$Id: asg3-smalltalk-hzip.mm,v 1.28 2019-01-29 18:24:30-08 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps112-wm/Assignments/asg3-smalltalk-hzip
URL: http://www2.ucsc.edu/courses/cmps112-wm/:/Assignments/asg3-smalltalk-hzip/

1. Overview

Smalltalk is a pure object-oriented language, where everything is an object, and all actions are accomplished by sending messages, even for what in other languages would be control structures. In this project we will be using Gnu Smalltalk, the Smalltalk for those who can type. References:

`http://smalltalk.gnu.org/`
`http://smalltalk.gnu.org/documentation`
`https://www.gnu.org/software/smalltalk/manual-base`
`ftp://ftp.gnu.org/gnu/smalltalk/smalltalk-3.2.5.tar.gz`

2. A file compression utility

We present the program specifications in the form of a Unix `man`(1) page. This program will use Huffman coding to compress and decompress files.

NAME

hzip.st — file compression and decompression utility

SYNOPSIS

`hzip.st -dtcu` *inputfile* [*outputfile*]

DESCRIPTION

A file is either compressed (with the `-c` option), uncompressed (with the `-u` option), one of which is required, unless the `-t` option is specified. The input filename is required. If the output filename is not specified, output is written to the standard output.

OPTIONS

All options must be specified as the first word following the name of the command, unlike the standard `getopt`(3) used for C programs. Exactly one of the options `-t`, `-c`, or `-u` is required.

`-d` Debug information is printed for the benefit of the application author. Exact details are not specified.

`-t` The compression algorithm is activated, and the decoding tree is printed to the standard output. The output filename may not be specified.

`-c` The input file is compressed and written to the output file, if specified, or to the standard output, if not.

`-u` The input file is assumed to be compressed, and is uncompressed, written to the output file, if specified, or to the standard output, if not.

OPERANDS

There are one or two filename operands. The first is the input filename and the second is the output filename. If no output filename is specified, the standard output is written.

EXIT STATUS

0	No errors occurred.
1	Errors occurred, either in scanning options or opening files.

3. Compression

The compression algorithm reads the input file twice, once to construct the decoding tree, and if the `-c` rather than the `-t` option is specified, a second time to perform the compression. It proceeds as follows:

(a) Read in the input file and create a frequency table, counting the number of times each character appears on input. The frequency table is indexed from 0 to 255 with characters. Add entry 256 with a count of 1 to indicate EOF.

(b) Iterate over the frequency table, and for each non-zero element, create a leaf node and insert that leaf node into a priority queue, with the character and the count. In Smalltalk, use a `SortedCollection`. The counts take precedence, but if two entries have the same count, the one with the smaller character (lexicographically) is considered smaller.

(c) Repeatedly remove the two smallest elements from the priority queue, creating a new tree which is then entered into the priority queue. The smaller tree or leaf removed becomes the left child, and the larger the right child. The charcter in the new tree is the left child's character. This process stops when there is only one tree left and the priority queue is empty.

(d) For each character that has appeared as non-zero in the frequency table, construct an encoding string, using a depth-first traversal. The encoding string is a sequence of bits indicating the path from the root to a leaf.

(e) If the `-t` option is specified, write out the encoding table sorted by character. The first column is a single character, if printable, or an integer if not. The second column is the frequency for that character. The third column is a sequence of 0 and 1 characters indicating the encoding. Format should appear as if done by one of the following format items. In the table, the symbol "␣" represents a space character.

format item	example
`"x%02X␣%5d␣%s"`	`x0A␣␣␣␣10␣11001`
`"␣%c␣%5d␣%s"`	`␣a␣␣␣␣␣20␣1101`
`"EOF␣%5d␣%s"`	`EOF␣␣␣␣␣1␣1100000`

(f) If the `-t` option is not specified, write out the encoding table as follows: Perform a post-order traversal of the decoding tree, writing out one bit at a time in big-endian format. for each leaf, write out a 0 bit, followed by the 8 bits of the corresponding byte. Write out the bits in the order bit 7, bit 6, . . ., bit 0, that is high bit first. As a special case, if the byte is 0, write out bit 8, which will be a 0 for a byte value of 0, and 1 for a byte value of 256 (the EOF marker).

(g) For each interior node, write out a 1 bit. When the tree is completely written out, write another 1 bit to indicate the end of the tree.

(h) Reopen the input file and write out the encoded version of each byte. At end of the buffer, write out the encoding string for EOF. Then pad with 0 bits out to a byte boundary, if needed.

4. Decompression

To uncompress a file, consider that the compressed file has a decoding tree followed by data. It is possible that the compressed version is large than the uncompressed version. Proceed as follows:

(a) Reconstruct the Huffman decoding tree. Read one bit.

(b) If it is a 0, read the next 8 bits and reconstruct the byte, giving a value from 0 to 255. If the value is 0, read one more bit, and if it is a 1, add 256 to the byte, giving the encoding for EOF. Then push this byte onto a stack.

(c) If it is a 1, pop the 1-subtree from the stack, then pop the 0-subtree from the stack, and create a new tree with both of those children and push the new tree back on the stack.

(d) The last extra 1 bit will be read when there is only one tree on the stack. Popping this tree will cause the stack to be empty, so this is the decoding tree.

(e) Now loop over the rest of the input file to reconstruct the original file: Initialize a pointer to the root of the decoding tree.

(f) Read a single bit and use it to move down the 0-link or the 1-link to the child of the current node.

(g) If thise node is a leaf, write out the corresponding byte and reset the pointer back to the root of the tree.

(h) If not, continue reading bits until you find a leaf.

5. Efficiency of the encoding.

The encoded tree contains only instances of those characters that actually occur in the file, and so is smaller than the complete tree. In the worst case, there are 257 different characters to write to the file, 255 of which are encoded as leaf nodes using 9 bits, the remaining two (0 and 256) being encoded using 10 bits. So the leaf nodes in the tree will consist of at most $255 \times 9 + 2 \times 10 = 2315$ bits. In a complete binary tree with n nodes there are $n - 1$ interior nodes, and since each of these are represented by 1 bit, there are at most 256 bits representing interior nodes, with a total of $2315 + 256 = 2571$ bits as the worst case for the entire tree. 2571 bits = 321.375 bytes.

Actual Huffman encoding will compress a file unless each byte in the file occurs exactly the same number of bytes in the file, which means that in the worst case, the file will increase in size by 322 bytes. Some files with poor statistical properties will increase a little, but most will in fact be compressed significantly. *The Complete Works of William Shakespeare* is 5,447,534 bytes long. The **`compress`** program, a patented algorithm, can compress this to 2,214,693 bytes. The **`gzip`** program, a free algorithm, is more efficient a compresses it to 2,015,136 bytes. How many bytes are taken up when the present Huffman algorithm is used?

6. What to submit

Submit a **README** and **hzip.st** with a hash-bang and marked as executable. Submit a **PARTNER** file if you are doing pair programming. See the rules for pair programming.