Assignment 6 DESIGN.pdf
Yash Sharma

Description of Program:

This assignment requires us to implement a Huffman encoder, decoder, node, priority queue, ADT, i/o module, and a stack structure. Within Huffman Data Compression, we must utilize optimal static encoding to assign the least amount of bits to a common symbol and the greatest number of bits to the least common symbol for an efficient Huffman data compression file.

Files to be included in directory “asgn6”:

1. encode.c:
    - This file contains the implementations of the Huffman encoder.
2. decode.c:
    - This file contains the implementation of the Huffman decoder.
3. defines.h:
    - This file contains the interface for the macro definitions in this assignment.
4. header.h:
    - This file contains the struct definition for a file header.
5. node.h:
    - This file contains the node ADT interface library.
6. node.c:
    - This file contains the implementation of the node ADT.
7. pq.h:
    - This file contains the interface for the priority queue ADT interface.
8. pq.c:
    - This file contains the implementation of the priority queue ADT.
9. code.h:
    - This file contains the interface for the code ADT interface.
10. code.c:
    - This file contains the implementation of the code ADT.
11. io.h:
    - This file contains the I/O module interface.
12. io.c:
    - This file contains the implementation of the I/O module.
13. stack.h:

- This file contains the stack ADT interface library.

14. stack.c:
    - This file contains the implementation for the stack ADT library.
15. huffman.h:
    - This file contains the interface for the Huffman coding module.
16. huffman.c:
    - This file contains the implementation of the Huffman coding module interface.

10. Makefile:
    - A file that formats program into clang-format and compiles it into program executables with make/make all from Makefile.
    - Additionally, make clean from Makefile must remove compiler-generated files (such as the executables)
11. README.md:
    - This file describes how to use the program and MakeFile. It will also list and explain any command-line options the program accepts.
12. DESIGN.pdf:
    - This describes the design for the program thoroughly with pseudocode and descriptions.

Pseudocode / Structure:

Encode.c

Include header files

Setup get-opt cases

Create switch cases for h, infile, outfile, and v

Calculate and print compression statistics to stderr

Create a histogram that checks the frequency distributions of all symbols in the file

Call priority queue to construct the Huffman tree

Call the code file to create a code table that has an index value storing the symbol's code

Create a traversal of the Huffman tree along with a stack of bits

Check through each symbol of the input file and encode the code to its output file

decode.c

Include header files

Setup get-opt cases

Create switch cases for h, infile, outfile, and v

Calculate and print decompression statistics to stderr

Read input file data

Create a stack of nodes to reconstruct the Huffman tree

Read input file bit-by-bit

Traverse through the tree one link at a time

Create a Loop that continues to check this process from the root based on where the leaf node is located

If file reads a 0,

Output results from left child

If file reads a 1,

Output results from right child

node.c

Include header files

Node_create

- Allocate memory for a node on the heap
- Set the left node pointer
- Set the right node pointer
- Set the symbol node pointer
- Set the frequency node pointer
- Return the node

Node_delete

- Call free on the node and set the pointer on n to null
- Set pointer to NULL after freeing memory for the node

node_join

- Create a parent node and set the symbol to $ and frequency = sum of left and right child's frequency
- Add left and right frequencies
- Set the parent node to left
- Set the parent node to right
- Return parent node

Node_print

- Print out the symbol and frequency of the node for debugging purposes

pq.c

Create a struct for the PriorityQueue

Fill the struct with head, tail, capacity and a double pointer Q for the Node

Pq_create

- Allocate memory on the heap for the priority queue using malloc
- If the priority queue exists
    - Set the head and tail = 0
    - Set the capacity pointer equal to capacity
    - Set the Node pointer equal to allocating memory for the Node pointer
    - If the priority queue points to Node Q
        - Return the priority queue
    - Free the priority queue
- Return the priority queue

Pq_delete

- If the pointer to the priority queue exists
    - Free the priority queue pointing to the Node
    - Free the priority queue pointer
    - Set the priority queue pointer to NULL

Pq_empty

- Set the head of the priority queue equal to 0

Pq_full

- Set the head of the priority queue equal to the capacity

Pq_size

- Set the head of the priority queue equal to the pointer

Enqueue

- Make a bool for full and set that equal to the pq_full function with the priority queue as a parameter
- If the queue is not full
    - If the priority queue is empty
        - Set the pointer of the priority queue equal to a variable
        - Set the pointer of the priority queue pointing at the header +1
        - Return true
    - Else
        - Set the pointer of the priority queue equal to a variable
        - Set a for loop that checks the range between 1 and the length of the array
        - Set the iterator equal to the variable
        - Initialize and move the value to a temp variable
        - Create another loop that checks the frequency
        - Subtract from the variable
        - Move stats and temp into an array
        - Set the pointer of the priority queue pointing at the header +1
        - Return true
- Return false

Dequeue

- If the priority queue exists
    - If the priority queue is empty
    - Subtract from the head of the queue
    - Return true
- Return false

Pq_print

- Create a debugging function to check the head, tail, and capacity of the priority queue

code.c

Code code_init

- Create a struct called Code on the stack
- Set the top = 0
- Set array of bits equal to null
- Return struct Code

