CS260 - Data Structure
Programming Assignment 4: Huffman Coding
Assignment Report

Fitsum Alebachew fa496

Question 1 : 8 points

Encoder: Answer the question below regarding the encoder program you have written:

(a) [2 points] Provide a big picture of your encoder program by briefly listing the main steps you took.

> **Solution:** The encoder first initializes an empty huffman code table and fills it with character and frequency values. That table is then used to create a heap that includes node pointers storing those values plus pointers to left and right nodes. Then the heap is used to create the huffman tree by manipulating the left and right node pointers to point to eachother in a specific way. Then the huffman code table is updated to include the binary value representation of each character in it.

(b) [2 points] Provide a list of the data structures you used in your program and explain the role each played in the program.

> **Solution:**
> Hash Table: The code table stores the character, frequency and binary values.
> Heap: The heap is the intermediary step towards creating the huffman tree.
> Binary tree: The huffman tree is how the binary values are calculated by traversing along it.

(c) [2 points] Once you built the Huffman tree, how did you generate the code table? In plain English, explain the algorithm you used to generate the code table.

> **Solution:** By using a recursive algorithm that traverses the tree and finds every leaf storing the characters, it uses a string and appends either 0 or 1 until the leaf is found, then updates the code table for the corresponding character.

(d) [2 points] Given an input of n characters long string, make an asymptotic runtime analysis of your encoder program.

> **Solution:** Initially inserting the characters with the frequencies takes $O(n)$ time. Inserting each character into the heap takes $O(n \log_2 128) = O(n)$ time, since the heap has a constant max size. Creating the binary tree also takes $O(n)$ time since getmin

and insert are used until the heap is only left with one element. The total runtime is thus $O(n)$.

Question 2 : 6 points

Decoder: Answer the question below regarding the decoder program you have written:

(a) [2 points] Provide a big picture of your decoder program by briefly listing the main steps you took.

Solution: The decoder runs through the code table line by line and recreates the huffman tree by using the binary values of the characters to creates nodes and make a path to the leaf from the root node. Then it uses that tree and traverses along it by reading the input file as direction until leaves are reached, then it prints it to the output file and resets to the root node, and so forth.

(b) [2 points] Did you need to use the binary codes that were stored inside your code table for decoding? If yes, explain how you used them for decoding. If you did not use it, how did you manage to decode the encoded string using by using the code table?

Solution: Yes. They were essential in recreating the code table in my implementation. They acted as the guide to recreating the huffman tree. The 0s and 1s told the algorithm to go either left or right while recreating the huffman tree.

(c) [2 points] Given an input of n characters long string, make an asymptotic runtime analysis of your decoder program.

Solution: Recreating the tree takes $O(1)$ time since it doesn't depend on the input string, but the number of unique characters which can only be as high as 128. Then traversing the tree takes $O(1)$ time for each binary value read, making it a total of $O(n)$ time.

Question 3 : 6 points

Overall:

(a) [3 points] Do you think your program could have been improved? If so, how? Would you change a data structure or an algorithm, or both? What would improve: runtime, memory usage, both?

Solution: Although the encoder can encode the newline character, it is printed as a new line in the code table as well, which will interfere with the decoding process. The code can be edited to accept new line characters by printing a unique identifier in the code table and also checking for that in the decoder.

(b) [3 points] There could be several code tables for compressing the same data while having the same compression ratio. How is that possible? Explain, possibly by giving an example.

Solution: How the heap is ordered is what matters. Since the heap uses the character frequencies to extract min value, different values could be extracted when doing getmin depending on how the heap is initialized. Although they might have different binary values, the sum of the binary values of characters with the same frequency will be the same, preserving the compression ratio.