Advanced Algorithm HW5

Seungho Jang

September 2020

1 Q1

When we use prefix codes to encode text, we can simply concatenate the codewords, i.e. we don't need a symbol to separate the codewords. Why doesn't this cause problems when decoding?

The reason is because of the definition of the **prefix property**. Prefix code can be defined in the following way.

- Mathematical Definition : $a_1, a_2...an \in A^*$ is a prefix of $b_1, b_2...b_m$ if $n \leq m$ and $a_1, a_2...an = b_1, b_2...b_n$ where A^* is a string.
- c is a prefix code if no codeward is a prefix of another codeward.

Then, what is not prefix code, if we were to have this code-set (0, 1, 01, 010), this can't be prefix code because of "0", "01", "010" due to each elements having same prefixed code. To note, if the two element in the code-set is shared, this is not a problem because as long as one value is not entirely used as a prefix of another value, then it is fine.

For example, if we were to decode this codeward(101110110) to codeset(a:0, b:10, c:110, d:111), then we can simply separate the codeward to each codeset such as (10 111 10 110) by using definition of prefix code, then output of decoding process will be bdac.

2 Q2

The assumption of this quiz was that a data file of 100,000 characters contains only the characters a - f with the frequencies and assign each character as 3-bit codeword.

Based on the table in the quiz, if you were to calculate the bits to code the entire file, then it's going to be this $(45 * 3 + 13 * 3 + 12 * 3 + 16 * 3 + 9 * 4 + 5 * 4) * 1,000 = 300,000$ bits to represents the file. However, if we were to calculate the variable length code, then the calculation will be (45*1 + 13*3 + 12*3 + 16*3 + 9*4 + 5*4)*1000 = 224,000 to represent the file. To calculate the how much percentage it was improved by using variable length code, $(1 - \frac{224,000}{300,000}) * 100 \approx 25.33\%$. This means that 25.33% increased by using variable length code as compared to fixed length code.

3 Q3

One of the question could be that if the elements in the table has equal frequency, such as a and e has same frequency in my name as shown in the table below. One of the reference was taken from this link : "What if the element has two equal frequency in Huffman encoding". What it says, Huffman algorithm is not deterministic. This Huffman algorithm will create the different result, however the requirement of Huffman algorithm is whether or not Huffman code can be properly transmitted along with the coded data in order to finish encoding and decoding process. The one characteristic that is not non-deterministic character of Huffman algorithm is that the set of frequencies for the symbol is not a sufficient descriptor of a Huffman Code. This shows that the answers from Huffman code with the same frequencies are fine, and this topics brings optimality of Huffman codinng

Name Encoding - Huffman Code									
letter	a	e	g	h	j	n	o	s	u
freq	1	1	2	1	1	2	1	1	1

Table 1: Table for Activity Selection Problem

4 Q4

This problem is similar to question 3 in regarding some of element having same frequency in the text. One of the important thing in here is that Huffman code would multiply two equal frequencies times the length, put it into priority queue in iteration, Then, this result of total number of bits to be coded would be the same in the root. Therefore, no matter what the order of element which has equal frequency are being multiplied, Huffman algorithm will chose the smallest sub-trees greedily. So then the output will always be the letter with the highest frequency in the shortest binary digits which is close to root of the Huffman Tree. So the answers can be varied depending which element is selected within same frequency.

Optional Testing : The code was tested with string of 'abccdddeeeeeffffffffgggggggggggggghhhhhhhhhhhhhhhhhhhhh', and testing can be found in this link : Q4-Testing Code

So the answer can be different depending on which letter was selected within equal frequency, such as a can be 1111111 or 1111110 vice versa for b.

However, in terms of efficiency of the Huffman coding, encoding and decoding process of Fibonacci number would be worst case scenario because the tree will be skewed tree, such that the tree will be unbalanced: internal node has at least one leaf descendent.

