Assignment 6

DESIGN

Prof Veenstra

Nguyen Vu

1. trie.c

This section will take about the design of the file trie.c

1.1. TrieNode *trie_node_create(uint16_t code)

This function is a constructor for TrieNode. It creates a TrieNode with the code set to ‘code’ and all children node pointers to NULL.

Pseudo Code:

Allocate memory for the TrieNode

Set the code to ‘code’

Set all children node to NULL

1.2. void trie_node_delete(TrieNode *n)

This function delete the TrieNode

Pseudo Code:

Free the memory of TrieNode n

1.3 TrieNode *trie_create(void)

Initializes a TrieNode with the code EMPTY_CODE. Return the root if successful, return NULL otherwise.

Pseudo Code:

Create TrieNode with root is EMPTY_CODE

Return root if succeeded, NULL otherwise

1.4 void trie_reset(TrieNode *root)

This function resets a trie to just the root TrieNode by deleting its children and make them NULL.

Pseudo Code:

Make a for loop for the range

Call to trie_delete (will talk about it later)

Set the children to NULL

1.5 void trie_delete(TrieNode *n)

This function deletes a sub-trie starting from the trie rooted at node n and will recursive calls on each of n's children.

Pseudo Code:

Check if TrieNode n is NULL or not

Loop through all children and recursively call the trie_delete

Call to trie_node_delete to delete the last generation

1.6 TrieNode *trie_step(TrieNode *n, uint8_t sym)

This function returns a pointer to the child node representing the symbol 'sym'. If the symbol doesn't exist, NULL is returned.

Pseudo Code:

Check if the children exists and return the pointer

If not, return NULL

2. word.c

This section will take about the design of the file word.c

2.1 Word *word_create(uint8_t *syms, uint32_t len)

This function construct the Word ADT where ‘syms’ is the array of word and ‘len’ is the length of the array

Pseudo Code:

Allocate memory for Word

Check if Word is NULL or not, if yes, return NULL

Set len to ‘len’

Allocate memory for Word->syms and set it to respective data point in ‘syms’

2.2 Word *word_append_sym(Word *w, uint8_t sym)

This function constructs a new Word from the specified Word, ‘w’, appended with a symbol, ‘sym’.

Pseudo Code:

Make new ‘len’ and reallocate memory for ‘syms’

Check if ‘sym’ is NULL, if yes, return NULL

Append ‘sym’ to the Word->syms

2.3 void word_delete(Word *w)

This function delete the Word ‘w’

Pseudo Code:

Free ‘syms’ and set it to NULL

Free w

2.4 WordTable *wt_create(void)

This function creates a new WordTable with size of MAX_CODE and initialized with a single Word at index EMPTY_CODE

Pseudo Code:

Create and allocate memory for WordTable

Set Word at index EMPTY_CODE

2.5 void wt_reset(WordTable *wt)

This function reset the WordTable wt to contain just the empty Word

Pseudo Code:

Make a for loop to go through all element in the WordTable

Check if the element is NULL, if yes, call word_delete and make it NULL

3. io.c

This section will talk about the design of the file io.c

3.1 int read_bytes(int infile, uint8_t *buf, int to_read)

This function reads the bytes from infile with the amount of to_read using read() built-in function.

Pseudo Code:

Create temporary variables to count read bytes

Make a loop to check if the counter is equal to to_read

Use read() function to read bytes from infile

Return the counter

3.2 int write_bytes(int outfile, uint8_t *buf, int to_write)

This function reads the bytes from infile with the amount of to_write using write() built-in function.

Pseudo Code:

Create temporary variables to count written bytes

Make a loop to check if the counter is equal to to_write

Use write() function to write bytes to outfile

Return the counter

3.3 void read_header(int infile, FileHeader *header)

This function reads the header information of an input file, swaps endian if needed, check the magic number, and stores the information in a FileHeader struct.

Pseudo Code:

Use function read_bytes to read the header data from infile

Swap endian if needed

Check if the magic number is correct, if not, print an error

3.4 void write_header(int outfile, FileHeader *header)

This function writes the header information of a compressed file to an output file, swaps endian if needed, and stores the information.

