. Length of encoded bits (length of final `BitSeq`):	1023
6. **Ratio 2** Encoded bits / number of bits:	0.56
7. Number of nodes in HTree:	55
8. Size of HTreeNode (in bits):	192
9. Size of HTree (in bits):	10560
10. **Ratio 3** (Size of HTree + encoded bits) / number of bits:	6.32

Input Message: Walter, you can't do that. These guys're like me, they're pacifists. Smokey was a conscientious objector. You got the wrong guy. I'm the Dude, man. I'm not Mr. Lebowski; you're Mr. Lebowski. I'm the Dude. I got a nice quiet beach community here, and I aim to keep it nice and quiet. Vee belief in nossing. Okay. Vee take ze money you haf on you und vee call it eefen.	
1. Number of distinct characters:	40
2. Length of message (in characters):	367
3. **Ratio 1**: Length of message / number of distinct characters:	9.18
4. Number of bits (Length of message * 8 bits/character):	2936
5. Length of encoded bits (length of final `BitSeq`):	1648
6. **Ratio 2** Encoded bits / number of bits:	0.56
7. Number of nodes in HTree:	79
8. Size of HTreeNode (in bits):	192
9. Size of HTree (in bits):	15168
10. **Ratio 3** (Size of HTree + encoded bits) / number of bits:	5.73

3 Reflection Questions

1. Which string provided you with the best naive improvement (that is, the best Ratio 2)?

   The First String with ratio2 being 0.47.

2. Which string provided you with the best Ratio 3?

   The Third String with ratio3 being 1.20.

3. What are the most important factors in improving Ratio 3?

   From the formula, we can see that the most important factor is the encoded bits.

4. What suggestion would you make to make Huffman encoding a reasonable approach for message transmission?

   The effectiveness of Huffman encoding will not appear until the characters of the input grows to a certain large amount, just because the huffman tree itself takes some memery, as we can see the ratio3, as the input grows bigger, the ratio is going to be smaller, once it becomes smaller than 1, until then, the advantages of huffman encoding appears.