Assignment 5: Huffman Coding

Kelly Liu

October 28, 2021

1 Description

In this assignment, we are making a program that encodes and decodes a message from a file by creating a min heap and post order traversal to travel along the tree also known as the Huffman's method.

2 Pseudocode

2.1 stack.c

First, I need to include all files and define certain variables.

2.1.1 Stack

```
This implements our structure declaration.
Set up the top,capacity and items variables
```

2.1.2 stack_create

```
Create our stack by allocating memory
Initialize top to 0
Initialize capacity to argument passed in
Initialize our items array by allocating memory
Use an if statement to check if our stack is not equal to our items array
If so, free up our stack and set it equal to null
Return stack
```

2.1.3 stack_delete

```
Delete our items stack and free up memory
```

2.1.4 stack_empty

Use an if statement to check if top equals 0
If so, return true
Else, return false

2.1.5 stack_full

Use an if statement to check if the top-1 is equal to the capacity
If so, return true
Else, return false

2.1.6 stack_size

Return top

2.1.7 stack_push

Run stack_full() to see if it's true
If it's true, return false
Else
Push argument x to the top of our stack
Update top to itself plus one
Return true

2.1.8 stack_pop

Run stack_empty() to see if it's true
If it's true, return false
Else
Update top to itself minus one
Pop an element of our stack into pointer x
Return true

2.1.9 stack_print

Debugging function

2.2 encode.c

Create a histogram that includes the inputs using a for loop to match the inputs with the corresponding times that they occurred
Create the Huffman tree using the histogram and functions in our pq.c and

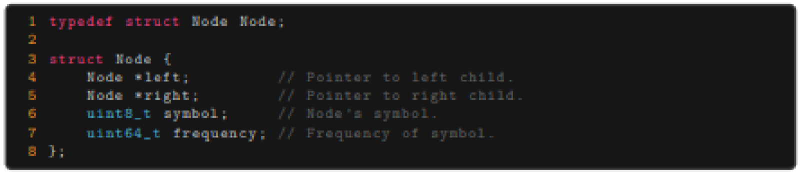
node.c files
Travel along the tree using traversal recursively to fill up the code table
Create the header for the file and write that header into our outfile by post order traversing the tree
Then we write the Huffman tree to our outfile using a recursion
Ended up using Eric's Pseudocode
Didn't exactly finish

2.3 decode.c

Read the header written in the outfile to recreate the Huffman tree
Create a stack of nodes and using the functions in node.c and stack.c to to create nodes, join nodes, and push nodes in order to finish our tree.
Read through the tree using bits to walk left or right to find the symbol needed to print to the outfile and recursively go through the tree until there are no symbols left.

2.4 node.c

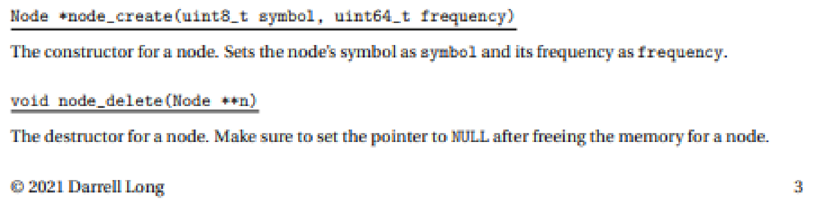
First, I need to include all files and define certain variables.



```c
typedef struct Node Node;

struct Node {
    Node *left;          // Pointer to left child.
    Node *right;         // Pointer to right child.
    uint8_t symbol;      // Node's symbol.
    uint64_t frequency; // Frequency of symbol.
};
```

This implements our structure declaration.
Set up the left,right,symbol and frequency variables



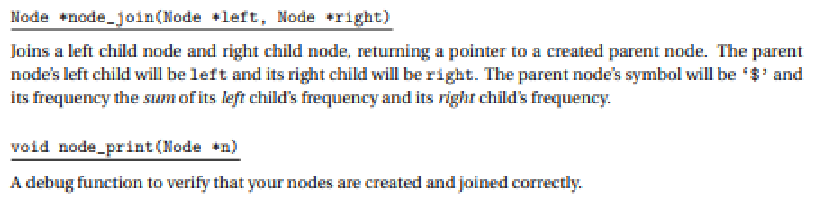
`Node *node_create(uint8_t symbol, uint64_t frequency)`

The constructor for a node. Sets the node's symbol as `symbol` and its frequency as `frequency`.

`void node_delete(Node **n)`

The destructor for a node. Make sure to set the pointer to `NULL` after freeing the memory for a node.

