Assignment 3

cpe 357 Spring 2011

"I’d crawl over an acre of ’Visual This++’ and ’Integrated Development

That’ to get to gcc, Emacs, and gdb. Thank you." (By Vance Petree, Virginia Power)

— /usr/games/fortune

Due by 11:59:59pm, Friday, May 6th.

This assignment is to be done **individually**.

Programs: hencode and hdecode

This assignment is to build a file compression tool, hencode, that will use Huffman codes to compress a given file, and a file decompression tool, hdecode, that will uncompress a file compressed with hencode.

Usage:

hencode infile [ outfile ]

hdecode [ ( infile | - ) [ outfile ] ]

hencode must:

• take a command line argument that is the name of the file to be compressed; and

• take a second, optional, command line argument that is the name of the outfile. (If this argument is missing, the output should go to stdout.)

• properly encode its input into a binary Huffman-coded file according to the algorithm below.

hdecode must:

• Take an optional command line argument that is the name of the file to be uncompressed. If this argument is missing, or if the input file name is “-”, hdecode will take its input from standard input.

• Take a second, optional, command line argument that is the name of the output file. (If this argument is missing, the output should go to stdout.)

• Properly decode its input and restore the original data. Both must:

• use only Unix unbuffered IO (read(2) and write(2)) for reading and writing the files; and

• share as much of the code base as is reasonable between the two programs; and

• demonstrate robust programming techniques including proper handling of errors; and

• adhere to a “reasonableness” standard with respect to performance. See also the “Pesky Details” section below.

Huffman Codes

In 1952, David A. Huffman proposed a method for compressing files by generating a variable length encoding of characters based on the relative frequency with which each character occurs in a file4 . This encoding is what is known as prefixless code because no letter encoding is a prefix of any other encoding. In practice, this means that the coding can be stored in a binary tree with the characters at the leaves. Each character’s encoding can be determined by looking at the sequence of right or left child node transitions and representing those as either zero bits or one bits.

Generating the Tree

Note: Be sure to follow the tiebreaker conventions outlined in this document. If you do something different, you will still get a valid encoding of your file, but will be different from the reference one. That means that both programs will have to work perfectly all the way through in order to be tested. If you follow the tiebreakers partial credit becomes a possibility.

To build the tree, it is first necessary to generate a histogram of all the characters in the file, remembering that any value from 0 to 255 represents a valid byte. We will demonstrate the algorithm through the following encoding example.

If the file you are encoding consists of the string “bmmmcccccttttttaaaaaaa” the histogram will be:

|  |  |
| --- | --- |
| Byte | Count |
| 0x61 (’a’)  0x62 (’b’)  0x63 (’c’)  0x6d (’m’)  0x74 (’t’) | 7  1  5  3  6 |

Take these counts and generate a linked list of them in ascending order of frequency. If there is a tie in frequency, do a secondary sort and place the characters also in ascending order. For the histogram above, this list will look like:

0x62 (’b’)

1

0x6d (’m’)

3

0x63 (’c’)

5

0x74 (’t’)

6

0x61 (’a’)

7

Now, remove the first two nodes on the list and construct a new node whose frequency is the sum of the two removed nodes. Make the first removed node the left child of the new node and the second removed node the right subchild:

4

0 1

0x62 (’b’)

1

0x6d (’m’)

3

Next, reinsert the new node into the remaining list of nodes in order. If there is a frequency tie, insert the new node before the other node in the list. After reinsertion of the new node, our list will look like:

4 David A. Huffman. A method for the construction of minimum-redundancy codes. Proceedings of the Institute of Radio

Engineers, 40(9):1098–1101, September 1952

4 0x63 (’c’)

5

0x74 (’t’)

6

0x61 (’a’)

7

0 1

0x62 (’b’)

1

0x6d (’m’)

3

Repeat the process until there is only one node left in the list. The rest of the process for this file is shown in Figure 94.

Now that you have the tree, to encode the file all that is necessary is to do a pass over the tree to extract the encodings into a table, then re-read the input file and translate each input character into the appropriate sequence of bits. If the file ends with a partially filled byte, pad the final byte with zeros.

File Format

The output format for the encoded file consists of two parts. First, comes a header that contains the frequency information necessary to re-create the tree, then the bits of the encoded file.

