SMU Guildhall Programming Assignment 2

Yu Lu

The program huffman_compress is a small console based program which compress a file **filename** using Huffman Compression algorithm, which is always guaranteed to be efficient. This algorithm turn bytes into a variable length of bits, the more frequent ones being shorter, and writing this compressed bit into a binary. On compression, it creates **filename_comp** and the associated info file **filename_info**, and will generate **filename_deco** upon decompression, which will be identical to the original file byte by byte. Note that **_deco**is inserted into the after the file name whereas **_comp** and **_info** is appended at the end.

The program requires at least one argument, being the **filename**, and multiple files may be worked on on a single call. More over, there are three arguments

Argument	Description	Effect
-s / --skip	Skip generating the compression files, which is under the assumption it's already built	Skip the compression phase
-v / --verify	Verify the decompressed file **filename_deco** with the original, making sure they're identical	Verify the decompressed content
-g / --generate	Keep the decompressed file **filename_deco**	Keep the decompressed file after program closes
filename1 filename2 ...	Load **filename** and compress the files	Change Input Files

The compression phase is only initiated when –skip is not toggled, and the decompression phase is only initiated when –verify or –generate is toggled. More over, on each phase, 4 errors can arise. Note that any one of them can terminate the program.

Error Message	Cause
File **filename** cannot be opened	Something wrong with the source file which render it unreadable, or is missing entirely.
Compression file **filename_comp** cannot be opened.	Something wrong with the compression file which render it unreadable, or is missing entirely. Only relevant is -skip is toggled.
Info file **filename_comp** cannot be opened.	Something wrong with the compression file which render it unreadable, or is missing entirely. Only relevant is --skip is toggled.
Provided **filename_comp** is faulty.	Only relavant when --verify is toggled.

The coding was done in about three days, uses vector, stack and unique pointer for the entire system and thus without any memory management. Note that the system is expressed in terms of graphs and allocated as unique pointers, and are completely separate. The program's split into four modules:

byte.cc/.h is the linked nodes with reference to its children and parents. This is occasionally revisited but largely remain the same. Takes about a total of four hours. Note that there are no unique pointer on this level, only inside storage.

storage.cc/.h is the compression module which is responsible for loading all the bytes and then building the compression tree file and the compressed binary. The tree file has two lines, the first referencing the effective size of the compressed binary due to the compressed file always generates two additional bytes for unknown reasons, and the fact additional bits are being loaded to complete a byte. About a day and a half is spent on this module.

tree.cc/storage.h is the decompression module which is responsible for turning the bit sequence back into bytes. It loads the tree file to rebuild the tree which the byte is converted into bit sequence. It took about a day to complete.

main.cc is the main and I/O and module, and is responsible for managing both the compression and decompression modules. It is built alongside the two modules, and an additional day is used to add program arguments to enhance the program, add multi-threading, and toying with bitwise operators.

A major problem initially was due to memory associated greed which set the limit of variables too small, which took half a day to handle.
Due to the nodes being stored inside the vector of unique pointers but consistently referenced as raw pointers, there is no need to worry about freeing any resources. More over, due to the additionally files directly appending the target file, it will create all associated files at the target folder.