ASSIGNMENT 7 DESIGN DOCUMENT

Pehara Vidangamachchi

November 27th 2022

Ethan L. Miller

About:

This assignment aims to create my implementation of a Huffman encoder and decoder. This will be done through the use of Nodes, Priority Queues, Codes, I/O, Stacks and finally a Huffman coding module to make the code usable. This program essentially does the same thing as Huffman's approach to optimal static encoding for information. The program uses the key idea defined as entropy which measures the amount of information within an array. By following the directions given in the assignment pdf I am able to replicate a Huffman-style encoder and decoder. With my code, I am to compress data without any loss of information in a fluid and efficient process.

Design Process for code.c

For this file, the task is to implement a working code program that helps maintain a stack of bits while travelling through the tree and creating symbols for the code. This ADT represents a stack for bit and follows a similar interface to that of bit vector from our last assignment. We are given the structure in the document and should use it to create the ADT. Using the information given within the assignment document I am able to implement my version of the code program that works in unison with the rest of the code.

#including any necessary header files

Create a constructor for the code function:

Create a new code on the stack

Set the top to zero and zero out the array of bits

Return the code once initialised

Create a function that returns the size of the code:

Return the size of the code which is equal to the number of bits pushed onto code

Create a function to check to see if the code is empty or not:

If the code is empty return true

Else return false

Create a function to check if the code is full:

If the code is full return true

Else return false

Create a function that clears bits:

Clear the bit index at i by setting it equal to zero

If i is out of range return false

Else return true

Create a function that gets the bits at an index i:

If i is out of range or i is equal to zero return false

If i is equal to 1 return true

Create a push bit onto the code function:

Set the value of the bit equal to the pushed bit

If the code is full before pushing return false

Else return true

Create a code pop-bit function:

Pop off a bit from the code and pass it back to the bit

If the code is empty before popping return false

Else return true

Create a print statement for code:

Create a debug statement for code to check for correctness

Design Process for huffman.c

This segment of code contains the interface for the Huffman coding module. This assists with allowing the functions to come together and execute their given task. The task at hand is assisted by the assignment pdf directions on how to go about efficiently performing the given tasks. My Huffman file contains the main structure behind each part to Huffman style encoding and decoding. It contains the mixture of all the other functions combing into one unit that makes up the main meat behind this assignment.

#including any necessary header files

Create a constructor for the node that builds a tree:

Creates a histogram that has alphabet indices

Each root of the node is returned

Create a constructor for codes:

Create symbols for the nodes in the Huffman tree

Add the constructed codes to the code table

Create a dump tree function:

Check for the root of the Huffman tree and return it to the outfile

Set a character equal to the character plus the byte of the symbol

Create a rebuild tree function:

Reconstruct the Huffman tree by checking through every node

Get the length of the bytes from nbytes

Return the root node from the constructed tree

Create a delete tree function:

Clear the tree and set it equal to NULL

Design Process for io.c

For this portion of the assignment, we are tasked with the creation of I/O functions that previously we used from the stdio.h library. Functions that were commonly accessible through other means will now how the implementation on our own. The I/O functions will then later be utilized by our Huffman encoder and decoder. Following the directions given on the assignment pdf, I am able to translate what's given into my code to create a usable I/O function to help with my encode and decode process.

#including any necessary header files

Create an io function that reads my bytes:

Create a while loop:

Continue running the read function until all the bytes have been read

Return the number of bytes read once there is none left to read

Create an io function that writes my bytes:

Create a while loop:

Continue running the write function until all the bytes have been written

Return the number of bytes written once there is none left to write

Create a read bits function:

Create a static buffer

Block the number of bytes when needed

Return false if there are no more bits

Else return true if there are still readable bits

Create a write code function:

Create a while loop:

Continue running the read function until all the code has been written

Make sure each bit in the code is buffered into the buffer

When the buffer is full of bits inform the outfile

Create a function to flush the code:

If there are any leftover bits set them equal to zero

Design Process for node.c

This segment of code contains my implementation of nodes. The Huffman algorithm has a heavy reliance on nodes, it consists of multitudes of nodes that each contain a pointer to its child. Each pointer behaves in a similar pattern going towards either its left or right child. The definition for our node is given in the header file, so using that and the assignment document I am able to create an ADT that satisfies this assignment's requirements.

