Assignment 8

Design: LZW Compression

Linhao Chen
CSE 13S - Fall 2019

Date:12/05/2019

1 Introduction

This program will compress and decompress a file by using LZW compression. For compression, it will use Trie ADT. And it will use Word Table for decompression. The program can only decompress the file which is compressed by the function. Expected to use trie, word table, i/o, etc.

2 Trie

```
Trie_node_create (Parameter code) {
    Allocate memory for TrieNode
    Test whether allocate successfully
    Set the code of node by using parameter
    return
}

Trie_node_delete (Parameter Pointer to Node) {
    If (pointer is not NULL)
        Free the memory allocated for node
    Return
}

Trie_create {
    Create Root by using trie_node_create and pass 0 as code
    For (I from 0 to num of ascii chars (256))
        Create the node Children[i] by using trie_node_create and pass i as code
    return
}
```

```
Trie_reset (Parameter Pointer to Node) {
    Create a new root by using trie_create
    Delete the previous root
    Set the pointer from root to new root
    return
}

Trie_delete (Parameter Pointer to Node) {
    If (pointer is not NULL)
        For (I from 0 to num of ascii chars (256))
            Delete the node Children[i] by using trie_delete
        Delete itself by using trie_node_delete
        Set the pointer to NULL
    return
}

Trie_step (Parameter Pointer to Node, sym) {
    Find the children[sym]
    If NULL
        Return NULL pointer
    Else
        Return the node
}
```

3 Word Table

```
Word_create (Parameter Word and Length) {
    Allocate memory for word
    Test whether allocate successfully
    Set the Word and length for Word
}

Word_delete (Parameter Pointer of table) {
    If pointer does not point to NULL
        Free the table
        Set the pointer to NULL
    return
}

wt_create {
    Allocate the memory for table
    Test whether allocate successfully
    For (I from 0 to num of ascii chars (256))
```

```
            Create the word entry[i] by using word_create and pass i and 1 as word and length
        return

}

wt_reset (Parameter the pointer to word table) {
        Create a new table by wt_create
        Delete the previous table
        Set the pointer from table to new table
        return
}

wt_delete (Parameter the pointer to word table) {
        If (pointer is not NULL)
            For (i from 0 to uint16 MAX)
                Delete the node Children[i] by using word_delete
            Free the memory of word table
            Set the pointer to NULL
        return
}
```

4 I / O

```
Global Variable Buffer for word
Global Variable Buffer for char
Global Variable Buffer for code
Global Variable Counter for word
Global Variable Counter for char
Global Variable Counter for code

Read_header (Parameter infile, Header) {
        Read the Header in infile to Header struct
        If magic number is not match, exit the program
        Else return
}

Write_header (Parameter outfile, Header) {
        Write the data in Header Struct to outfile
        Return
}


Next_char (Parameter infile) {
```

```
    If (buffer for char is NULL)
        Read 4kb block from infile
        Set the counter to 0
    Output = buffer [counter]
    Make buffer [counter] empty
    Return output
}


Buffer_code (Parameter outfile, code, bit length) {
    If (buffer for code is full)
        Write the code to outfile (size is 4kb block)
        Empty the buffer
        Set counter to 0
    Use the counter to find the current location for store the code
    Store the code into buffer (the length of data is parameter bit length)
    Counter add bit length
    return
}

Flush_code (Parameter outfile) {
    Write all code in buffer to outfile (size is the counter of code buffer)
    Return
}


Next_code (Parameter infile, bit_len) {
    If (Buffer of word is empty)
        Read 4kb block to buffer
        Set the counter to zero
    Use counter to find the current location for reading a code
    Read (bit_len) bits code from Buffer
    Make these part to empty
    Counter add bit_len
    Return the code
}


Buffer_word (Parameter outfile) {
    If (Buffer of word is full)
        Write the buffer to outfile (size is 4kb block)
        Empty the buffer
        Set the counter to zero
    Store one word
```

```
    Increment counter 1
    return
}


Flush_word (Parameter outfile) {
    Write all word in buffer to outfile (size is the counter of word buffer)
}
```

5 Main

```
Set the bool variables for verbose/ compress/ decompress mode
Set the bool variables for know whether user provides the infile and outfile
Set the bool variables for check whether choose c/d mode twice
  Switch to case depend on the command-line option
    In case ‘v’
      User choose verbose function, set the bool variable to true

    In case ‘i’
      Set the infile variable to false
      Read the file by using provided argument

    In case ‘o’
      Set the outfile variable to false
      Read the file by using provided argument

    In case ‘c’
       If the bool variable of twice mode is true
          Print error: user cannot choose -c -d at same time
      Else
          Set the compress mode bool to true


    In case ‘d’
      If the bool variable of twice mode is true
          Print error: user cannot choose -c -d at same time
      Else
          Set the decompress mode bool to true

If (infile bool is true)
    Read the infile from stdin
If (outfile bool is true)
```

Set stdout as outfile

5.1 Compression

This part Pseudocode comes from Eugene in Assignment Sheet

```
Read the statistic from infile and store in header
Write the header to outfile
Create a Trie and get the pointer to root
Set a pointer of TrieNode as curr_node to root
Set a pointer of TrieNode as next_node
Set uint8_t variable curr_char
set uint16_t variable next_avail_code to 256
set uint64_t variable encoded_chars to 0
while (encoded_chars != header.file_size):
    curr_char = next_char()
    next_node = trie_step(curr_node, curr_char)
    if(encoded_chars == 0 or next_node != NULL):
        curr_node = next_node
    else:
        bit_len = log2(next_avail_code)+1
        buffer_code(curr_code,bit_len)
        curr_node.children[curr_char] = trie_node_create(next_avail_code)
        curr_node = root.children[curr_char]
        next_avail_code += 1
    if(next_avail_code == UINT16_MAX):
        trie_reset(root)
        curr_node = root.children[curr_char]
        next_avail_code = 256
bit_len = log2 (next_avail_code) + 1
buffer_code (curr_code, bit_len)
flush_codes ()
```

5.2 Decompression

This part Pseudocode comes from Eugene in Assignment Sheet

```
read_header from infile and check the Magic Number
create a word table
set next_avail_code to 256
set bool variable reset to false
while (decoded_chars != header.file_size):
    bit_len = log2 (next_avail_code + 1) + 1
```

```
    curr_code = next_code (bit_len)
    curr_entry = table [curr_code]
    if (decoded_chars == 0 or reset):
        buffer_word (curr_entry)
        prev_word = curr_char reset = false
    else if (curr_entry != NULL):
        curr_word = curr_entry.word
        prev_entry = table [prev_code]
        prev_word = prev_entry . word
        new_word = prev_word . append (curr_word [0])
        table [next_avail_code] = word_create (new_word)
        next_avail_code += 1
        buffer_word (curr_entry)
        decoded_chars += curr_entry.length
    else:
        prev_entry = table [prev_code]
        prev_word = prev_entry.word
        curr_word = prev_word . append (prev_word [0])
        missing_entry = word_create (curr_word)
        table [ next_avail_code ] = missing_entry next_avail_code += 1
        buffer_word (missing_entry)
        decoded_chars += missing_entry.length
        prev_code = curr_code
    if ( next_avail_code == UINT16_MAX - 1) :
        wt_reset ()
        next_avail_code = 256
        reset = true
flush_words ()
```

5.3 Verbose Mode

Print the Original file size
Print the Compressed file size
Calculate the compress ratio and print out (ratio = (100 * (1 – (ori_size / comp_size)))
Print out the longest word size