Assignment 6
Huffman Coding

DESIGN DOCUMENT

Zack Traczyk
CSE13S - Spring 2021

Due: May 23th at 11:59 pm

1 Objective

Implement static lossless file compression using Huffman encoding.

2 Given

- Header files for definitions, header, node, priority queue, code, io, stack, Huffman
- C file for entropy

3 Data Structures

This assignment required a couple implementing three different datatypes to work: nodes, a stack, and a priority queue. The implementation for these abstract datatypes will be described in detail below.

3.1 Nodes

Nodes are implemented in `node.c` and follow the assignment document specifications. As given in the document, a node is a structure that holds pointers to its left and right child, an 8-bit unsigned integer for its symbol, and a 64-bit unsigned integer for its frequency. Nodes are used to construct a Huffman tree which is required for encoding.

3.2 Stack

Stacks are implemented in `stack.c`. A stack is a structure that contains an array of pointers to Nodes (the elements the stack holds), an unsigned 32-bit integer to the next empty value in the stack, and an unsigned 32-bit integer that stores the total capacity of the queue.

3.3 Priority Queue

Priority queues are implemented in `pq.c` using a min heap. A priority queue is a structure that contains an array of pointers to Nodes (this is where elements that are enqueued and dequeued are stored), an unsigned 32-bit integer to the next empty value in the stack, and an unsigned 32-bit integer that stores the total capacity of the queue.

4 Programs

Before creating the two required programs, encode and decode, some more helpful functions needed to be implemented to help with io functionality and to assist with the encoding and decoding.

4.1 IO

Two global variables are defined in `io.c`: `bytes read` and `bytes written`. These variables track how many times bytes have been read and written to/ from a file. Additionally, there are four main io functions: read bytes, write bytes, read bit, write codes.

4.2 Encode

- -h : Command line options
- -i `infile` : The file containing bytes to be encoded (default is `stdin`)
- -o `outfile` : The output file to store encoded bytes (default is `stdout`)
- -v : Prints compression statists of the encoding process to `stderr`

4.3 Decode

- -h : Command line options
- -i `infile` : The file containing bytes to be decoded (default is `stdin`)
- -o `outfile` : The output file to store decoded bytes (default is `stdout`)
- -v : Prints statists of the decoding process to `stderr`

4.4 Entropy

The source code for this program (`entropy.c`) is provided in the class resources repository. Entropy calculates the entropy, or variation of contents, for a file.

5 Parse

Program arguments are parsed and stored using flags. A set is not used since the order of the inputs is irrelevant and the parsed data will ultimately be stored in variables anyway.

After program arguments are parsed the input file is parsed with encode or decode. Execution happens with every grab of a byte to the buffer. Refer to execute for how the algorithm works.

6 Execute

6.1 Encode

Compressing a file requires a few steps. First, a histogram is constructed of the most used characters. Once each character has a frequency, the histogram elements are converted to nodes and enqueued into a priority queue. Then, two nodes are popped off at at time and joined together and queued back into the priority queue until only one node remains. This last node is the root to the Huffman tree. Finally the Huffman coding happens as the tree is traversed and codes are assigned to all the characters.*alg.* (1).

Algorithm 1: Encode

6.2 Decode

Decode is (unsurprisingly) the opposite of encode. Codes are read from the file and decoded. The pseduocode is shown in *alg.* (2).

Algorithm 2: Decode