Pseudo Code:

Swap endian if needed

Use function write_bytes to write the data to outfile

3.5 bool read_sym(int infile, uint8_t *sym)

This function buffers symbols from infile, read out those symbols to ‘sym’ and return true if there are symbols to be read.

Pseudo Code:

Create temporary variable to keep track of read symbols and index

Call function read_bytes and read in block

Update index and ‘sym’

3.6 void write_pair(int outfile, uint16_t code, uint8_t sym, int bitlen)

This function writes out code and ‘sym’ pairs to an output file, it swaps endian if needed, gets the bits of the code and symbol and writes them to the bit buffer. When the bit buffer is full, it writes it to the output file.

Pseudo Code:

Swap endian if needed

Make the ‘code’ and ‘sym’ into a pair of bits and put them to the bit buffer

If the buffer is full, write it to outfile

3.7 void flush_pairs(int outfile)

This function flushes any remaining pairs from the bit buffer to the output file.

Pseudo Code:

Check if the buffer is bigger than 0, if yes, write to outfile

3.8 bool read_pair(int infile, uint16_t *code, uint8_t *sym, int bitlen)

This function buffers ‘code’ and ‘sym’ pairs from an input file and outputs the ‘code’ and ‘sym’ to their specified parameters, respectively.

Pseudo Code:

Calculate the size of the pair in bytes and loop through each bytes

Read the code and symbol bits from the read buffer

Read one byte at a time and add it to the pair

If the read buffer is empty, read a new block of data

Extract the code and symbol from the pair to respective data in the struct

3.9 void write_word(int outfile, Word *w)

This function buffers the symbols in the Word ‘w’ to a write buffer, and writes the buffer to the output file when it is full.

Pseudo Code:

Loop through the len of w

Add the symbol to the write buffer

If the write buffer is full, write it to the output file

Reset the write buffer index to zero

3.10 void flush_words(int outfile)

This function flushes any remaining words from the write buffer to the output file.

Pseudo Code:

Write them to the output file

Reset the write buffer index to zero

4. encode.c

This main file compresses infile and the compressed output goes to outfile

Pseudo Code:

Create a trie root node.

Set the current node to the root node.

Set the previous node to NULL.

Set the current symbol to 0.

Set the previous symbol to 0.

Set the next code to the start code.

While the function to read the next symbol from the input file returns true:

Attempt to step to the next node in the trie based on the current symbol. Store the result in the "next_node" variable.

If the "next_node" variable is not NULL:

Set the "prev_node" variable to the current node.

Set the "curr_node" variable to the "next_node" variable.

Otherwise:

Write a pair to the output file consisting of the current node's code, the current symbol, and the bit length of the "next_code" variable.

Create a new node in the current node's children with the key of the current symbol and the value of the "next_code" variable.

Set the current node back to the root node.

Increment the "next_code" variable.

If the "next_code" variable equals the maximum code:

Reset the trie by creating a new root node and setting the current node to the root node.

Set the "curr_node" variable to the root node.

Set the "next_code" variable to the start code.

Set the "prev_sym" variable to the current symbol.

If the current node is not the root node:

Write a pair to the output file consisting of the previous node's code, the previous symbol, and the bit length of the "next_code" variable.

Increment the "next_code" variable and wrap around if it reaches the maximum code.

Write a pair to the output file consisting of the stop code, a symbol of 0, and the bit length of the "next_code" variable.

Flush any remaining pairs to the output file.

5. decode.c

This main file decompresses infile and the decompressed output goes to outfile

Pseudo Code:

Create a table to store the dictionary entries

Initialize variables curr_sym and curr_code to 0

Initialize next_code to START_CODE

While read_pair(infile, &curr_code, &curr_sym, BIT-LENGTH(next_code)) returns true

Store the current pair in the table at index next_code as a word by appending curr_sym to the word at index curr_code

Write the word at index next_code to the output fileIncrement next_code

If next_code is equal to MAX_CODE, reset the table

Set next_code to START_CODE

Flush the remaining words in the write buffer