CSE3081-2: Design and Analysis of Algorithms (Fall 2019)
Machine Problem 3: Huffman Coding for File Compression

Handed out: November 14, Due: December 5, 11:59PM (KST)

1. Goal

The goal of this MP is to design and implement a Huffman Coding-based file compression utility program.

2. Problem Description

File compression is important when you want to save disk space or send files through the network. You want to write a utility program which you can use to compress a file and recover the original file from a compressed file.

For compression, you are going to use Huffman Coding, which is a greedy algorithm. The goal of the algorithm is to assign long bit strings to rare characters and assign short bit strings to characters that appear frequently. Refer to the lecture slides for details.

3. Your task and requirements (Read Carefully!)

(1) You will write a single C/C++ program which takes an input file and produces an output file. Your file should have two functions: compress and decompress. The user command indicates whether we are compressing (-c) or decompressing (-d).

(2) When you compress, your output file will have ".zz" appended at the end of the input file name. If the user runs the program as follows:

```
$ mp3_20180001 -c input.txt
```

Then, your output file should be "input.txt.zz".

(3) When you decompress, your output file will have ".yy" appended at the end of the input file name. If the user runs the program as follows:

```
$ mp3_20180001 -d input.txt.zz
```

Then, your output file should be "input.txt.zz.yy".

(4) If the user gives options other than "-c" or "-d", the program should print an error message and stop. Also, if the input file does not exist, the program should print an error message and stop.

(5) In this MP, you do not need to worry about whether the user will try to decompress a non-compressed file. The TA will compress the file using your program and decompress the output of the compression using your program.

For example, the TA will run commands in (2) and (3), and compare "input.txt" and "input.txt.zz.yy". Obviously, they should be exactly the same.

(6) When you decompress, you should NOT use any other input file except the compressed file itself. In other words, you should not create a separate file which will help you in decompression.

(7) The input files will be text files. Specifically, you can safely assume that all characters used in the file are from the ASCII code table. (http://www.asciitable.com/) No Korean (or non-English characters) will be used. However, we will test your program with files that have different distribution of characters (e.g. A file that mostly consists of numbers, and a file that mostly consists of alphabets.)

(8) The format of the compressed file is up to you. You may embed any additional information necessary for decompression in the compressed file. However, watch out for the size of the compressed file. That is the performance of your algorithm.

(9) When evaluating your work, we are more interested in the size of the compressed file. For performance in terms of time, you will not be deducted points unless your program takes unreasonably large amount of time for compressing or decompressing a file.

(10) Use Huffman Coding! Do not use other compression algorithms you find on the Internet. A slight variation will be allowed. For example, instead of encoding a byte (character) into a bit string, you can encode two bytes (characters) into a bit string.

(11) Similar to mp1 and mp2, your code should build and run on a **Linux machine**. So make sure you test on the machine before you submit the files.

(12) You should write a **Makefile** this time too. The TA will build your code by running 'make'. It should create the necessary binary file.

(13) Your binary file should be named **mp3_20180001**. The red part should be your student ID. There should be only a single binary file. It is up to you to make a single or multiple source code files.

4. Report

In addition to code files, you should also submit a report document.

In the report, you should include at least the following two things:

(1) A description of how you implemented your algorithm. For example, what kind of data structure you used, the format of the compressed file, any implementation issues or design choices you made.

(2) A table showing example run of your program on several different types files. You should include sizes of the original file and compressed file, and compression ratio.

compression ratio = original file size / compressed file size

Is the compression ratio different for different types of files? Write any comments on your results.

5. Submission

You should submit the Makefile, the source code(s), and the report document. Make the file into a zip file named cse3081_mp3_**20180001**.zip. The red numbers should be your student ID. You can submit your work on the cyber campus.

6. Evaluation Criteria

(1) Correctness of your implementation: 30%

- Does your implementation produce correct results? When we compress a file and then decompress the compressed file, do we get the original file back?

※ The maximum size of the input file will not exceed 100 megabytes.

(2) Performance of your program: 30%

If your program produces wrong result, you will get 0 points here. Otherwise, the points will be given based on the size of the compressed file (after comparing performance with other students.) In addition, if your program takes unreasonably long time, that will also be counted towards the performance evaluation.

※ The input files will include multiple files with different sizes.

(3) Comments and coding style: 10%

- Is the code organized and easy to read? (indentation, spaces, styles)
- Do variable names reflect what they really are?
- Are there enough comments that explains what the code blocks are doing?

(4) Your report: 30%

- Can the reader understand how you implemented the program? Imagine you write the compress function and another person writes the decompress function. The other person should be able to write the decompress function after reading your report.

7. Notes

- You should write your own code. You can discuss ideas with other students, but definitely should not copy their work. For this particular MP, since you can see the example code on the lecture slides, you will lean towards following the code exactly. My advice is that you just catch the idea, and start writing your own code without looking at the example code.

- As announced in the first class, duplicates will receive zero grade.

- 10% of the evaluation goes to the practice of writing the code. Your code should not just work, but should also look good to other programmers. There are many books and materials on code writing practice. One example is a book called “Code Complete” by Steve McConnell.

- You may write your program in your own environment (Windows, Linux, MAC). But you should test your code on the Linux machine before submitting the file. Each of you will be given an account to the department Linux server.

- Remember that the TA will place your files in a directory, build your code using ‘make’, and run the code with the test inputs. Make sure everything works before submitting your work.

- Do NOT submit binary files. Submit only the files listed in the Submission section.