Justin Marion

Prof Long

CSE 13S

May 13, 2021

Asgn6 DESIGN.pdf

NODE ADT:

The ADT here will be easy to implement directly from the design doc, besides create as this ADT introduces itself as a struct. Instead of recursively creating a left and right node, we will set the left and right pointer node to NULL.

When joining two nodes, its left and right node will be set accordingly

PRIORITY QUEUE ADT:

The implementation for the priority queue ADT will be very similar to my previous queue implementation in asgn3. I will simply have to add a priority character in the struct aswell as reconstruct the entire enqueue() function. This will use an insertion sort as described in the assignment doc to find where in the queue the respective node should be inserted. Shifting the nodes will be done by shifting them all by one and if and end reaches the top or end of a queue then we will simply move the node to the other end and adjust the tail/head as such.

```
enqueue() {
        For i in backwards queue:
                If the frequency of a node is greater than the one we are on:
                        Shift the node we are on to the right
                Else:
                        Insert the node
```

}

CODE ADT:

This code ADT will be fairly simple and reuse the same bitwise operations as my bit vector ADT from asgn5.

This will use very similar code from our bv.c and stack.c

I/O ADT:

This will be implemented first.

The read and write operations can be implemented directly from the asgn doc by looping indefinitely using read() until we have read/written the desired number of bytes. In order to implement read_bit I will be using the same bitwise operations as we have been using in the previous assignment.

read/write bytes:

    While we still have bytes to read/write:

        read/write to/from the buffer

        Update the buffer_idx

        If we had an error reading, return -1

        If we reached the end of the file break

    Return bytes read

Read_bit:

    If the buffer_idx is 0:

        Read more bytes into the buffer:

If we read less than asked we are nearing the end of the file and must remember the idx

Store the next bit in the bit pointer

Increment the idx

If the idx equals the last bit we return false

STACK ADT:

To implement the stack ADT I will be able to reuse the same stack I had used in previous assignments with the exception of storing nodes.

Exactly the same implementation as stack.c from previous assignments except we store nodes.

Huffman.c

Most these functions will use post order tree traversals:

Post order traversal:

If you are in a leaf node:

Update the table

Else:

Go to the right node and traverse

Go to the left node and traverse

Build tree:

Create a pq

Create a node for each element and enqueue it

While there are still slements in the oq:

Pop 2 elements, join them and enqueue the join

Return the last node

Rebuild tree:

    Create a stack

    Loop for nbytes:

        If the element is an L:

            Read the next element

        If the element is an I:

            Join the past two nodes

    Return the root node