CS 241 Data Organization using C
Project 3: Huffman Coding

Fall 2014

Project Overview

For this assignment, you will write two programs to encode and decode files using Huffman Coding.

`huffencode`

The program in `huffencode.c` takes two command line arguments. The first is the name of the file to encode and the second is the name of the file where the encoded data will be written. Exit with an error if either of the files are unable to be opened.

In addition to writing the encoded file, print a table of byte frequencies and huffman codes to standard output. Separate columns with a tab characters. Visible ASCII characters (33-126) are printed as characters. Values outside that range are displayed as equals sign followed by integer value of the byte.

For example, if the file `small.txt` contains the text: `a small sample string` (with no newline at the end of the file), the command

```
./huffencode small.txt small.huff
```

will result in the following being printed to standard out.

```
Symbol  Freq    Code
=32     3       111
a       3       110
e       1       1010
g       1       0101
i       1       0100
l       3       100
m       2       001
n       1       0001
p       1       0000
r       1       10111
s       3       011
t       1       10110
Total chars = 21
```

`huffdecode`

The program in `huffdecode.c` takes two command line arguments. The first is the name of the file to decode and the second is the name of the file where the decoded data will be written. Exit with an error if either of the files are unable to be opened.

`huffman.c` and `huffman.h`

Obviously, there is a lot of common Huffman code functionality needed for both encoding and decoding. Place this in `huffman.c` and expose the necessary declarations in `huffman.h`. If you want to break down your code among additional files, feel free to do so, but make sure that you update the `Makefile` accordingly.

Huffman Tree Algorithm

- Count frequency of symbols in a stream.
- Create leaf nodes from symbol frequency data. (Don't include symbols with zero occurances.)
- Add all nodes to priority queue.
- While queue has more than one item:
  - Remove two trees from queue.
  - Create new tree with those two as children. (First removed should be left child.) Tree frequency count is sum of children's counts.
  - Add new tree to priority queue.
- Use final Huffman tree to generate codes.
- Huffman code for symbol at leaf is the path from root to leaf, with '0' for each left branch and '1' for each right branch.

Huffman Tree Node

- All nodes have a frequency count – `unsigned long`
- Leaf nodes have a symbol – `unsigned char`
- Internal nodes have references to two subtrees.
- You may want some additional bookkeeping fields, depending on your exact implementation. (isLeaf, parentNode, etc.)

Comparing Huffman Trees

The exact code generated depends on the order of the trees in the priority queue. For this project, we'll compare trees in the following manner.

- First, compare frequency values for the trees. The tree with the lower frequency has higher priority.
- If the two trees have the same frequency, compare the symbols in the leftmost leaf nodes of the two trees. Larger value has higher priority.

Encoded File Format

Binary files written on one system may not match file written by same program on another system, but a file encoded with `huffencode` should be able to be decoded with `huffdecode` if they are running on the same system.

- Number of symbols in table – `unsigned short`
- Symbols and Frequency pairs
  - Symbol – `unsigned char`
  - Frequency for symbol – `unsigned long`
- Total number of encoded symbols – `unsigned long`
- Bits of encoded data. Last byte may be padded with zeros if code string did not end on byte boundary.

  Bit 0 of the encoded stream should be in bit 0 of the first byte.

data	a	*space*	s	m	a	l	l	*space*	...
codes	110	111	011	001	110	100	100	111	...
code stream	110111011001110100100111...								
bytes	10111011 10111001 11100100 ...								
hex	BB B9 E4 ...								

Turning in your assignment

Make a single zip file containing `huffencode.c`, `huffdecode.c`, `huffman.h`, `huffman.c`, and `Makefile` and submit it to the Huffman Coding assignment in UNM Learn. If you divided your program into additional files, include them all in the zip file and make sure that your makefile is updated to compile them.

Grading Rubric: 100 points

-5 point : The programs do not start with a comment stating the students first and last name and/or the source files are not named correctly.

-5 points: Programs compile with warnings on moons.cs.unm.edu using `/usr/bin/gcc` with no options (using the makefile)

-5 points : Programs leak memory (tested with valgrind)

-5 points : Using non-standard types such as `int128`. If your data doesn't fit in a `long long`, you'll have to figure out some other way of dealing with it.

15 points: Follows CS-241 Coding Standard: including quality, quantity and neatness of comments, no dead code, and Best Practices (functions not being too long, nestings not needlessly deep, and avoidance of duplicate code).

40 points: Frequency and code table printed out by `huffencode` matches expected output for each of `mississippi.txt`, `sense.html`, `ugly.txt`, `MobyDick.txt`, and `ralph.bmp`. (8 points each)

20 points: Encoded file produced by `huffencode` passes binary diff with encoded versions of above files. (4 points each)

20 points: Decoded file produced by `huffdecode` with encoded versions of above files matches binary diff with original files. (4 points each)

5 points: Decoded file produced by `huffdecode` with `evil.jpg.huff` matches binary diff with original file.