UMass Boston CS 444
Project 1
Posted Tuesday, February 4, 2025
Due Monday, February 17, 2025 at 11:59 pm

1 Project Description

Write a C program that, given a text file, constructs Huffman code based on the character frequencies of the file and then encodes the file accordingly. It must be compiled on the CS server. The executable must accept the following command-line options and default files names:

- `-i filename` for the input file — default is `completeShakespeare.txt`
- `-o filename` for the output file — default is `huffman.out`

The instructor's executable is `/home/hdeblois/cs444/proj1/huffman` on the CS Linux servers. The test file `completeShakespeare.txt` and the `Makefile` are also in the `proj1` directory. You need to make sure your code works perfectly with it. You can test your implementation with the following commands.

```
./yourExecutable -i completeShakespeare.txt -o myExec.out
./instructorsExecutable -i completeShakespeare -o tmp.out
diff myExec.out tmp.out
```

If your code works correctly, `diff` will find no difference between the two output files.

2 Project Details

In the `cs444` folder under your `courses` directory on the CS server, create a folder called `proj1`, and put all your files in there, including C code, a `Makefile`, an executable and a `readMe.txt`. Spell the names carefully and exactly as given because we will use a script to collect your work.

For your C code, choose any `<name>.c` you like. Change the `Makefile` so you can compile your code by typing: `make`. The `<name>` of your code must match the name you include in the `Makefile`. Type: `man make` to see information about GNU Make Utility or see its entry on the instructor's website.

Write your C code on the server using a text editor, e.g., `nano` or `emacs`. Compile and run on server each time you work on it. Put your name in the code in a comment at the top. If, in addition, you compile your code on your laptop, be aware that the C libraries on the server may not be the same as on your laptop so you must recompile on the server.

For command-line options, you may want to include the `getopt.h` library and use it to implement the options. As variables, include a debug switch `-d` to set `int debug = 1` to turn on `printf` statements. You could use multiple debug options. When debug is not set, your code should output only the huffman-coded version of the input file. Follow the four-step implementation described on slide 27 of the `huffman.pdf` slides:

1. Read the input file and use a 256 element array to count the frequency of each character. Turn debug on to print out frequencies.

2. First, use the frequencies to build a forest of tree nodes for characters that have non-zero counts. Second, sort by frequency and build the huffman tree by combining the two nodes with lowest frequency into a new node with those nodes as children, resorting and repeating. Turn debug on to print out the total combined frequency, which will match the size of the input file.

3. Traverse the tree to collect the code for each leaf by length and bit sequence. Turn debug on to print out each character with its code.

4. Read the file again, code each character, put the bits in a buffer keeping track of how many and output a byte using `fwrite` when the buffer has eight bits, saving the rest for the next byte if necessary.

You can run the instructor-supplied executable with various command line options. Set debug=1 to see the huffman codes.
Prepare a `readMe.txt` to list how you developed your code in increments. For each time you work, make an entry in your readMe.txt file with date and what you worked on. Keep incremental copies of your code on the server. For example, each day copy your code to ¡name¿.c-Feb4. Code will be collected from the server often. Enter your full name at the start of the readMe.txt file and in a comment in your code since some user names bear no resemblance to real names.
If you consult and/or use (small features of) code on the internet, list each source in your readMe.txt and cite the source in your code according to the MIT Writing Code formats. This applies to code you adapt as well. Do not cite fellow student's code because it is not a published source and you are not supposed to share your code. You do not have to cite any code given you by the instructor. In grading, we run Stanford's Measure of Software Similarity (MOSS) to check that no one duplicates someone else's code, but if a too high level of duplication occurs, both students will be penalized.

3 Grading Rubric

Be sure to "make your code your own," by commenting it and by knowing exactly how it works. Do not use AI generated code in your code. Also, the instructor and graders may require you to come in and explain how particular sections of your code work. You will find that knowing the algorithm before you start coding and reviewing its steps carefully allows you code your own way.

The first step in grading is to run your executable in your course directory and read your `readMe.txt`. After we check that your executable runs, we copy your code to another location, recompile it and test it again, and then run MOSS on all pairs of code.

Be sure to notice how the uppercase and lowercase letters are used in naming the files and directories that you are required to create (`Makefile`, `proj1`, `readMe.txt`). Names must match exactly or your submission will not be collected.

Any file changes made after 11:59 pm on the due date will not be taken into account in grading. Any request for an extension must be received before the due date/time and will only be considered if substantial (compiled/run)

work has been done along the way and is on the server. Extensions may result in taking some points off your grade. Code submitted late will lose 15 points.

- (20 points) `readMe.txt` contains full name, list of what was worked on in each increment and list of sources consulted if any.
- (20 points) all combinations of command-line arguments and defaults work
- (10 points) code is indented and names well-chosen, and citing is specific
- (50 points) correct output for `completeShakespeare.txt`