Assignment 6 DESIGN.pdf

Version 2

Vincent Liu

3/9/23

Purpose:

The purpose of this assignment is for students to familiarize themselves with the concept of data compression, how it works, as well as how it can be implemented with the use of the C programming language. Students are expected to compress text and binary files, as well as decompress said files. Additionally, interoperability between little and big endian systems is expected. Read and Write blocks are expected to be in blocks of 4KB. Both encode (compress) and decode (decompress) are to use variable bit-length codes. For this assignment, students will be working with TrieNodes, as they offer a simple and easy to understand introduction to the concept of compression. In particular, word tables will be utilized, which in essence is a dictionary for letters. Each letter will be saved into a struct and associated with a symbol that represents said letter. The type of data compression and decompression students will be doing in this assignment is known as Lempel-Ziv Compression, in particular. LZ78-a lossless compression algorithm.

Note*

-pseudocode for encode.c and decode.c for this assignment was provided in the assignment pdf

---

Files in Directory:

1. encode.c:
    - This contains the main() function for the encode program.
2. decode.c:
    - This contains the main() function for the decode program
3. trie.c:
    - the source file for the Trie ADT
4. trie.h:
    - the header file for the Trie ADT.
5. word.c:
    - the source file for the Word ADT
6. word.h:
    - the header file for the Word ADT.
7. io.c:
    - the source file for the I/O module.
8. io.h:
    - the header file for the I/O module

9. endian.h:
    - the header file for the endianness module.
10. code.h:
    - the header file containing macros for reserved codes.
11. Makefile
    - Used to clean directory, generate associated executable files, and properly format .c files.
12. README.md
    - Text file in Markdown format that describes how to build and run the program.
13. DESIGN.pdf
    - covers the purpose of the program
    - layout/structure of the program
14. WRITEUP.pdf
    - Describes what the program does, gives insight on the results found, explains what was learned, and clarifies steps taken to complete the assignment.

Structure of Program//Pseudocode (a):

Main files:

encode.c

- Helper function to get bitlen. Takes in uint16 “code”
    - counter = 0;
    - while code is not 0
        - code = code >> 1;
        - counter+=1
    - return counter
- Helper function that prints synopsis
- Use getopt to accept commands from terminal
    - -v : Print compression statistics to stderr.
        - Set verbose to true
    - -i : Specify input to compress (stdin by default)
        - Set using_stdin to false
        - infile=arg
    - -o : Specify output of compressed input (stdout by default)
        - Set using_stdout to false
        - outfile-arg
    - -h : displays program synopsis and usage.
    - If an argument is given but no following argument, print error synopsis

- If using_stdin is true
    - Set infile to 0
- If using stdin is false
    - Open infile with open(filename, O_RDONLY)
    - If error in opening infile
        - print error message
        - Return exit_failure

- Use fstat() to determine file size and protection bit mask

- Set magic number in header.magic
- Set header protection to be st_mode;

- If using_stdout is true
    - Set outfile to 1
- If using_stdout is false
    - Open outfile with open()
    - If error in opening outfile
        - print error message
- If fchmod failed to set header protection
    - Print error message
- Write header to outfile
- root = TRIE_CREATE()
- curr_node = root
- prev_node = NULL
- curr_sym = 0
- prev_sym = 0
- next_code = START_CODE
- while READ_SYM(infile, &curr_sym) is TRUE
    - next_node = TRIE_STEP(curr_node, curr_sym)
    - if next_node is not NULL
        - prev_node = curr_node
        - curr_node = next_node
    - Else
        - WRITE_PAIR(outfile, curr_node.code, curr_sym, BIT-LENGTH(next_code))
        - curr_node.children[curr_sym] = TRIE_NODE_CREATE(next_code)
        - curr_node = root
        - next_code = next_code + 1

  - if next_code is MAX_CODE
    - TRIE_RESET(root)
    - curr_node = root
    - next_code = START_CODE
  - prev_sym = curr_sym
- if curr_node is not root
  - WRITE_PAIR(outfile, prev_node.code, prev_sym, BIT-LENGTH(next_code))
  - next_code = (next_code +1) % MAX_CODE
- WRITE_PAIR(outfile, STOP_CODE, 0, BIT-LENGTH(next_code))
- FLUSH_PAIRS(outfile)
- Close infile and outfile
- Trie_delete root
- If verbose is true
  - if total_bits % 8 is not 0
    - my_total_bytes = (total_bits / 8) + 1
  - Else
    - my_total_bytes = (total_bits / 8)
  - my_total_bytes += 8
  - Compression ratio = 100 * (1 - (my_total_bytes / ((double) total_syms)))
  - Print compressed file size, uncompressed file size and compression ratio

decode.c

- Helper function to get bitlen. Takes in uint16 “code”
  - counter = 0;
  - while code is not 0
    - code = code >> 1;
    - counter+=1
  - return counter
- Helper function that prints synopsis
- Use getopt to accept commands from terminal
  - -v : Print decompression statistics to stderr.
    - Set verbose to true
  - -i : Specify input to decompress (stdin by default)
    - Set using_stdin to false
    - Input file=arg
  - -o : Specify output of decompressed input (stdout by default)
    - Set using_stdout to false
    - Output file-arg
  - -h : displays program synopsis and usage.
  - If an argument is given but no following argument, print error synopsis

