Assignment 6 Lempzel Ziv Cryptography

Katrina VanArsdale

March 12, 2023

1 Description of Program

This program provides an encode program and a decode program. The encode program performs LZ78 compression on a given file while the decode program performs LZ78 decompression.

2 Files In Directory

- decode.c
  This contains the implementation and main() for the decode program
- encode.c
  This contains the implementation and main() for the encode program
- trie.c
  This is the source file for the Trie ADT
- trie.h
  This is the header file for the Trie ADT
- word.c
  This is the source file for the Word ADT
- word.h
  This is the header file for the Word ADT
- io.c
  This is the source file for the I/O module
- io.h
  This is the header file for the I/O module
- endian.h
  This is the header file for the endianness module
- code.h
  This header file contains macros for reserved codes
- Makefile
  This file compiles all of the files and creates .o files for every .c file. It also cleans up all those files afterward and can clang format them.
- README.md
  This markdown file will describe how to use my program and Makefile. It also lists and explains the command line options that my program accepts.
- WRITEUP.pdf
  This file will include things I learned about compression and lessons I learned while working on the assignment.
- DESIGN.pdf
  This file describes the design for this program with pseudo code. This is the file you are reading.

3 Pseudo Code

3.1 Tries

- TrieNode
  - TrieNode is a struct
  - TrieNode * children[ALPHABET]
  - uint16_t code
- TrieNode *trie_node_create(uint16_t code)
  - Allocate memory for TrieNode
  - Check that it was allocated
  - Set the trienode's code to passed in code.
  - loop through children[] and set it equal to NULL
  - return trienode
- void trie_node_delete(TrieNode *n)
  - Deletes a passed in node from TrieNode
  - sets n to NULL
- TrieNode *trie_create(void)
  - Creates a Trie with a root empty code
  - Returns trie_node_create(EMPTY_CODE);
- void trie_reset(TrieNode *root)
  - Resets the TrieNode to just the root
  - if root is null return
  - loop through the children of the root
  - – call trie_reset for each child
  - – set child to null
  - This is called if TrieNode reaches MAX_CODE
- void trie_delete(TrieNode *n)
  - Deletes a sub-trie from TrieNode from root n
  - loop through the children of n
  - – call trie_delete for each child
  - – set child to null
  - Free n with trie_node_delete
- TrieNode *trie_step(TrieNode *n, uint8_t sym)
  - Returns NULL if n is NULL
  - Returns n.children[sym]

3.2 Word Tables

- Word
  - Word is a struct
  - uint8_t *syms
  - uint32_t len
- WordTable
  - Define an array of Words
  - typedef Word * WorldTable
- Word *word_create(uint8_t *syms, uint32_t len)
  - Constructs a word where syms is an array of symbols that Word represents
  - Length of the array syms is given by len
  - Allocate memory for Word
  - If unsuccessful return NULL
  - Allocate memory for the word's symbols
  - If len is 0 set syms to NULL
  - If sysms is null and len is no 0 free syms and the word and return NULL
  - Copy each sym from the passed in array of syms into the word syms
  - set w.len to len
  - Returns Word * if successful otherwise returns NULL
- Word *word_append_sym(Word *w, uint8_t sym)
  - Constructs a new word from w appended with sym
  - If w is empty the new Word should only contain the symbol
  - Define a variable len
  - if the passed in word is not null set len to w.len and plus one
  - else len = 1
  - Define an array of syms with length len
  - If the passed in word is not null then loop through its syms and copy them into the array of syms
  - set the last index of syms to the passed in sym
  - return word_create(syms, len)
- void word_delete(Word *w)
  - If the words syms aren't null free them
  - free the word
- WordTable *wt_create(void)
  - Creates a new WordTable with a size of MAX_CODE
  - allocate memory for the wordtable
  - Call word_create(NULL, 0) at wt[EMPTY_CODE]
  - return the wordtable
- void wt_reset(WordTable *wt)
  - Resets a WordTable, wt to contain only an empty word, all other words are set to NULL

  - Loop from 2 to MAX_CODE
  - if wt[i] not NULL called word_delete(wt[i]) and set it to NULL
- void wt_delete(WordTable *wt)
  - Call word_delete(wt[EMPTY_CODE]) to delete the first word
  - call wt_reset(wt) to delete the rest of the words
  - free wt

3.3 I/O

- FileHeader
  - FileHeader is a struct
  - uint32_t magic
  - uint16_t protection
- int read_bytes(int infile, uint8_t *buf, int to_read)
  - Reads the number of bytes till to_read or no more bytes to read
  - While the number of bytes read is less than to_read
  - – Set variable readB to read(infile, buf + i, to_read - i)
  - – i += readB
  - if readB == 0 end while loop
  - Returns number of bytes read
- int write_bytes(int outfile, uint8_t *buf, int to_write)
  - Writes the number of bytes till to_write or no more bytes to read
  - While the number of bytes writen is less than to_read
  - – Set variable writeB to write(outfile, buf + i, to_write - i)
  - – i += writeB
  - if writeB == 0 end while loop
  - Returns number of bytes written
- void read_header(int infile, FileHeader *header)
  - Reads the FileHeader from input file
  - if it isn't little endian swap order of magic and protection
  - If the header magic is not 0xBAADBAAC print error
- void write_header(int outfile, FileHeader *header)
  - If the header isn't little endian swap order of magic and protection
  - Writes size of FileHeader into output file
- bool read_sym(int infile, uint8_t *sym)
  - If the current position in the buffer is greater than or equal to the length of the buffer read another block
  - reset current position
  - If no bytes were read return false
  - Set sym to the sym in the current position of the buffer
  - increment position

