Amber Liu
Al7bf
11/29/2018
postlab10.pdf

Implementation

My implementation of the Huffman encoding program read in a normal file and stored unique characters and their frequencies. I utilized multiple data structures, such as a heap and a tree composed of nodes. The heap structure was a vector of huffmanNodes, which stored int frequency, char character, string prefix, and left and right node pointer. The vector is really useful because it is an array based data structure and can easily add and remove. For a tree it is good to be able to find an element by tracing down from the root and vector can map to where elements are. Then I used an array for the ascii characters with a length of 128 to store the frequency for every character. Because 128 is a set size for ascii and I know that I wouldn't need to resize it, I used array. As for decoding, it was less difficult than the encoding. I reused the huffmanNode and constructed another tree for the encoded message.

Efficiency Analysis

Compression Encoding

array of frequencies: Integer array with size of 128 to hold the frequencies of ascii characters occurring has the worst case run time of O(n) since finding a value requires linear operation through each of the elements

setPrefix():O(n) because it needs to find the character to set, which is linear, and then assign appropriate prefix for it.

creating the tree: multiple calls on insert() is required as the frequency rates may differ, has time complexity of O(log(n))

printPrefix(): O(n) since the function would print every element in the vector linear worst case.

Worst case space complexity with use of sizeof function:
Node = 13 bytes
Int Frequency = 4 byte
Char character = 1 byte
Left right pointer = 8 byte
Frequency (int array with size of 128) = 512 byte
Heap = node * size = 13 byte * size
Total space: 13 byte * size + 512 + 8+ 1 + 4+ 13

Decompression Decoding

Reading in file: reads every character until the end, linear time O(n)

createTree(): search for the encoded text and create node based on value of the bit. Then added to the root. Time complexity of O(n)

Worst case space complexity with use of sizeof function:
(left, right) Node = 13 bytes
Ascii array of char [256] = 256 byte
Total space: 256 + 13 bytes