- If using stdin is true
    - Set Infile = 0
- If using_stdin is false
    - Open infile with open()
    - If there is an error in opening infile
        - print a helpful error message
        - Return exit failure
- Use read_header() and verify the magic number
- If using stdout is true
    - Set stdout to 1
- If using_stdout is false
    - Open output file with open()
    - If error in protection bits
        - Print error message
    - If there is an error in opening outfile
        - print a helpful error message
- table = WT_CREATE()
- curr_sym = 0
- curr_code = 0
- next_code = START_CODE
- while READ_PAIR(infile, &curr_code, &curr_sym, BIT-LENGTH(next_code)) is TRUE
    - table[next_code] = WORD_APPEND_SYM(table[curr_code], curr_sym)
    - WRITE_WORD(outfile, table[next_code])
    - next_code = next_code + 1
    - if next_code is MAX_CODE
        - WT_RESET(table)
        - next_code = START_CODE
- FLUSH_WORDS(outfile)
- Close infile
- Close outfile
- wt_delete(table)
- If verbose is true
    - if total_bits % 8 is not 0
        - my_total_bytes = (total_bits / 8) + 1
    - Else
        - my_total_bytes = (total_bits / 8)
    - my_total_bytes += 8
    - Compression ratio = 100 * (1 - (my_total_bytes / ((double) total_syms)))
    - Print compressed file size, uncompressed file size and compression ratio

---

Structure of Program//Pseudocode (b):

Files with associated functions:

---

trie.c

- **TrieNode *trie_node_create(uint16_t code)**
    - Mynode = Malloc size of TrieNode
    - Trienode variable -> code
    - Loop to set each of the children node pointers to NULL
    - Return Mynode

- **void trie_node_delete(TrieNode *n)**
    - Pass in a single pointer and call free()

- **TrieNode *trie_create(void)**
    - Call trie_node_create(EMPTY_CODE)
    - Return Mynode

- **void trie_reset(TrieNode *root)**
    - For every item in ALPHABET
        - If child are not set to NULL
            - Delete child
            - Set child to null

- **void trie_delete(TrieNode *n)**
    - For every item in ALPHABET
        - If child are not set to NULL
            - Delete child
            - Set child to null
    - trie_node_delete(n)

- **TrieNode *trie_step(TrieNode *n, uint8_t sym)**
    - Returns n->children[sym]

---

word.c

- **Word *word_create(uint8_t *syms, uint32_t len)**
    - Myword = Malloc Word sizeof word
    - If myword is null

        - Free myword
        - Return NULL
    - myWord -> len = len
    - myWord -> syms = Malloc len*sizeof(uint8_t)
    - If myword->syms is null
        - Free myword
        - Return NULL
    - For i in len
        - myWord->syms[i] = syms[i]
    - Return myWord

- **Word *word_append_sym(Word *w, uint8_t sym)**
    - Myword = Malloc Word sizeof word
    - If myword is null
        - Free myword
        - Return NULL
    - myWord ->syms = malloc (w->len+1) * sizeof uint8_t
    - myWord->len = w->len + 1
    - For i in w->len
        - Word -> syms[i] = w->sym[i]
    - myWord->syms[w->len] = sym
    - Return myWord

- **void word_delete(Word *w)**
    - Free w-> syms
    - Free w

- **WordTable *wt_create(void)**
    - thing = Malloc MAX_CODE * sizeof word*
    - For i in maxcode
        - thing [i] is set to NULL
    - thing[EMPTY_CODE] = word_create(NULL, 0)
    - Return thing

- **void wt_reset(WordTable *wt)**
    - For i+1 in MAX_CODE
        - If wt[i] is not null
            - word_delete(wt[i])
            - Set wt[i] to NULL
    - Wt[empty_code] = word_create(NULL,0)

- **void wt_reset(WordTable *wt)**
    - For i in max_code
        - If wt[i] is not null
            - word_delete(wt[i])
            - Set wt[i] to NULL
    - free(wt)

---

io.c

- **help_refill(int) (used to help reset the location of indexes when refilling input buffer)**
    - set last index to end of next block in buffer
    - set current index to 0