The first four bytes of the header consists of an integer containing the number of characters in the frequency table. (Remember, not all files will contain all 256 possible bytes.) Following this comes the frequency table, in alphabetical order5 . Each element of the frequency table consists of a single byte that it the byte itself, followed by a four-byte integer that is the frequency.

|  |  |  |
| --- | --- | --- |
| . . . | cn | count of cn |
| . . . | 1 byte | (int) (4 bytes) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of chars  (int) (4 bytes) | c1  1 byte | count of c1  (int) (4 bytes) | c2  1 byte | count of c2  (int) (4 bytes) |

Encoding

To encode the file, build the tree, extract the encodings into a code table, then re-read the input generating the output as you go.

Decoding

To decode, read the header of the encoded file, regenerate the tree, then use the bits of the file to walk the tree to regenerate the file. Start at the root. For each zero bit, go left, for each one bit, go right. Because this is a prefixless code, you will know you have decoded a character when you reach a leaf. Output this character and start again at the root.

You know how many characters are in the output from the counts encoded in the frequency table.

Tricks and Tools

Coding Standards and Make

See the pages on coding standards and make on the cpe 357 class web page.

Pesky Details

Endianness

Because you are reading and writing binary data, the endianness of integers will be apparent. For this assignment you do not need to be concerned with this, but know that files encoded on a big-endian machine

5 This is not necessary for file compression, but it will make your output match the reference program’s output.

0x74 (’t’)

6

0x61 (’a’)

7 9

0 1

0x63 (’c’)

4 5

0 1

0x62 (’b’)

1

0x6d (’m’)

3

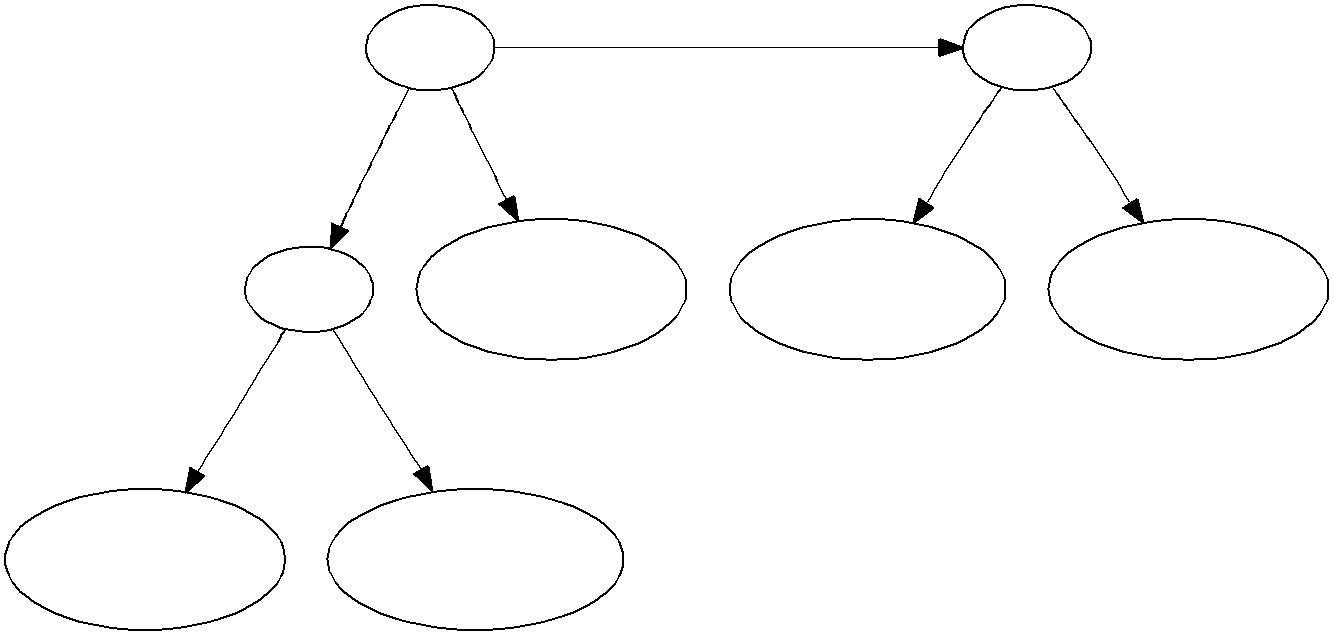
(a) After a second pass

9 13

0 1 0 1

4 0x63 (’c’)

