Assignment 7 – Huffman Coding

Jayden Sangha

CSE 13S – Winter 24

Purpose

Audience for this section: Pretend that you are working in industry, and write this paragraph for your boss. You are answering the basic question, "What does this thing do?". This section can be short. A single paragraph is okay.

Do not just copy the assignment PDF to complete this section, use your own words.

The purpose of these programs is to losslessly compress and decompress any given files.

Questions

Please answer the following questions before you start coding. They will help guide you through the assignment. To make the grader's life easier, please do not remove the questions, and simply put your answers below the text of each question.

- Describe the goal of compression. (As a hint, why is it easy to compress the string "aaaaaaaaaaaaaaaa")

  The goal of compression is to lower data size in order to save space and increase run speed.
- What is the difference between lossy and lossless compression? What type of compression is huffman coding? What about JPEG? Can you lossily compress a text file?

  Lossy compression compressed data resembles the original data and it is not reversible. Lossless compression compressed data does not resemble the original data, is reversible, and must be decompressed. JPEG files are lossy compressed and Huffman code is lossless compressed.
- Can your program accept any file as an input? Will compressing a file using Huffman coding make it smaller in every case?

  Yes our program can accept any file, but huffman code only makes it smaller if the file can be losslessly compressed.
- How big are the patterns found by Huffman Coding? What kind of files does this lend itself to?
- Take a picture on your phone. What resolution is the picture? How much space does it take up in your storage (in bytes)?

  The pictures resolution is 3024 x 4032 and it takes up 2200000 bytes of storage space.
- If each pixel takes 3 bytes, how many bytes would you expect the picture you took to take up? Why do you think that the image you took is smaller?

  I would expect the picture to be 12,192,768 bytes. I think it is smaller because my phone automatically compresses the photo.
- What is the compression ratio of the picture you just took? To get this, divide the actual size of the image by the expected size from the question above. You should not get a number above 1.

The compression ratio of my photo is .18

- Do you expect to be able to compress the image you just took a second time with your Huffman program?1 Why or why not?

  I do not believe that I could compress the image anymore, because the phone would have already compressed it more.

- Are there multiple ways to achieve the same smallest file size? Explain why this might be.

  There are multiple ways to achieve a smaller file size, because you can compress files lossly or losslessly.
- When traversing the code tree, is it possible for a internal node to have a symbol?

  No it is not possible, symbols are stored in the leaves.
- Why do we bother creating a histogram instead of randomly assigning a tree.

  We create a histogram, because it is easier to compress and shows us the frequency of symbols.
- Relate this Huffman coding to Morse code. Why does Morse code not work as a way of writing text files as binary? What if we created more symbols for things like spaces and newlines?

  Morse code does not work, because it is a character scheme, which means it would take up a lot of storage and it does not have code for symbols. If we added symbols for spaces and newlines it would work but be very slow and large.
- Using the example binary, calculate the compression ratio of a large text of your choosing

N/A

Testing

List what you will do to test your code. Make sure this is comprehensive. [1] Be sure to test inputs with delays and a wide range of files/characters.

How to Use the Program

Audience: Write this section for the user of your program. You are answering the basic question, "How do I use this thing?". Don't copy the assignment exactly; explain this in your own words. This section will be longer for a more complicated program and shorter for a less complicated program. You should show how to compile and run your program. You should also describe any optional flags or inputs that your program uses, and what they do.

First you must compile the code by running

```
make
```

then to compress a file you enter

```
./huff -i (inputfile name - the orginal file) -o (output file name - new outfile)
```

then to decompress a file

```
./dehuff -i(outfile name - new outfile name)
```

```
Here is some code in lstlisting.
```

And if you want a box around the code text, then use `\begin{lstlisting}[frame=single]` and `\end{ lstlisting}`

which will look like this:

```
Here is some framed code (lstlisting) text.
```

Want to make a footnote? Here's how.[2]

Do you need to cite a reference? You do that by putting the reference in the file `bibtex.bib`, and then you cite your reference like this[1][2][3].

[1] This question is a whole lot more vague than it has been the last few assignments. Continue to answer it with the same level of detail and thought.