© 2021 Darrell Long 3



`Node *node_join(Node *left, Node *right)`

Joins a left child node and right child node, returning a pointer to a created parent node. The parent node's left child will be `left` and its right child will be `right`. The parent node's symbol will be `'$'` and its frequency the *sum* of its *left* child's frequency and its *right* child's frequency.

`void node_print(Node *n)`

A debug function to verify that your nodes are created and joined correctly.

Psuedocode for the following fucntions within the file is provided. I will figure out the exact implementations while I code.
Didn't exactly finish

2.5 huffman.c

2.5.1 build_tree

```
Create a Priority Queue
Use a for loop to loop from 0 to ALPHABET
Check if the value of hist[i] (i from the for loop is greater than 0
If so, create a node and enqueue it
Exit for loop
While the size of the q is bigger than 2
Create the left and right node and then dequeue the node in the stack to those temp variables
Then we join those two nodes to a parent node
Enqueue that parent node
End of while loop
Set another temporary node and dequeued our priorityqueue to that root
Make sure to delete our pq variable afters
Return our temporary node that holds our dequeued our priorityqueue
```

2.5.2 build_code

```
Check if the root is not null and if it is check if the left and right of the root is null
If so set the table[symbol of the root] to our c variable passed in
End of if statement
Check if root is null and if so return
Else
Push our pointer c and then we recursively build our code by inputting our left node of our root
Pop that and do the same thing with the right node
```

2.5.3 dump_tree

```
check if root is not null and if so recursively dump the left and right side of the root node
Then we use an if statement to check if the right and left is null, and if so we essentially right the character L
Else we do the same thing but with character I
```

2.5.4 rebuild_tree

```
Create a stack and a parent node
Loop through from 0 until nbytes
Check if the dump[i] of the for loop is == to L
And if so create a node of dump[i+1] and push it onto our stack
If the dump[i] is equal to I then we create a left and right node and pop two nodes on our stack to them and then join them to a parent node and then finally push it onto our stack
End of for loop
Pop our parent node and delete our stack
```

2.6 io.c

2.6.1 read_bytes

```
Set a count variable
While nbytes is bigger than 0
Set a temp variable to the read of buf + count
Set a if statement to check if temp is smaller or equal to 0 and if so break
Increment my variables by temp
End of while loop
Return count
```

2.6.2 write_bytes

Basically the same as read bytes except we are writing instead of reading and then decrementing nbytes instead of incrementing

2.6.3 read_bit

Create my buf array and top and pos
Check if the pos is equal to my top multiplied by 8 since it's bytes
And if so set top to the read byte
Set pos = 0
Use an if statement to check if top is less than 0 and if so return false
End of if statement
Do bit math to be able to read the bit correctly
return true

2.6.4 write_bit

Use a for loop to loop from 0 to top
Then do bit math to get the correct bit to write
Increment my c pos global variable
Check if the cpos divided by 8 is equal to block and if so flush
End of for loop

2.6.5 flush_code

Flush our codes by writing to byte depending on the position of my cpos global variable
Then use memset to set everything to 0
set cpos to 0

2.7 code.c

First, I need to include all files and define certain variables.

2.7.1 code_init

Create all our variables

2.7.2 code_size

Return my c top

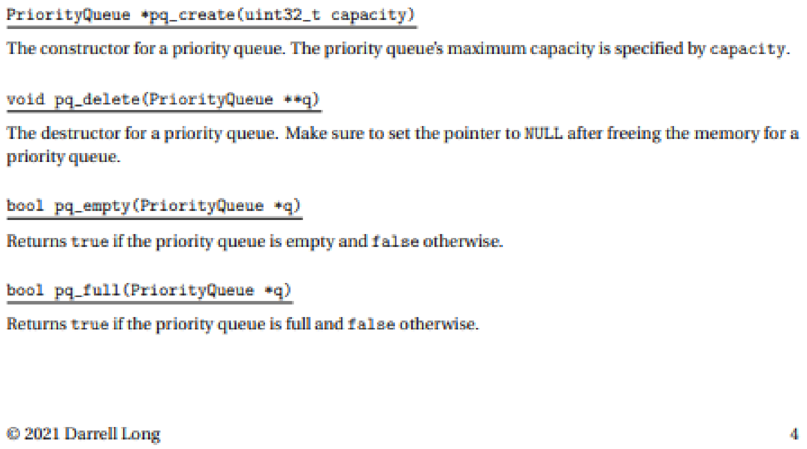
2.7.3 code_empty

Check if it's empty and if so return true else return false

2.7.4 code_full

Check if it's full and if so return true else return false

2.8 pq.c



`PriorityQueue *pq_create(uint32_t capacity)`

The constructor for a priority queue. The priority queue's maximum capacity is specified by `capacity`.

