DESIGN - Assignment 4

Brian Nguyen

October 28, 2021

1 Description

This program contains an implementation of the Huffman encoder and decoder used to compress a data file. With this, the encoder has the job of compressing the file as it receives and reads an input then assigns the least amount of bits to the most common symbol and the greatest amount of bits to the least common symbol. The decoder does the opposite by reverting the compressed file back to its original raw size.

2 Files

1. encode.c
    - This source file contains the code for the Huffman encoder.
2. decode.c
    - This source file contains the code for the Huffman decoder.
3. node.c
    - This source file contains the code for the node abstract data type.
4. pq.c
    - This source file contains the code for the priority queue abstract data type.
5. code.c
    - This source file contains the code for the code abstract data type.
6. io.c
    - This source file contains the code for the I/O module.
7. stack.c

- This source file contains the code for the stack abstract data type.

8. huffman.c
    - This source file contains the code for the Huffman coding module.
9. defines.h
    - This header file contains the macros for this program.
10. header.h
    - This header file contains the struct definitions.
11. node.h
    - This header file contains the interface for the node abstract data type.
12. pq.h
    - This header file contains the interface for the priority queue abstract data type.
13. code.h
    - This header file contains the interface for the code abstract data type.
14. io.h
    - This header file contains the interface for the I/O module.
15. stack.h
    - This header file contains the interface for the stack abstract data type.
16. huffman.h
    - This header file contains the interface for the Huffman coding module.
17. Makefile
    - This make file contains the code that builds and compiles the program to be run. It also cleans all compiler generated files and formats the code to be submitted.
18. README.md
    - This markdown file describes the program, how to build it, how to run it, and also lists and explains all the command-line options that the program accepts. It also documents any false positives given by scan-build.

19. DESIGN.pdf

    - This pdf is the manual that explains the program, files included, layout or structure, and pseudo-code of the program.

3 Structure

4 Pseudo-code

4.1 Encoder

```
define encoder
    use getopt
    use switch cases (h,i,o,v)
    #for v print with 100 *(1 (compressed size/uncompressed size)) for space saving
    get occurences of each symbol
    call pq() # to create huffman tree
    construct code table #to decipher the symbols
    emit encoding of huffman tree to file #to traverse the tree
    convert each symbol of input file to output
```

4.2 Decoder

```
define decoder
    use getopt
    use switch cases (h,i,o,v)
    #for v print with 100 *(1 (compressed size/uncompressed size)) for space saving
    read emitted tree from input
    call on stack() to make nodes for Huffman tree reconstruction
    read and traverse tree
    #0 for left side and 1 for right side, when node reach, emit symbol and start over
```

4.3 Node

```
define Node structure
    initialize Node left
    initialize Node right
    initialize symbol
    initialize freq

define node create
```

```
    make node using struct, symbol and frequency

define node_delete
    free node pointer
    null node memory

define node_join
    join left | right node #left child = left, right child = right
    parent node symbol = '$'
    frequency of symbol = sum of left and right
    return parent node pointer

define node_print
    print out nodes for debug
```

4.4 Priority Queue

```
define pq struct
    initialize capacity

define pq_create
    set max capacity of pq to capacity var

define pq_delete
    free pq pointer
    null pq memory

define pq_empty
    if pq empty
        return true
    return false

define pq_full
    if pq full
        return true
    return false

define pq_size
    return items in pq
```

4.5 Code

```
define code_init
    set top to 0
    zero array bits
    return code

define code_size
    return size of code

define code_empty
    if code empty
        return true
    return false

define code_full
    if code full
        return true
    return false

define code_set_bit
    if i out of range
        return false
    set bit index i to 1
    return true

define code_clr_bit
    if i out of range
        return false
    set bit index i to 0
    return true

define code_get_bit
    if i out of range or bit index i is 0
        return false
    return true

define code_push_bit
    if code is full
```

```
        return false
    push bit to code
    return true

define code_pop_bit
    if code is empty
        return false
    pop bit from stack to pointer
    return true

define code_print
    print code push and pop for debugging
```

4.6 I/O

```
define read_bytes
    loop through infile to read all bytes in file
    put into nbytes buffer
    return number of bits read

define write_bytes
    loop through outfile to read all bytes in file
    put into nbytes buffer
    return number of bits read

define read_bit
    initialize static buffer and index
    use buffer to read through infile
    scan bit and send to a pointer to store
    if no more bits
        return false
    return true

define write_code
    initialize static buffer and index
    use buffer to write bit one by one
    if buffer of block index is bits
        write to outfile with buffer
```

```
define flush_codes
    write extra leftover bits
    set extra bits to 0
```

4.7 Stack

```
define stack structure
    initialize top
    initialize capacity
    initialize Node (items pointer)
define stack_create
    initialize stack pointer
    set capacity to max number of nodes in stack

define stack_delete
    free stack pointer
    null stack memory

define stack_empty
    if stack empty
        return true
    return false

define stack_full
    if stack full
        return true
    return false

define stack_size
    return number of nodes in stack

define stack_push
    if stack full
        return false
    push node to stack
    return true

define stack_pop
    if stack empty
```

```
        return false
    pop node from stack to pointer
    return true

define stack_print
    print stack for debugging
```

4.8 Huffman

```
define build_tree
    construct Huffman tree with histogram

define build_codes
    call code table
    build code of each symbol to Huffman tree
    copy to code table

define dump_tree
    post-order traverse the Huffman tree #write to outfile too
    while loop
        write L for leaf
        I: interior nodes #do not write a symbol for it though

define rebuild_tree
    reconstruct Huffman tree with tree_dump array
    set length of tree dump to nbytes
    return root node

define delete_tree
    free all nodes in tree using post-order traverse
    set all nodes to null
```

5 Error Handling

6 Credits

1. I used the asgn5.pdf from Professor Long for explanations and pseudocode.