[2] This is my footnote.

Program Design

Audience: Write this section for someone who will maintain your program. In industry you maintain your own programs, and so your audience could be future you! List the main data structures and the main algorithms. You are answering the basic question, "How is this thing organized so that I can have a chance of fixing it?". This section will be longer for a more complicated program and shorter for a less complicated program.

Pseudocode

Give the reader a top down description of your code! How will you break it down? What features will your code have? How will you implement each function.

Def *bit_write_open(filename) Allocate a BitWriter object Create buffer using write_open buf-¿underlying_stream = underlying_stream Return a pointer to buf Report error if unable to perform prior steps

Def bit_writer_close(**pbuf) If bit_position ¿ 0 Write byte to underlying_stream write_close() Free *pbuf Set *pbuf to NULL

Def bit_write_bit(buf, x) If bit_position ¿ 7 Write byte to underlying_stream using write_uint8() Clear byte to 0x00 Clear bit_position to 0 If x & 1 then byte —= (x & 1) ¡¡ bit_position ++bit_position

Def bit_write_uint8(buf, x) For i = 0 to 7 Write bit i of x using bit_write_bit()

Def bit_write_uint16(buf, x) For i = 0 to 15 Write bit i of x using bit_write_bit()

Def bit_write_uint32(buf, x) For i = 0 to 31 Write bit i of x using bit_write_bit() node.c

Def node_create(symbol, weight) Create node and set symbol and weight, return node pointer

Def node_free(**node) Free node and set to null

pq.c

Def pq_create(void) Allocate a new priorityqueue and return a pointer using calloc()

Def pq_free(**q) Free the priorityqueue and set it to NULL

Def pq_is_empty(*q) Return true if the priorityqueue is empty.

Def pq_size_is_1(*q) Return false if q is empty Return true of the next element of queue's list is NULL

Def pq_less_than(*n1, *n2) If weight of n1 is less than weight of n2, return true If weight of n1 is greater than weight of n2, return false Return n1's symbol is less than n2's symbol

Def enqueue(*q, *tree) Allocate new listelement e, and set e-¿tree = tree If queue empty Set q-¿list = e If weight of the new tree is less than the weight of the head Set e-¿next to e-¿list q-¿list = e If new element goes after one existing element Starting q-¿list, go through the list until the next field is NULL Insert e after the queue element

Def dequeue(*q, **tree) If queue is empty return false otherwise set *tree to e-¿tree Call free on e, return true

huff.c

Def fill_histogram(inbuf, histogram) Create uint64 variable filesize and set it to 0 For i to 256 Set histogram to 0 Create uint8 variable x While read_uint8 is true Increment histogram Increment filesize Increment histogram by 0x00 Increment histogram by 0xff Return filesize

Def create_tree(histogram, num_leaves) Create and fill a priority queue Run huffman coding algorithm Dequeue the queue's only entry and return

Def fill_code_table(code_table, node, code, code_length) If node is internal fill_code_table(code_table, node-¿left, code, code_length + 1); Code —= 1 ¡¡ code_length fill_code_table(code_table, node-¿right, code, code_length + 1); Else code_table[node_¿symbol].code = code; code_table[node-¿symbol].code_length = code_length;

Function Descriptions

For each function in your program, you will need to explain your thought process. This means doing the following

- The inputs of every function (even if it's not a parameter)
- The outputs of every function (even if it's not the return value)
- The purpose of each function, a brief description about a sentence long.
- For more complicated functions, include pseudocode that describes how the function works
- For more complicated functions, also include a description of your decision making process; why you chose to use any data structures or control flows that you did.