`void pq_delete(PriorityQueue **q)`

The destructor for a priority queue. Make sure to set the pointer to `NULL` after freeing the memory for a priority queue.

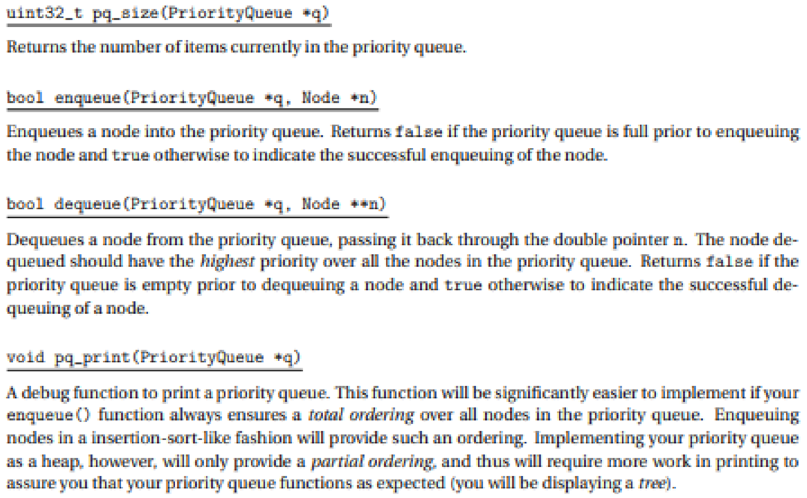
`bool pq_empty(PriorityQueue *q)`

Returns `true` if the priority queue is empty and `false` otherwise.

`bool pq_full(PriorityQueue *q)`

Returns `true` if the priority queue is full and `false` otherwise.

© 2021 Darrell Long 4



`uint32_t pq_size(PriorityQueue *q)`

Returns the number of items currently in the priority queue.

`bool enqueue(PriorityQueue *q, Node *n)`

Enqueues a node into the priority queue. Returns `false` if the priority queue is full prior to enqueuing the node and `true` otherwise to indicate the successful enqueuing of the node.

`bool dequeue(PriorityQueue *q, Node **n)`

Dequeues a node from the priority queue, passing it back through the double pointer `n`. The node dequeued should have the *highest* priority over all the nodes in the priority queue. Returns `false` if the priority queue is empty prior to dequeuing a node and `true` otherwise to indicate the successful dequeuing of a node.

`void pq_print(PriorityQueue *q)`

A debug function to print a priority queue. This function will be significantly easier to implement if your `enqueue()` function always ensures a *total ordering* over all nodes in the priority queue. Enqueuing nodes in a insertion-sort-like fashion will provide such an ordering. Implementing your priority queue as a heap, however, will only provide a *partial ordering*, and thus will require more work in printing to assure you that your priority queue functions as expected (you will be displaying a *tree*).

Psuedocode for the following fucntions within the file is provided. I will figure out the exact implementations while I code.

3 Files

encode.c- A source file for implementing of my huffman encoder.

decode.c- A source file for implementing of my huffman decode.

node.c- A source file for implementing the node ADT.

node.h- A header file that specifies the interface for node.c.

stack.c- A source file for implementing my stack ADT.

stack.h- A header file that specifies the interface for stack.c.

huffman.c- A source file for implementing my Huffman coding module interface

huffman.h- A header file that specifies the interface for huffman.c.

io.c- A source file for implementing my I/O module.

io.h- A header file that specifies the interface for io.c.

code.c- A source file for implementing my cod ADT.

code.h- A header file that specifies the interface for code.c.

pq.c- A source file for implementing my priority queue ADT.

pq.h- A header file that specifies the interface for pq.c.

defines.h- A header file that contains macros.

header.h- A header file that contains our definition for our file header.

Makefile- This allows us to use clang and compile our program.

README.md- In markdown format, it tells us how to run the program and how the program was made.

DESIGN.pdf- This is how I started thinking about how to code the program.

4 Credit

1. Professor Long has provided pseudocode in the Assignment 5 description PDF for some files as well as describing specific things that we should be doing.
2. Professor Long has provided us with a few files in the resources folder in Assignment 5.
3. Professor Long has provided us with pseudocode for stack.c in Assignment 4 description 4 PDF which I turned into workable code and used in this assignment as well.
4. Use Eric's Pseudocode provided in his notes for encode.c and my pq.c files.