# DESIGN.pdf

Githika Annapureddy

March 2023

## 1 encode.c

in encode.c a buffer is made that is 4KB long. In encode.c, the infile (message to be compressed) is read. As it is being read, pairs are being generated for each symbol and corresponding code. The codes are being stored in the trie. Each pair is being written to buffer. Once the buffer is full, we output it to outfile (the compressed message) and empty the buffer. Then we start filling it again and keep doing this process until the entire message in infile has been read.
so the output for encode.c is a bunch of pairs of (code, symbol). All of the write functions (and flush pair, which are used to output the remaining pairs in buffer if buffer is not full) are used in encode.c. The read bytes function is also used to initially read from the infile, or do we read from the infile using a regular fread()?
if verbose is true, print the compressed file size, the uncompressed file size, and the ratio between them

## 2 decode.c

in decode.c, the infile (message that has been compressed) needs to be decompressed and written out to outfile. Infile contains a bunch of pairs of (code, symbol), it is decode.c's job to output words from that. So, a buffer of size 4KB is made. First, the FileHeader is read in, and magic is checked.
Then, the rest of infile is read into the buffer. The pairs are read from infile using read pair().
Once the buffer is full, it is processed. The corresponding words are written to outfile using write word. Then the buffer is emptied, and it is filled again from infile. This process is repeated until there are no more pairs in infile. Flush words is used to output words to outfile if the buffer is not full but everything in infile has been read.
if verbose is true, print the compressed file size, the uncompressed file size, and the ratio between them

## 3 io.c

read bytes is a function that returns an int, the number of bytes read. it accepts the parameters int infile, uint8t *buf, and int to read.
bytes read = 0
while (bytes read ¡ to read)
bytes read += read(infile, buf, to read)
if bytes read is 0, break the loop because this means EOF was reached
outside of the loop, return bytes read
read() is a function that is in the unistd.h library.

write bytes is a function that returns an int, the number of bytes written to outfile. it accepts the parameters int outfile, uint8t *buf, and int to write.
bytes wrote = 0
while (bytes wrote ¡ to write)
bytes wrote += write(infile, buf, to write)
if bytes wrote is 0, break the loop because this means EOF was reached
outside of the loop, return bytes wrote
write() is a function that is in the unistd.h library.

read header is a function that returns nothing and accepts an int infile and FileHeader *header as parameters
first, we check the endiannes using big endian()
then we use read bytes(infile, header, sizeof(FileHeader)) to read in the FileHeader. it is read into the header parameter
if it is big endian, then we have to swap the endianness
the buffer (a FileHeader) includes a uint64t called magic and a uint16t called protection
each has to be extracted and swapped using swap32() and swap16() from endian.h
then we have to verify that magic is the correct number. if it is not correct, we use an assert to output an error (stop decompression program all together since you cannot decompress it if it is the wrong magic number).

write header is a function that returns nothing and accepts an int outfile and FileHeader *header as parameters
first, we check the endiannes using big endian()
if it is big endian, then we have to swap the endianness
the buffer (a FileHeader) includes a uint64t called magic and a uint16t called protection
each has to be extracted and swapped using swap32() and swap16() from endian.h
then we use write bytes(infile, header, sizeof(FileHeader)) to write the FileHeader.

read sym is a function that returns a boolean. it accepts an int infile and a

uint8t *sym as parameters
if there are no more symbols to read, it returns false.
it will be called with the value of 1 symbol at each time.
a variable bytes read is made equivalent to read bytes (infile, sym, 1) since each sym is 8 bits or 1 byte
if bytes read == 1, return true because the read was successful
if the read was successful, the parameter *sym is where the symbol is read to
else, return false because the read was unsuccessful

write pair is a function that returns nothing. it accepts int outfile, uint16t code, uint8t sym, and int bitlen
first we write the code to outfile, updating total bits
then we check if the buffer is full, if it is then we write the buffer to outfile using write bytes()
we reset the buffer position to 0 if this is true.
I think we don't need to rewrite the buffer since we just set the buffer position to 0.
then we do the same thing with symbol.

flush pairs is a function that returns nothing. it takes an int outfile as a parameter.
if the buffer position is greater than 0 and less than or equal to the BLOCK * 8,
we write out the buffer to outfile using write bytes
then we reset the buffer position to 0

read pairs is a function that returns a boolean (true if there are pairs left to read in the buffer, false otherwise). it takes parameters int infile, uint16t *code, uint8t *sym, and int bitlen.
if 0 symbols have been read, then this is the first time calling read pairs so its helper funciton read block is called
read pairs first reads the code from infile using a buffer
then it checks if the buffer is full, and calls read block if it is
if the code read in is STOP CODE, the function returns false
then the symbol is read in from infile using a buffer
then it checks if the buffer is full, and calls read block if it is
return true (since the code that was read was not STOP CODE)

read block is a helper function for read pairs that reads in a block of data using read bytes(). It resets the buffer position to 0.