5



0x74 (’t’)

6

0x61 (’a’)

7

0 1

0x62 (’b’)

1

0x6d (’m’)

3

(b) After a third pass

22

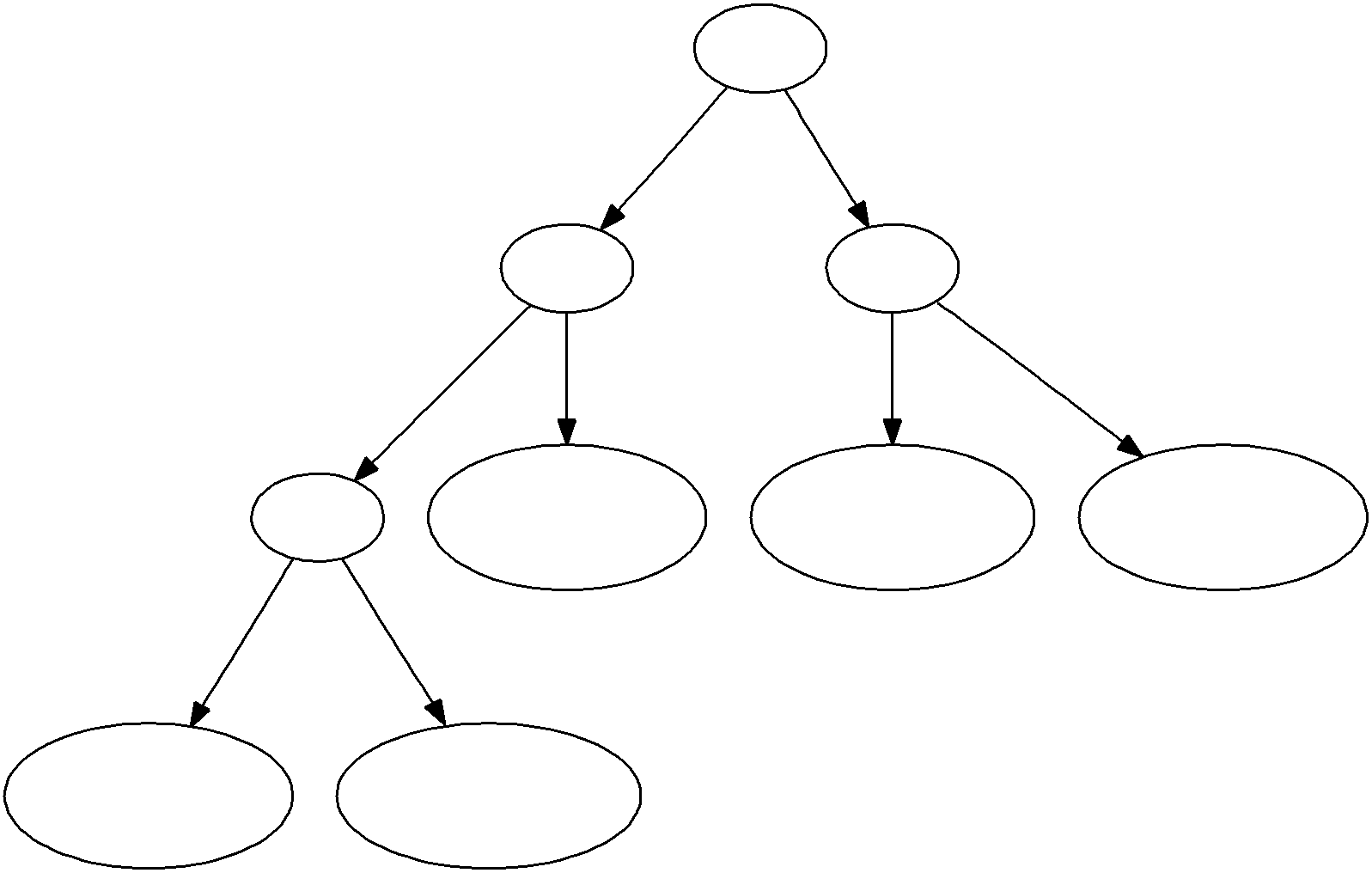
0 1

9 13

0 1 0 1

4 0x63 (’c’)