  - if the current position in the buffer is greater than or equal to the length of the buffer set buffer length and current position to 0
  - Returns true

- void write_pair(int outfile, uint16_t code, uint8_t sym, int bitlen)
  - Loop from 0 to bitlen
  - – Bits of the code are buffered first starting from LSB
  - – increment pair position
  - – if pair position is greater than or equal to BLOCK * 8 then call write_bytes(outfile, buf_pair, BLOCK)
  - – reset pair position
  - – reset the buffer to all zeros
  - loop from 0 to 8
  - – Bits of the symbol starting from LSB
  - – increment pair position
  - – if pair position is greater than or equal to BLOCK * 8 then call write_bytes(outfile, buf_pair, BLOCK)
  - – reset pair position
  - – reset the buffer to all zeros

- void flush_pairs(int outfile)
  - The amount to write is the current pair position in the buffer divided by 8
  - If the current position is not divisible by 8 then add one to the amount to write
  - Writes out remaining pairs to output file

- bool read_pair(int infile, uint16_t *code, uint8_t *sym, int bitlen)
  - If the current position in the buffer is greater than or equal to the length of the buffer read another block and reset current position
  - Loop from 0 to bitlen
  - – Bits from the buffer started at LSB are put into the code pointer
  - – Increment current position
  - – If the current position in the buffer is greater than or equal to the length of the buffer read another block and reset the current position
  - Loop from 0 to 8
  - – Bits from the buffer started at LSB are put into the sym pointer
  - – Increment current position
  - – If the current position in the buffer is greater than or equal to the length of the buffer read another block and reset current position
  - If code equals STOP_CODE return false otherwise return false

- void write_word(int outfile, Word *w)
  - Each symbol of the Word is placed into buffer
  - increment position in the buffer
  - Buffer is written once filled

- void flush_words(int outfile)
  - Write the remaining words into outfile

3.4 Programs

- Program options
  - Getopt() will take in 'vi:o:'
  - -v prints statistics to stderr
    - This includes compressed file size
    - uncompressed file size
    - space saving calculated by 100 x (1 - compressed size / uncompressed size)
  - -i ¡input¿ specifies input to compress for encode or decompress for decode (stdin by default)
  - -o ¡output¿ specifies output (stdout by default)
- Encode
  - Open() infile - print an error if unsuccessful and exit code
  - The magic number in the header must be 0xBAADBAAC
  - use fstat() to find the file size and protection bit mask
  - open outfile - print an error if unsuccessful and exit code
  - Write_header() to print file header into outfile
  - Check that the permissions for outfile match the protection bits from file header
  - Create a trie with a root EMPTY_CODE
  - Make a root node copy called curr_node
  - uint16_t next_code starts at START_CODE
  - Previous node will be called prev_node and previous symbol will be called prev_sym
  - While read_sym() = curr_sym until it returns false
    - Set next_node = trie_step(curr_node,curr_sym)
    - If next_node not NULL
      - prev_node = curr_node
      - curr_node = next_node
    - else
      - Write (curr_node -¿ code, curr_sym) where the bit length is the bit length of next_code
      - curr_node -¿ children[curr_sym]
      - curr_node = root of trie
      - next_code += 1
    - if next_code = MAX_CODE
      - trie_reset()
      - curr_node = root
      - next_node = START_CODE
    - prev_sym = curr_sym
  - If curr_node is not pointing to root of TrieNode
    - Write (prev_node -¿ code, prev_sym) the bit length should be the bit length of next_code
    - next_code += 1 (within the limit of MAX_CODE)
  - Write (STOP_CODE, 0) the bit length should be the bit length of next_code
  - Call flush_pairs()
  - close files
- Decode
  - Open() infile - print an error if unsuccessful and exit code

- The magic number in the header must be 0xBAADBAAC
- use fstat() to find the file size and protection bit mask
- open outfile - print an error if unsuccessful and exit code
- Write_header() to print file header into outfile
- Check that the permissions for outfile match the protection bits from file header
- Create a word table with wt_create() with an empty word at index EMPTY_CODE
- uint16_t next_code starts at START_CODE
- uint16_t curr_code
- While read_pair() = curr_code until curr_code = STOP_CODE
  - Set table[next_node] = word_append_sym(table[curr_code], curr_sym
  - write_word(table[next_code])
  - next_code += 1
  - if next_code is MAX_CODE
    - wt_reset
    - next_node = START_CODE
- Call flush_pairs()
- close files

4 Credits

- Pseudo code is referenced from the asgn6.pdf
- I looked at https://www.geeksforgeeks.org/trie-insert-and-search/ to better understand tries and how to make them
- I looked at https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/ to understand how to open the files correctly
- I also looked at https://www.digitalocean.com/community/tutorials/trie-data-structure-in-c-plus-plus to figure out how to create tries
- I looked at https://stackoverflow.com/questions/6647783/check-value-of-least-significant-bit-lsb-and-most-significant-bit-msb-in-c-c to figure out how to find the LSB
- I looked at https://en.cppreference.com/w/c/program/EXIT_status to use exit failure
- Looking at people's questions and answers on the discord was very helpful. I also asked my own questions and got helpful answers.