#including any necessary header files

Include any structure if needed

Create a constructor for the node:

Set the nodes symbol as a symbol

Set the nodes frequency as the frequency

Create a destructor for the node:

Clear the node and set it equal to NULL

Create a join node function:

Allow the left child node and right child node to combine

Return a pointer creating a parent node

Set the parent's nodes from left child to left and right child to the right

Set frequency equal to the sum of the left and right child's frequency

Create a node print function:

Create a print statement to help debug the code

Design Process for pq.c

This segment of code creates priority queue nodes, which perform similarly to a regular queue but with the added step of assigning each element as a priority so the order shifts. Elements with a higher priority are dequeued from their corresponding counterparts with a lower priority. In the assignment pdf, we are given several different ways to compute this process, ranging from picking an insertion sort to using a minimum heap. So with the use of my functions from assignment four and the directions given on the assignment pdf, I am able to implement my version of a priority queue algorithm.

#including any necessary header files

Create the constructor for the priority queue:

Specify the maxim for the priority queues capacity

Create the destructor for the priority queue:

Clear the priority queue and set it equal to NULL

Create a priority queue empty function:

Check for if the priority queue is empty

Return true

Else return false

Create a priority queue full function:

Check for if the priority queue is full

Return true

Else return false if it's empty

Create a priority queue size function:

Return the number of items in the priority queue currently

Create an enqueue function for a node ion priority queue:

Enqueue a node into the priority queue

Return false if the queue is full

Return true if the queue is not full

Create a deque node for the priority queue:

Dequeue a node from the priority queue

Set the dequeued node to the highest priority node

In the case that the priority queue is empty return false

In the case, the priority queue is not empty return a true

Create a print function for the priority queue:

Create a debug print statement that prints the queue

Design Process for stack.c

This segment of code contains my implementation of stack which is necessary to reconstruct our Huffman tree. The stack functions will help with organising the nodes and allowing them to properly order themselves. By following the assignment document I am able to implement my version of the stack programs to help me create a functioning encoder and decoder.

#including any necessary header files

Create a constructor for the stack:

Set the max amount of nodes the stack can hold equal to the capacity

Create a destructor for the stack:

Clear the stack and set it equal to NULL

Create a stack empty function:

If the stack is empty return true

Else return false

Create stack full function:

If the stack is full return true

Else return false

Create a stack-size function

Return the number of nodes within a stack

Create a stack push function:

Push a given node onto the stack

Create an if statement so that if the stack is full beforehand

It returns true

Else it will push false

Create a stack pop function:

Pop off a node from the stack and pass it back

Return false when the stack is empty

Else return true if the stack is not empty

Create a stack print function:

Create a debug print statement for the stack that prints its contents.

Design Process for decode.c

For the Huffman decoder, the program am able to read a compressed input file and decompress it and expand it back to the original size without any loss. For this assignment, the algorithm am able to read the dumped tree from the input file and then use that to read the rest of the input file bit by bit which finally leads back to the original file. With the directions given in the pdf, I believe I can implement an algorithm that follows the process defined in an efficient manner.

#including any necessary header files

Create the main decode program:

Initialize any variables that will be used

Parse through the options for the decode using case statements

Read the header given in the infile

If the magic number is present continue

Else, an invalid file passed through the program

Exit the program

Set the permissions using fchmod

Get the size of the dumped tree and reconstruct the tree

Read the infile again one bit at a time

If the code arrives at a leaf node

Write the leaf node to the outfile

Reset the current node back afterwards

End the code when the decoded symbols match the original file

Close the files

Design Process for encode.c

This segment of code functions as the main program which encodes the file so it becomes compressed. My implementation of this code must be able to support a help message, a specific input file, an output file and available statistics for compression. The program will first create a histogram, followed by the construction of a Huffman tree, then a code table, steep through each symbol and emit its corresponding code. With the assignment pdf directions, I am able to follow the directions and create a working code.

#including any necessary header files

Create the main encode program:

Initialize any variables that will be used

Create a while loop for each case:

Parse through the options for the encoding using case statements

Open the files given to the program

Read the input file

Construct a histogram

If the symbols count is zero and if the symbols count is one:

Return true and continue with the code

Construct the Huffman tree using the build tree function

Construct the code table from travelling the Huffman tree

Construct a header file that goes towards the outfile

Write all the constructed information in the outfile

Start at the beginning of the infile and write its corresponding code

Close the infile and outfile