Name: Trevor Buck

ID: 109081318

CSCI 3104, Algorithms **Profs. Grochow & Layer**
Explain-It-Back 5 **Spring 2019, CU-Boulder**

At a recent infections disease seminar, you hear about how the speaker is sequencing millions of *Plasmodium falciparum* genomes (the human malaria parasite) in order to better characterize how patients respond to treatment. In the presentation, the speaker complained to the audience that although they are making the data sets a small as possible by only using 2 bits to encode the 4 nucleotides of the genomes their IT department is still struggling to store all the data. Later in the presentation, you learn that the Plasmodium falciparum genome is "AT-rich." That is, over 80% of the nucleotides in the genome are either A or T. Please help this team understand how they can leverage Plasmodium falciparum's AT-richness to help their IT department deal with the influx of data.

To 'the speaker'

I just listened to your presentation, and I wanted to thank you! You did a great job, and I learned quite a bit. However, I am writing this letter for something more substantial. I couldn't help but notice that you seemed a bit bothered by the lack of data storage. You mentioned that even with a set, 2-bit encoding method, your processors are still filling up faster than you would like. This intrigued me because I am a Computer Science major, and I spend my time learning how to improve situations like these. I had a few ideas in mind, but once I heard about your 'AT-rich' environment, I knew immediately the best way to help you. You would benefit from something called Huffman Encoding.

Huffman encoding is designed to reduce the total amount of data that you are storing. Notice that I said the words total amount. I say this because Huffman encoding involves reducing the amount of coding bits on the largest portion of your data, with the offset of adding a few coding bits to the smaller portion of data. In your case, we would be changing the genome with the most recurrences to only one bit, and then adding a bit to the genomes with a smaller number of recurrences.

I took a few notes and you said that %80 of your genomes are either A or T. For arguments sake, let's pretend that 50% of your genomes are A, and 30% of your genomes are T. This leaves 20% for the remaining two genomes. Now let's consider a sample size of exactly 100 genomes. (ie. 50 genomes of type A, 30 genomes of type T, and 20 genomes of type X or Y). If we were to use the coding that you are currently implementing, that would require 200 -> (100 * 2) bits.

Now let's look at Huffman encoding. To start, it would take the genome with the highest frequency and give it the value of the bit '1'. The next highest frequency would be given '01'. The next highest frequency would be given '001'. And so on and so forth until you get to the last bit. Here we can use a short cut and give the last bit the value of '000...000' with (n-1) zeros. In simpler terms, it would have the same number of bits as the second to last genome, but with a zero at the end instead of a one. In our case, genome X would be '001' and 'Y', the last genome, would be '000'.

Name: Trevor Buck

ID: 109081318

CSCI 3104, Algorithms **Profs. Grochow & Layer**
Explain-It-Back 5 **Spring 2019, CU-Boulder**

This would mean that A would be encoded with '1', T would be encoded with '01' and, as stated, X and Y would be '001' and '000'. If we look at the total required bits now, it would be 170 -> ((50 * 1) + (30 * 2) + (20 * 3)). This saves you 30 bits for every 100 genomes.

After reading this, you might wonder how we would separate the bits. Well, every time you hit a '1', that would mark the end of that genome. The only exception to this would be if you hit three 0's in a row. That would mean it was the last genome. I have included an example of what that might look like below.

Genomes -> A A T A T X T Y T T X A X A T A A T A A
Binary Code -> 00000100011001110101100010000010000010000
Huffman Code -> 10101101001011000010110011010110111
(Interpertation) -> 1 1 01 1 01 001 01 1 000 01 01 1 001 1 01 1 1 01 1 1

Notes: A. Each bit ends with a 1 in the Huffman code
B. The only exception is when we have 3 0's which means gene X
C. The Huffman code saved 7 bits on 20 genomes.

I really hope that this helps. It might be a little confusing at first, but it really could save you some time and storage. Plus, if you have any questions, I am more than willing to help! Huffman encoding is a valuable tool and it is perfect for this exact situation.

Thank you for your time,

Trevor Buck