DESIGN

Wenyu Liang

March 13, 2023

1 Program Description

This program will implement Lempel-Ziv Compression algorithm, LZ78 specifically. This is a dictionary-based algorithm, where the key is a prefix, also known as a word, and the value is a code that we can utilize for fast look-ups, during compression. The decompression is very similar, except that the key and value for the decompressing dictionary is swapped. Users can compress and decompress their data with this program.

2 Files to be included in directory asgn6

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**: This is the source file for the Trie ADT.
4. **trie.h**: This is the header file for the Trie ADT.
5. **word.c**: This is the source file for the Word ADT.
6. **word.h**: This is the header file for the Word ADT.
7. **io.c**: This is the source file for the I/O module.
8. **io.h**: This is the header file for the I/O module.
9. **endian.h**: This is the header file for the endianness module.
10. **code.h**: This is a header file containing macros for reserved codes.

3 Structure

trie will help searching strings with identical prefix, which will be used in compression. word will saves the words, used in decompression. io.c will help read and write files. encode.c contains the main() function for the encode program and decode.c contains the main() function for the decode program.

4 Pseudocode

1. **tire.c**

   ```
   TrieNode *trie_node_create(uint16_t index){
       TrieNode *node = (TrieNode*)malloc(sizeof(TrieNode));
       node -> code = index;
       for (i in range(0, ALPHABET)
       {
           node -> children[i] = NULL;
       }
       return node;
   }

   void trie_node_delete(TrieNode *n){
       reverse step of trie_node_create using free;
   }

   TrieNode *trie_create(void){
       call trie_node_create;
       return node == NULL ? NULL : node;
   }

   void trie_reset(TrieNode *root){
       for child in children:
           trie_delete(child);
   }

   void trie_delete(TrieNode *n){
       for (uint16_t i  = 0; i < ALPHABET; i++) {
           trie_delete(n->children[i]);
           n-> children[i] = NULL;
       }
   }

   TrieNode *trie_step(TrieNode *n, uint8_t sym){
       return n->children[sym] != NULL ? n->children[sym]: NULL;
   }
   ```

2. **word.c**

   ```
   Word *word_create(uint8_t *syms, uint32_t len){
       malloc for Word;
       malloc for syms;
   }

   Word *word_append_sym(Word *w, uint8_t sym){
   ```

```
    create new word by increase len+1;
}

void word_delete(Word *w){
    free memory for syms;
    free memory for word;
}

WordTable *wt_create(void){
    WordTable* word = calloc(UINT16_MAX,sizeof(word));
}

void wt_reset(WordTable *wt){
    pointer accept the one in index 1 will be null;
}
```