5



0x74 (’t’)

6

0x61 (’a’)

7

0 1

0x62 (’b’)

1

0x6d (’m’)

3

(c) The final tree

Figure 94: Finishing tree generation

|  |  |
| --- | --- |
| od(1) | Useful for reading binary files to see what’s what. |
| open(2)  close(2) read(2) write(2) lseek(2) | The Unix basic IO functions. Check out the man pages. |

Figure 95: Some potentially useful library functions

will not decode properly on a little-endian one (and vice-versa). Note that the x86 (vogon) is little-endian but if you’re developing on a PowerPC Mac, it’s big-endian.

Bits and their order

• If it is necessary to pad the final byte of the file, pad it with zero bits.

• When filling bits, fill from the high order bits. That is, if the final four bits of an encoded file are 1010, the final byte of the file will be 0xA0.

What to turn in

Submit via handin to the lab3 directory of

* grader-ph account (for professor Hatalsky)
* graderjw account (for professor Workman)

• your well-documented source files.

• A makefile (called Makefile) that will build your programs when given the command “make all”.

• A README file that contains:

– Your name.

– Any special instructions for running your program.

– Any other thing you want me to know while I am grading it.

The README file should be plain text, i.e, not a Word document, and should be named

“README”, all capitals with no extension.

Sample executables:

~phatalsk/www/357/demos/hencode

~phatalsk/www/357/demos/hdecode

**Sample runs**

phatalsk@vogon:~/www/357/demos $ time -p hencode /usr/share/dict/linux.words myWords

real 19.07

user 1.88

sys 16.74

phatalsk@vogon:~/www/357/demos $ ls -l

total 3185

-rwx--x--x 1 phatalsk Faculty\_Users 12184 2011-04-07 22:46 fw

-rw------- 1 phatalsk Faculty\_Users 9769 2011-04-27 11:51 fw\_compressed

-rwx--x--x 1 phatalsk Faculty\_Users 421088 2011-04-07 22:54 fw\_new

-rw------- 1 phatalsk Faculty\_Users 334002 2011-04-27 11:59 fw\_new\_compressed

-rw------- 1 phatalsk Faculty\_Users 4167 2011-04-18 12:55 hash.c

-rw------- 1 phatalsk Faculty\_Users 2801 2011-04-27 11:51 hash.c\_compressed

-rw------- 1 phatalsk Faculty\_Users 2764551 2011-04-27 11:55 myWords

phatalsk@vogon:~/www/357/demos $ time -p hdecode myWords linux.words

real 13.89

user 1.72

sys 11.61

phatalsk@vogon:~/www/357/demos $ ls -l

total 8369

-rwx--x--x 1 phatalsk Faculty\_Users 12184 2011-04-07 22:46 fw

-rw------- 1 phatalsk Faculty\_Users 9769 2011-04-27 11:51 fw\_compressed

-rwx--x--x 1 phatalsk Faculty\_Users 421088 2011-04-07 22:54 fw\_new

-rw------- 1 phatalsk Faculty\_Users 334002 2011-04-27 11:59 fw\_new\_compressed

-rw------- 1 phatalsk Faculty\_Users 4167 2011-04-18 12:55 hash.c

-rw------- 1 phatalsk Faculty\_Users 2801 2011-04-27 11:51 hash.c\_compressed

-rw------- 1 phatalsk Faculty\_Users 4953717 2011-04-27 12:01 linux.words

-rw------- 1 phatalsk Faculty\_Users 2764551 2011-04-27 11:55 myWords

phatalsk@vogon:~/www/357/demos $ diff linux.words /usr/share/dict/linux.words

phatalsk@vogon:~/www/357/demos $

Note the times it took to compress (hencode) and decompress (hdecode) the entire dictionary.

Also note the compression ratios (as a % reduction in size):

fw 20%

fw\_new 21%

hash.c 33%

linux.words 44%

As expected, text files seem to compress more than binary files. Why?