Final Report - File Compressor

John Ellmore
December 2, 2013
CS374 Software Engineering

Summary

The `squash` compressor is a fully-functional file compressor able to compress text files, binary files, and directories into a single file. It supports recursively compressing nested directories. It can compress data using the LZW compression algorithm.

I was originally working on this project with Fraser, but he dropped the course before writing any working code. None of his code was used in the final product.

Requirements

POSIX system (Mac, Linux, etc.) with bash and GNU g++ compiler. The version of g++ used must support C++11.

Use Cases

Found in “assignments/Use Cases.pdf”.

Interaction Diagrams

I think I missed this day, and I couldn’t find any information about it on Canvas. So I don’t have one.

Class Diagrams

Found in “assignments/Class Diagram.pdf”. The final design was slightly modified from this document.

Test Cases / Harness

In the `output` subfolder of the repository, there are two shell scripts: `test.sh` and `cleantest.sh`. When `squash` is installed and compiled, running the `test.sh` script will put the program through its paces with 12 differerent test cases. This will test the ability of LZW to compress different types of strings, including simple strings, strings with null bytes, long complicated text documents, and binary files (specifically images). It will also test the compressor’s ability to combine multiple files and folders into one file, including nested folders. The generated files will be placed in the same directory as the `test.sh` script and are available for inspection. When the `test.sh` script is finished, the `cleantest.sh` script may be used to reset the output folder back to a clean state.

Compression Algorithms Implemented

The LZW algorithm was implemented, using variable-length encoding to achieve better compression ratios. On the long text documents included in the test files (Declaration of

Independence, the Constitution, and Hammurabi's Code), compression ratios can be as good as 50% size reduction.

How to Install

Navigate to the repository directory and run the `build.sh` script. This will generate the `squash` executable. You're ready to go!

How to Compress Files

```
./squash [-a ALGORITHM] [-o OUTPUT_FILE] INPUT_FILE [...]
```

Options (can be in any order):

- `-a ALGORITHM` allows the compression algorithm to be specified. The only supported options are `lzw` and `none`.
- `-o OUTPUT_FILE` allows the user to specify a custom output file location. The default is `compressed.cmp`
- One or more input files may be specified. All of the input files will be pulled into the archive.

How to Decompress files

```
./squash -e INPUT_FILE
```

Options (can be in any order):

- `-e` specifies that the archive be expanded, not compressed.
- One input file may be specified, and it must be a valid archive file. The contents of the file will be extracted into the current working directory.

Known Issues

- It should be possible to add two input files with the same name to the compressed file with no problems. However, the behavior when extracting this is undefined and may not cooperate (related: a real-world hack exploiting this flaw in Google's Android APK archive format).
- Large files (over the memory capacity of the host computer) will cause the compressor to run out of memory, as files are read entirely into memory before compressed. I could fix this with a bit of work, but the current way it works uses less disk seeks and is thus faster for smaller files.
- When expanding an archive, the output directory is always the current working directory. This can't be modified by the user. I could have implemented this, but it was not a very high priority compared to getting LZW working.

Reflections / Lessons Learned

- I had a number of small bugs with my implementation of LZW that only appeared on sufficiently large inputs. I probably wasted three hours trying to fruitlessly troubleshoot them without deeply tracing the code. I finally made progress when I heavily modified the algorithm using 30 extra lines of debug code, which took awhile but allowed me to fix the real problem in 30 minutes. **Lesson learned:** Be willing to rework complex code for debugging purposes instead of just relying on occasional "`cerr <<`" statements; it'll save time in the long run.

- I found that my repository file structure became complicated and crowded very quickly. It resulted in me often mixing up files and having to think hard for a second what I was trying to do. This was fairly annoying until I finally cleaned it out in the final few commits. **Lesson learned:** Keep a clean file structure with clear naming conventions, and make copious use of .gitignore files and subfolders.