Code_size

- Set a pointer to the top using a pointer

Code_empty

- Set the top of the code equal to 0

Code_full

- Set the top of the code equal to the number of bits in the code minus one

Code_set_bit

- Divide 8 from the number of bits in an index from the pointer c
- Check if that is not equal to the range
    - If it is out of the range
        - Return false
    - Else
        - Return true

Code_clr_bit

- Divide 8 from the number of bits in an index from the pointer c
- Check if that is not equal to the range
    - If it is out of the range
        - Return false
    - Else
        - Return true

Code_get_bit

- Get bit at index i in the Code
- Divide 8 from the number of bits in an index from the pointer c
- Check if that is not equal to the range
    - If it is out of the range
        - Return false
    - Else

        - Return true

Code_push_bit

- Push bit onto the Code by calling the set bit function
- Add to the top of the code
- If code is full
    - Return false
- Else
    - Return true

Code _pop_bit

- Pop bit off the Code bt calling the get bit function
- Set the bit equal to the top
- Subtract from the top
- If code is not full
    - Return false
- Else
    - Return true

Code_print

- Debugging function for the program, checks if the code is full or empty.

io.c

Set header files

Initialize static and nonstatic variables for byte, mask, index, max index, buffer, bytes read and bytes written

Read_bytes

- Initialize a bytes variable that will store the number of bytes read by the file
- Loop through the program for when bytes are greater than 0
    - Set that bytes variable equal to the buffer and the file that store the bytes
    - Increment bytes
- Return total bytes

Write_bytes

- Initialize a bytes variable that will store the number of bytes written by the file
- Loop through the program for when bytes are greater than 0
    - Set that bytes variable equal to the buffer and the file that store the bytes
    - Increment bytes
- Return total bytes

Read_bit

- Check if the index equals 0 or if the index is equal to the max index
    - If so, read the bytes into Max Index
    - If max index equals 0
        - Return false
    - If buffer of the index and mask is not equal to 0
        - Set the pointer of bit equal to an arbitrary number
    - Else
        - Set the bits equal to 0

    - Add to the mask
    - If the mask equals zero, then increment the index by 1
    - Return true

Write_code

- Create a static buffer of bytes using malloc/calloc and tracks which bit to return with a pointer bit
- Loop through the size of the code and increment the iterator
    - If the bit equals 1, mask the byte
- Sort through each bit in the code and start buffering them into the buffer from the 0th bit in c
    - Add to the index of the buffer
- If the index is equal to block, write bytes to the index and set the index to 0
- While the buffer is filled with bits, write output to outfile
- Set the index to 0

Flush_codes

- If the mask is not equal to an 8byte set, then increment the index of the buffer by 1
- Write the bytes to the outfile

stack.c

Set header files

Stack_create

- Create a struct for Stack and initialize variables for top, capacity, and a Node.

Stack_delete

- Frees the pointer stack and points it to items
- Sets the stack pointer to null
- Frees stack pointer
- Sets stack pointer equal to null

Stack_empty

- Set top of stack equal to 0

Stack_full

- Set the top of stack equal to capacity

Stack_size

- Sets the top of stack

Stack_push

- Push node onto stack
- Increment the top of stack
- If stack is full
    - Return false
- Else
    - Return true

Stack_pop

- If stack is empty

        - Return false
    - Pop node off the stack
        - Subtract from top
        - Set the pointer of items equal to the pointer of the node
        - Return true

Stack_print

- Debugging function that prints out the top and capacity of the stack

Huffman.c

Include header files

Build_tree

- Create a Huffman tree using the histogram
- Create a priority queue with alphabet as the histogram
- Loop through an index and check if the characters are greater than the index
- Increment the index
- Set the left node
- Set the right node
- Set the top node
- Set the root
- Loop through the size of the while loop while pq_size is greater 1
- Dequeue the pq and the left node
- Dequeue the right node from the pq
- Join the two child nodes into one parent node
- Enqueue the parent node from the pq
- Dequeue the root from the pq
- Return root

Build_codes

- Initalize the Code table
- Check if the root exists
    - If the left and right node exists
        - Set the table alphabet equal to the code
    - Else
        - Push the code bit
        - Build the code for the left root and table
        - Pop the bit for the code
        - Push the bit for the code
        - Build the code for the right root and table
        - Pop the bit for the code

Dump_tree

- Initialize the character L and I
- If the root Node exists
    - Dump the left node
    - Dump the right node
    - If the right and left node exists
        - Write the left one
        - Set the symbol for the Node

        - Write the symbol to the outfile
    - Else
        - Write the I character to the outfile

Rebuild_tree

- Set the left node
- Set the right node
- Set the top node
- Set the root
- Create a new stack with nbytes
- Create a for loop that loops through the nbytes and increments it whenever the nbytes are greater than the iterator
    - If the index of the tree is equal to the character of L, create a node and push it to the stack
        - Create a node with the tree
        - Push the node created to the stack
    - If the index of the tree is equal to the character of I, pop the left and right bits off the stack and join them in a parent node.
    - Push them onto the stack
    - Pop the remaining node root

Delete_tree

- If the root pointer exists
    - Delete the left node
    - Delete the right node
    - If the left node and the right node exist
        - Delete the root

Credit:

- I attended Eugene's section on 2/18/21, which helped give me general guidance on how to approach this lab, as well as sketch out how the program runs.
- I used the pseudocode from the asgn6 documentation.
- I also used the asgn6 documentation and reviewed the pseudocode provided by Elmer in discord.