- **int read_bytes(int infile, uint8_t *buf, int to_read)**
    - While there are more bytes to_read ((reading = read(infile, buf, to_read)) is not 0)
        - If End of file
            - exit_failure
        - Add to variable 'bytes_read'
        - buf += reading
        - Subtract # of read bytes from *to_read*
    - Return number of bytes read

- **int write_bytes(int outfile, uint8_t *buf, int to_write)**
    - While (wrote = write(outfile, buf, to_write)) does not equal 0)
        - If end of file
            - Print error message
            - Exit_failure
        - Bytes_written += wrote
        - Buf += wrote
        - Subtract #of written bytes from to_write
    - Return number of bytes written

- **void read_header(int infile, FileHeader *header)**
    - Call read_bytes(infile, cast header as uint8_t, sizeof header)
    - If big endian == true
        - Swap magic header swap32(header->magic)
        - Swap protection header swap16(header->protection)
    - If magic number is not correct
        - Exit failure

- **void write_header(int outfile, FileHeader *header)**
    - If big endian == true
        - Swap magic header swap32(header->magic)
        - Swap protection header swap16(header->protection)
    - Write bytes to outfile with (uint8_t *) header, sizeof(header)

- **bool read_sym(int infile, uint8_t *sym):** An Index to keep track of currently read symbol in buffer
    - If current index == last index
        - help_refill(infile)
        - Return false
    - *symbol = input_buffer[current index]
    - Add one to current index
    - Add one to total symbols
    - Return true

- **bool read_bit(int infile) (helper function to read a single bit)**
    - if bit_index is 8
        - Set bit_index to 0
        - Add 1 to curr_index
    - if curr_index == last_index
        - help_refill(infile)
    - x = 1 << bit_index
    - uint8_t y = x AND input_buffer[curr_index]
    - Add 1 to total_bits
    - Add 1 to bit_index
    - return y != 0;

- **void write_bit(int outfile, bool bit) (helper function to write a single bit)**
    - if bit_index is 8
        - Set output_bit_index to 0
        - output_index+=1
    - if output_index == BLOCK
        - flush_pairs(outfile)
    - x = output_buffer[output_index];
    - if bit
        - x = x OR 1 << output_bit_index;
    - if bit is false

        - x = x & ((0xFF << (output_bit_index + 1)) or (0xFF >> (8 - output_bit_index)))
    - output_buffer[output_index] = x
    - Add 1 to output_bit_index
    - Add 1 to total_bits

- **void write_pair(int outfile, uint16_t code, uint8_t sym, int bitlen)**
    - For i in bitlen
        - bit=code AND (1<<i)
        - write_bit(outfile, bit does not equal 0)
    - For i in 8
        - bit=sym AND (1<<i)
        - write_bit(outfile, bit does not equal 0)

- **void flush_pairs(int outfile)**
    - if output_bit_index is 0
        - write_bytes
    - Else
        - Output_buffer[output_index] = output_buffer[output_index] & (0xFF >> (8 - output_bit_index));
        - write_bytes
    - output_index = 0;
    - output_bit_index = 0;

- **bool read_pair(int infile, uint16_t *code, uint8_t *sym, int bitlen):** Reads a pair (code and symbol) from input file
    - Place read symbol into pointer to sym (*sym=val)
    - Keep track of current bit in buffer
    - When all bits are processed
        - Read another block
    - (The first bitlen bits are the code, starting from LSB)
    - (The last 8 bits of the pair are the symbol, starting from LSB)
    - If read code is not STOP_CODE (there are pairs still left to read in buffer)
        - Return true
    - Else
        - return false

- **void write_word(int outfile, Word *w):** Writes a pair to the output file.
    - For i in bitlen
        - Bit = read_bit(infile)

```
            *code = *code OR (bit<<i)
    For i in 8
            Bit = read_bit(infile)
            *sym = *sym OR (bit<<i)
    Return *code
```

- **void flush_words(int outfile):** Writes out remaining symbols in buffer to the outfile

```
    For i in w->len
        If output index is block
            flush_words(outfile)
        Output_buffer[output_index] is set to w->syms[i]
        Add 1 to output index
    Total symbols += w->len
```

Makefile

- Set compile for C language
- Cflags -Wall -Wpedantic -Werror -Wextra -standard c17
- Object that contains io.o, trie.o, word.o
- Generate ‘encode’, ‘decode’ executables if “all” or “make” command is given
- Generate all .o files from .c files and associated header files
- Exclusively create ‘decode’ or ‘encode’ given the command ‘make encode’ or ‘make decode’
- Clean:
    - Removes all executable files as well as associated ‘.o’ files
- Format:
    - Formats all c files to c17 standard

Citation/references

- Assignment 6 pdf . gave out function variable names, pseudocode for encode and decode
- TA Michael for helping me understand the assignment