Do not simply use your code to describe this. This section should be readable to a person with little to no code knowledge. **DO NOT JUST PUT THE FUNCTION SIGNATURES HERE. MORE EXPLANATION IS REQUIRED.**

BitWriter *bit_write_open(const char *filename); This function takes a filename and opens it. It then returns a pointer to a bitwriter struct in the file.

void bit_write_close(BitWriter **pbuf); This function takes a pointer to a bitwriter and closes the file and frees the writer.

void bit_write_bit(BitWriter *buf, uint8_t bit); This function takes a pointer to a bitwriter and a bit. This function writes the given bit into the given writer.

void bit_write_uint8(BitWriter *buf, uint8_t x); This function takes a pointer to a bitwriter and 8 bits. It then writes all of the bits into the writer.

void bit_write_uint16(BitWriter *buf, uint16_t x); This function takes a pointer to a bitwriter and 16 bits. It then writes all of the bits into the writer.

void bit_write_uint32(BitWriter *buf, uint32_t x); This function takes a pointer to a bitwriter and 32 bits. It then writes all of the bits into the writer.

BitReader *bit_read_open(const char *filename); This function takes a filename opens it and returns a pointer to a bitreader.

void bit_read_close(BitReader **pbuf); This function takes a pointer to a bitreader and frees the reader and closes the file it is in.

uint8_t bit_read_bit(BitReader *buf); This function takes a pointer and reads the bit that it is pointing at.

uint8_t bit_read_uint8(BitReader *buf); This function takes a pointer and reads the 8 bits that it is pointing at.

uint16_t bit_read_uint16(BitReader *buf); This function takes a pointer and reads the 16 bits that it is pointing at.

uint32_t bit_read_uint32(BitReader *buf); This function takes a pointer and reads the 32 bits that it is pointing at.

Node *node_create(uint8_t symbol, uint32_t weight); This function takes a symbol and weight. It then creates a node with the given weight and symbol.

void node_free(Node **pnode); This function takes a pointer to a node and frees it.

void node_print_tree(Node *tree); This function takes a pointer to a tree and prints out all of the values in a tree.

PriorityQueue *pq_create(void); This function creates a priority queue object and then returns a pointer to it.

void pq_free(PriorityQueue **q); This function takes a pointer to a priority queue and then frees it.

bool pq_is_empty(PriorityQueue *q); This function takes a pointer to a priority queue and checks if its empty then returns the result.

bool pq_size_is_1(PriorityQueue *q); This function takes a pointer to a priority queue and checks if it contains only 1 element then returns true or false.

bool pq_less_than(ListElement *e1, ListElement *e2) This function takes 2 pointers to different elements in a list and returns true if the first element is less than the second.

void enqueue(PriorityQueue *q, Node *tree); This function takes a pointer to a priority queue and a new pointer to a tree. This function then creates a tree with the given pointer inside of the given priority queue.

Node *dequeue(PriorityQueue *q); This function takes a pointer to a priority queue then removes and returns the item with the smallest weight.

void pq_print(PriorityQueue *q); This function takes a pointer to a priority queue and prints all of the trees in the pointer queue.

uint32_t fill_histogram(FILE *fin, uint32_t *histogram) This function updates the number of each distinct byte value from the input file in a histogram array of uint32_t values. The entire size of the input file is also returned.

Node *create_tree(uint32_t *histogram, uint16_t *num_leaves) This function takes a histogram and the number of leaves in the tree. It then creates a huffman tree with the given number of leaves and returns the number of leaf nodes in the tree.

fill_code_table(Code *code_table, Node *node, uint64_t code, uint8_t code_length) This function takes a code table, node, code, and code length. This function then goes through the table and files in the code table with the symbol of every leaf node.

void huff_compress_file(outbuf, fin, filesize, num_leaves, code_tree, code_table) This function takes a pointer to a bitwriter, pointer to a file object, the file size, number of leaves in the tree, a pointer to the root of the tree, and a pointer to a code table. This function compresses a file using huffman code.

void dehuff_decompress_file(FILE *fout, BitReader *inbuf) This function takes a file and a pointer to a bitreader. It then decompresses the given files using huffman code and returns the decompressed file.

References

[1] Wikipedia contributors. C (programming language) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/C_(programming_language), 2023. [Online; accessed 20-April-2023].

[2] Robert Mecklenburg. *Managing Projects with GNU Make, 3rd ed.* O'Reilly, Cambridge, Mass., 2005.

[3] Walter R. Tschinkel. Just scoring points. *The Chronicle of Higher Education*, 53(32):B13, 2007.