Winston Yi
wyi10@ucsc.edu
5/1/2021

CSE13s Spring 2021
Assignment 6: Huffman Encoding
Design Document

OVERVIEW:

For this assignment, I have created a data compression program following the Huffman coding algorithm. My program will encode a series of bytes into a compressed version. I have also created a program that will decode the compressed version into their original sequence of bytes.

TOP LEVEL DESIGN:

The top level design is shown by the following pseudocode:

```
Overview of stack.c:
You've done a stack like 5 times.
```

```
overview of code.c:
a stack, but with bitvectors. constructed with an initializer
instead of a m/calloc call. Bytes are items in stack, use bit
operations to get each bit when popping/pushing.
```

```
overview of node.c:
node_join(): joins two nodes
  make the parent node
  SET THE LEFT/RIGHT PTRS OF THE PARENT NODE TO THE CHILDREN
```

```
overview of pq.c:
enqueue:
  for (head to tail, and head != tail)
    copy node1 to next in line (now node2 (node1 == node2 btw))
    if frequency of enqueued node > node1:
      replace node2
  if enqueued node has not been inserted:
    q[head] = node
  call an insertion sort on queue
dequeue:
  set ptr to head
  increment head
```

```
overview of io.c:
read_bytes:
  while there are bytes to read and no EOF:
    read bytes,
    decrement bytes to read by bytes read
    increment buffer ptrs by bytes read
write_bytes: same as above, except with write()
read_bit:
  index 0 means read in bytes
  if bytes read in < BLOCk, that means we reached EOF, set the
last bit to read in accordingly
  store bit in address ptr points to
  bit index += 1
  if BLOCK bytes read in, reset bit index
  if last bit has been hit, return
write_code:
  get bit
  if bit 1/0: set bit in buffer accordingly
  if bit_index is block:
    write out buffer
flush_codes: write out remainder of buffer with
write_bytes(bit_index)
```

```
overview of huffman.c:
build_tree:
  make a pq based on hist
  build tree (dequeue x2, join, enqueue parent)
  return root
build codes:
  create a code
  call build
build(root, table, code):
  perform post-order (PO), pushing/(1/0) when left/right and
  popping when coming back
  NOTE: SEPARATE STATEMENTS FOR EACH (LEFT NULL) OR (RIGHT NULL)
NOT (AND NULL)
rebuild_tree:
  if 'L'
    push symbol onto stack
  else (internal node):
    same as build tree
delete_tree:
  perform PO delete nodes
```

```
overview of encode.c:
  getopt command lines
  set header fields
  clear histogram
  fill histogram (adding two nodes to always create a tree)
  get #unique syms, set to header field tree size
  make the tree, print the tree_dump (PO), make the codes
  reset file ptr to beginning (at end since we parsed file when
making histogram)
  write codes
  flush codes
```

```
overview of decode.c:
  read in/check header, set file perms
  read tree_dump and rebuild tree
  while there are bits to read and characters written <
head.file_size:
    go through tree (based on codes written by encode) and when
leaf node, print node's symbol
```

DESIGN PROCESS:

I first attempted to learn Huffman encoding before Eugene's section. Big mistake. After attending section, I made my stack, queue, code, Huffman, and node ADTs and started encoding. I was then stuck when building a tree from Friday-Friday as my priority queue was incorrect. I was unable to get help after debugging while waiting for help up to 6+ hours a day, whereupon I said screw it and deleted everything, ignored what previous TA's/Tutors told me, and did everything this class taught me not to (coding like a monkey). It ended up working. After a long night, I was able to get my tree working. The other Huffman and io functions were not as difficult compared to debugging my tree but they were still time-consuming processes. I did run into a small roadblock where I was writing each code as a whole byte (which in hindsight, doesn't make sense as it would not compress the file at all or save space). Fortunately, I did not run into any other large week-long roadblocks. Later, I realized that I needed to use the extern variables in io.h to create compression statistics and allow files to be run in stdin/stdout which would require a rather large rehaul of my io and coding functions. Although this was definitely feasible, I, being fed up with this assignment (especially the incredibly frustrating tree fix) decided I had already invested unhealthy levels of time and work into this assignment and instead recuperated from the severe side effects of the 2nd COVID shot.

FINAL THOUGHTS:

Although this assignment was incredibly difficult. Eugene's sections and the assignment sheet, however, were very helpful. The two weeks were definitely required to do this assignment. However, if accurate help was available from the start and some parts were cut out (such as low-level syscalls and a provided priority queue) I believe this assignment would be doable in one week. Overall, what this assignment drove home was the fact that you are a much better programmer than you think and that you can only rely on yourself to resolve bugs and progress through the assignment. And Eugene. Eugene's sections and assignment sheet are godsends.