Assignment 6 – Huffman Coding

Yazmyn Sims

CSE 13S – Spring 2023

Purpose

This program aims to produce a data compression using Huffman's Code.

How to Use the Program

The user will have to type command line options (-i: Sets the name of the input file, -o: Sets the name of the output file, -h: will remind the user how to use the command line options) into vim followed by a filename after i and o. For example user input could look like:

```
./huff -i files/zero.txt -o files/zero.huff
```

OUTPUT: The output will be a compressed file.

Program Design

Data Structures

- Struct: groups several related variables into one place. Like an array that can hold elements of different types.
  - Used in bitwriter.c, node.c, pq.c, and huff.c.
- Buffer: use a single, constant-size buffer as though they link end to end
  - Used in bitwriter.c, huff.c
- File: representations for data
  - Used in bitwriter.c and huff.c
- Priority Queue: a queue where each element has its own level of priority. High priority is served first.
  - Used in pq.c and huff.c.
- Tree: a collection of nodes connected by edges
  - Used in node.c, pq.c and huff.c.

Algorithms

- Huffman Coding Algorithm

Example Pseudocode for compressing a file:

```
huff_compress_file(outbuf, inbuf, filesize, num_leaves, code_tree, code_table)
8 'H'
8 'C'
32 filesize
16 num_leaves
huff_write_tree(outbuf, code_tree)
for every byte b from inbuf
    code = code_table[b].code
    code_length = code_table[b].code_length
    for i = 0 to code_length - 1
        /* write the rightmost bit of code */
       1 code & 1
       /* prepare to write the next bit */
       code >>= 1
```

Function Descriptions

BitWriter Functions

- BitWriter *bit_write_open(const char *filename);
    -
- void bit_write_close(BitWriter **pbuf);
    -
- void bit_write_bit(BitWriter *buf, uint8_t x);
    -
- void bit_write_uint8(BitWriter *buf, uint8_t x);
    -
- void bit_write_uint16(BitWriter *buf, uint16_t x);
    -
- void bit_write_uint32(BitWriter *buf, uint32_t x);
    -

Node Functions

- Node *node_create(uint8_t symbol, double weight);
    -
- void node_free(Node **node);
    -
- void node_print_tree(Node *tree, char ch, int indentation);
    -
- void node_print_tree(Node *tree, char ch, int indentation) {
    -

Priority Queue Functions

- PriorityQueue *pq_create(void);
    -

- void pq_free(PriorityQueue **q);
    -
- bool pq_is_empty(PriorityQueue *q);
    -
- bool pq_size_is_1(PriorityQueue *q);
    -
- void enqueue(PriorityQueue *q, Node *tree);
    -
- bool dequeue(PriorityQueue *q, Node **tree);
    -
- void pq_print(PriorityQueue *q);
    -
- bool pq_less_than(Node *n1, Node *n2)
    -

Huffman Coding Functions

- uint64_t fill_histogram(Buffer *inbuf, double *histogram)
    -
- Node *create_tree(double *histogram, uint16_t *num_leaves)
    -
- fill_code_table(Code *code_table, Node *node, uint64_t code, uint8_t code_length)
    -
- void huff_compress_file()
    -
- huff_compress_file(outbuf, inbuf, filesize, num_leaves, code_tree, code_table)
    -
- huff_write_tree(outbuf, node)
    -

Results

N/A

References

N/A

N/A

Figure 1: N/A