3. **io.c**

```
function buf_clear():
    for i in range(BLOCK):
        buf_pair[i] = 0

function read_bytes(infile , buf, to_read):
    total_read = 0
    num_read = 0
    buf_ptr = buf

    while to_read > 0:
        num_read = read(infile , buf_ptr , to_read)
        if num_read == -1:
            return -1
        else if num_read == 0:
            break
        else:
            buf_ptr += num_read
            total_read += num_read
            to_read -= num_read
    return total_read

function write_bytes(outfile , buf, to_write):
    total_write = 0
    num_write = 0
    buf_ptr = buf

    while to_write > 0:
        num_write = write(outfile , buf_ptr , to_write)
        if num_write == -1:
            return -1
        else if num_write == 0:
            break
        else:
            buf_ptr += num_write
            total_write += num_write
            to_write -= num_write
    return total_write

function read_header(infile, header):
    total_bits += 8 * read_bytes(infile, header, sizeof(FileHeader))
    if big_endian():
        header.magic = swap32(header.magic)
        header.protection = swap16(header.protection)
    if header.magic != 0xBAADBAAC:
        fprintf(stderr, "Error:␣Unmatched␣magic␣number.\n")
        exit(1)

function write_header(outfile, header):
    if big_endian():
        header.magic = swap32(header.magic)
        header.protection = swap16(header.protection)
    total_bits += 8 * write_bytes(outfile, header, sizeof(FileHeader))

function write_pair(outfile, code, sym, bitlen):
    sym_bit = 8
    bit_to_write = 0
    while bitlen > 0:
        bit_to_write = code & 1
        buf_pair[buf_pair_pos] |= (bit_to_write << (8 - bit_left))
        code >>= 1
        bitlen -= 1
        bit_left -= 1
        if bit_left == 0:
            buf_pair_pos += 1
            bit_left = 8
        if buf_pair_pos == BLOCK:
            total_bits += 8 * write_bytes(outfile, buf_pair, BLOCK)
            buf_clear()
            buf_pair_pos = 0

    while sym_bit > 0:
        bit_to_write = sym & 1
        buf_pair[buf_pair_pos] |= (bit_to_write << (8 - bit_left))
        sym >>= 1
        sym_bit -= 1
        bit_left -= 1
        if bit_left == 0:
            buf_pair_pos += 1
            bit_left = 8
        if buf_pair_pos == BLOCK:
            total_bits += 8 * write_bytes(outfile, buf_pair, BLOCK)
            buf_clear()
            buf_pair_pos = 0

function flush_pairs(outfile):
    if bit_left == 8:
        total_bits += 8 * write_bytes(outfile, buf_pair, buf_pair_pos)
    else:
        total_bits += 8 * write_bytes(outfile, buf_pair, buf_pair_pos + 1)
    buf_pair_pos = 0
    bit_left = 8

function write_word(outfile: int, w: Word pointer) {
    for i from 0 to w->len-1 {
        buf_sym[buf_sym_pos] = w->syms[i]
        buf_sym_pos += 1
        if buf_sym_pos >= BLOCK {
        total_syms += write_bytes(outfile, buf_sym, BLOCK)
        buf_sym_pos = 0
        }
    }
}

function flush_words(outfile: int) {
    // write out the rest in the buffer
    total_syms += write_bytes(outfile, buf_sym, buf_sym_pos)
    buf_sym_pos = 0
}

function read_pair(infile, code, sym, bitlen) -> boolean
    if buf_pair_pos >= buf_pair_len
        buf_pair_len = read_bytes(infile, buf_pair, BLOCK)
        total_bits += 8 * buf_pair_len
        buf_pair_pos = 0

    if buf_pair_len == 0
        return false

    code = 0
    bit_processed = 0
    while bitlen > bit_processed
        bit_to_write = ((buf_pair[buf_pair_pos] >> (8 - bit_left)) & 1) << b
        code |= bit_to_write
        bit_processed += 1
        bit_left -= 1
        if bit_left == 0
            buf_pair_pos += 1
            bit_left = 8

        if buf_pair_pos == buf_pair_len then
            buf_clear()
            buf_pair_len = read_bytes(infile, buf_pair, BLOCK)
            total_bits += 8 * buf_pair_len
            buf_pair_pos = 0

    sym = 0
    sym_bit_processed = 0
    while sym_bit_processed < 8
        bit_to_write = ((buf_pair[buf_pair_pos] >> (8 - bit_left)) & 1) << s
        sym |= bit_to_write
        sym_bit_processed += 1
        bit_left -= 1
        if bit_left == 0
            buf_pair_pos += 1
            bit_left = 8
        if buf_pair_pos == buf_pair_len then
            buf_clear()
            buf_pair_len = read_bytes(infile, buf_pair, BLOCK)
            buf_pair_pos = 0
            total_bits += 8 * buf_pair_len

    if code == STOP_CODE
        return false

    return true

function read_sym(infile, sym) -> boolean
    if buf_sym_pos >= buf_sym_size
        buf_sym_size = read_bytes(infile, buf_sym, sizeof(buf_sym))
        buf_sym_pos = 0
    if buf_sym_size == 0
        return false
    sym = buf_sym[buf_sym_pos]
    buf_sym_pos += 1
    total_syms += 1
    return true
```

4. **encode.c, decode.c** will follow the pseudocode provided in the resources!