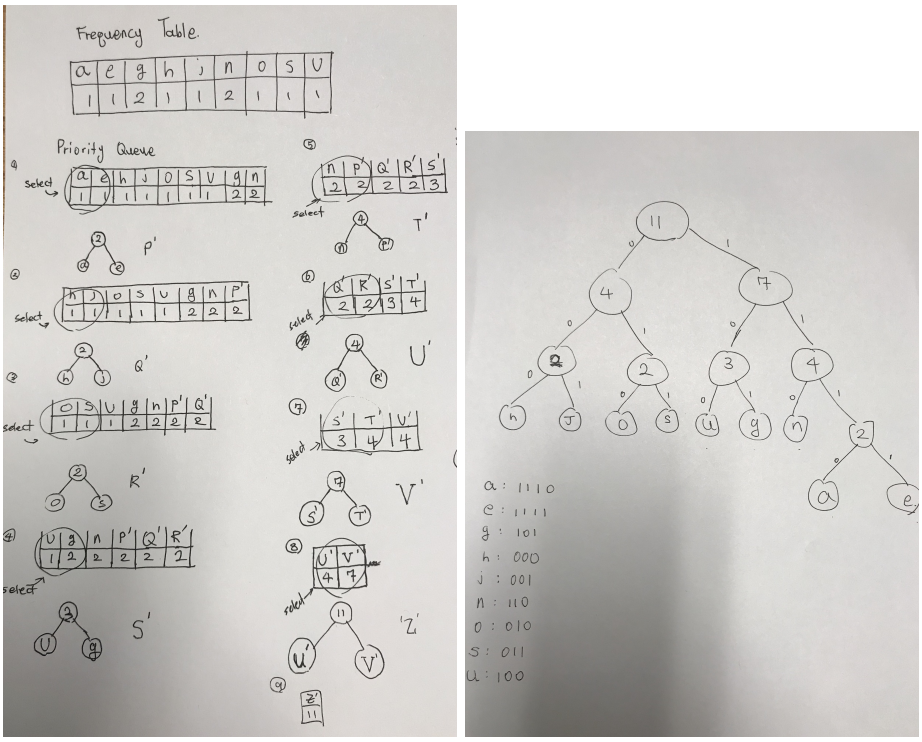

Figure 1: Huffman Algorithm and Huffman Tree

5 Q5

The code can be found in this link: Application of Huffman Coding in Reddit

To check whether the code-wards are making sense from output from Huffman Coding, simply manually check the specific letter's frequency from output of frequency table generated from Huffman code. Two of the characters found to be the least frequent characters were '-' and ' '—', their frequency from the Huffman code was 1. Then, compare with that frequency to "manually count that character" was 1 as well.

Also, the element that has the highest frequency was 'a', which was 11340 from frequency table of Huffman code. The number of counting 'a' was 11340 as well. So we can validate that the Huffman code makes sense.

The other thing to note here is to check optimality of Huffman code, which can be prove that find each code-wards length and sum of $2^{-}(length)$. Then, this sum has to be 1. For our problem, the result was 1.

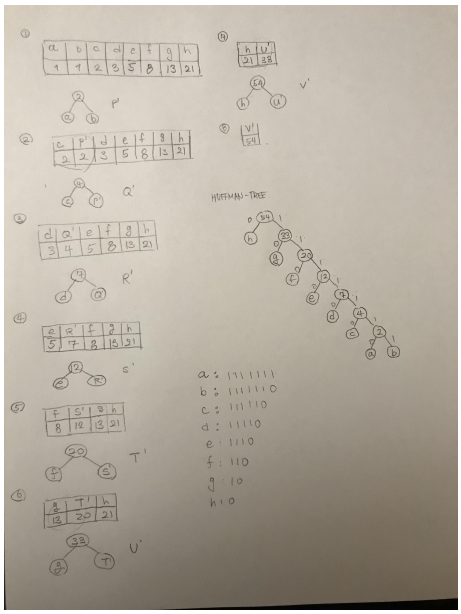

Figure 2: Huffman Algorithm and Huffman Tree