write word is a function that returns nothing. it takes int outfile and Word *w as parameters.
it writes the word to outfile using the same logic as writepair
total syms is updated since another symbol has been processed

flush words is a function that returns nothing. it accepts an int outfile as a parameter.
it flushes out the rest of the buffer to outfile using the same logic as flush pairs

## 4 trie.c

trie node create is a function that returns a TrieNode pointer.
create a Trienode pointer and set it to malloc for the size of one Trienode
using a for loop that goes from 1 to ALPHABET, make all the children of the pointer equal to NULL (use ptr-¿children[i] == NULL)
set the code of the pointer to index, the parameter that was passed in
return the pointer

trie node delete is a function that returns nothing
it takes a pointer to a Trienode
free the pointer
return nothing

trie create is a function that returns a TrieNode pointer.
create a Trienode pointer and set it to malloc for the size of one Trienode
using a for loop that goes from 1 to ALPHABET, make all the children of the pointer equal to NULL (use ptr-¿children[i] == NULL)
set the code of the pointer to EMPTY_CODE
return the pointer

trie reset is a function that returns nothing
using a for loop that goes from 1 to ALPHABET, make all the children of the pointer equal to NULL (use ptr-¿children[i] == NULL)
return nothing

trie delete is a function that returns nothing. it accepts Trienode pointer as a parameter.
using a for loop that goes from 1 to ALPHABET iterate through all the children
if ptr-¿children[i] != NULL
call trie delete to free the child
outside of the loop, free ptr

trie step is a function that returns a Trienode pointer. it accepts a Trienode pointer n and a symbol sym as parameters
set a variable position to the sym parameter - 48 to get the ASCII value of the sym
we can check if that particular sym is a child of the node *n by checking if the node of n at position is not NULL
if it is NULL, return NULL since the sym node does not exist
if does exist, return it

## 5 word.c

word create is a function that returns a Word pointer. it accepts a symbol pointer and len (the length of the symbol) as parameters.
create a word pointer using malloc (sizeof(Word))
if malloc fails, return an error
create a uint8t pointer called temp using malloc (sizeof(uint8t) * len)
if malloc fails, free the word and return an error
since the *syms member of the word ADT is an array (pointer), you need a pointer for it.
the memory that *syms points to is also apart of the word.
use memcpy to set the *syms parameter (which is an array of uint8ts) to the memory location pointed to by temp
set the word-¿syms = temp
set the word-¿len = length
return the word

word append sym is a function that returns a pointer to a word. it accepts a pointer to a word w and a symbol sym as parameters
create a word pointer called new wordusing malloc (sizeof(Word))
if malloc fails, return an error
set new word-¿ len = w-¿len + 1
create a uint8t pointer called temp using malloc (sizeof(uint8t) * new word -¿ len)
if malloc fails, free the word and return an error
if w-¿ syms is not empty, copy it to temp using memcpy
temp[new word -¿ len - 1] = sym;
this is to assign the last place of the array new word -¿ syms to sym (the parameter)
set the new word-¿syms = temp
free(w -¿ syms)
free(w)
return(new word)
we deleted the old word because we created a new word and assigned the desired parameters to the new word and returned it. When this funciton is called, the caller does not know that a different word instance was returned (it doesn't matter).

word delete is a function that returns nothing. it accepts a word pointer as a parameter.
free word -¿ syms
free the word pointer

wt create is a function that initializes and returns a pointer to a wordtable. it has no parameters.
create a word table pointer called wt using malloc (sizeof(Word) * MAX CODE)

create a word pointer called w using the word create function. the parameters should be ”” for the sym and 0 for the length
wt[EMPTY CODE] should be made equal to the word pointer
return wt

wt reset is a funciton that returns nothing. it accepts a pointer to a word table as a parameter. it resets the wordtable to only contain an empty word at index 0.
make all the words at every index of the wordtable NULL using a for loop that goes from 1 to MAX CODE
create a word pointer called w using the word create function. the parameters should be ”” for the sym and 0 for the length
wt[START CODE] should be made equal to the word pointer

wt delete is a function that returns nothing. it accepts a word table pointer as a parameter and deletes the word table.
use a for loop to loop through each word at every index of the word table
for each word, if it is not NULL, delete it using the word delete function
finally, outside the loop free the word table