Algorithm Design, Anal. & Imp., Programming Assignment

Out: Feb 26, 2013, Due: March 20, 2013

Note:

- This assignment will carry 10 points towards your final score.
- You can use any of the following languages (C++, Java). If you want to use any other programming language, please consult with me beforehand.
- You can complete the assignment in a group of two students. If you want to work individually, you are welcome to do so.
- You need to submit source code, which should run on a standard Linux machine. Your output should match exactly as outlined in this document.

Huffman Coding based Compression Library

Background

Huffman code is used to compress data file, where the data is represented as a sequence of characters. Huffman's greedy algorithm uses a table giving how often each character occurs; it then uses this table to build up an optimal way of representing each character as a binary string. We call the binary string the codeword for that character. A property of Huffman code is that it is a prefix code, i.e., in Huffman coding, no codeword is a prefix of some other codeword. The advantage of prefix code is that it makes decoding easier, as we do not need to use delimiter between two successive codewords. Given the frequency of each of the character, we can devise a greedy algorithm for finding the optimal Huffman codeword of each of the characters. For details of the greedy algorithm, please read Section 16.3 of the textbook.

In this assignment, we will build a compression library that compress text files using Huffman coding scheme. This library will have two programs: `compress`, and `decompress`; `compress` accepts a text file and produces a compressed representation of that text file; `decompress` accepts a file that was compressed with the `compress` program, and recovers the original file.

Implementation

Input to the `compress` is a text file with arbitrary size, but for this assignment we will assume that the data structure of the file fits in the main memory of a computer. Output of the program is a compressed representation of the original file. You will have to save the codetable in the header of the compressed file, so that you can use the codetable for decompressing the compressed file. Input to the `decompress` is a compressed file, from which the program recovers the original file. For sanity

check, you should have a specific magic word at some position in the header of the compressed file, so that `decompress` can identify whether the given file is a valid Huffman compressed file. You should pay attention to the following issues:

- The file that we will use for testing can be very large, having size in Gigabytes, so make sure that your program is bug-free and it works for large input file.
- Write efficient algorithm, we will take off as much as 20 points if we feel that the program is taking unusually long time.
- You must make sure that your program runs on a Linux Machine (such as, in pegasus.cs.iupui.edu), and identically follows the formatting instructions. For formatting error, as much as 15 points can be taken off.
- You must provide a Makefile to compile your programs. Also, a README.txt file should be provided that will have the instruction to compile and run the programs.

Command-Line options

Compression:

```
C++: ./compress -f myfile.txt [-o myfile.hzip -s]
Java: sh compress.sh -f myfile.txt [-o myfile.hzip -s]
```

Decompression:

```
C++: ./decompress -f myfile.hzip[-r -v]
Java: sh decompress.sh -f myfile.hzip [-o myfile.txt -s]
```

The command-line options that are within the square bracket are optional. The option "-f" precedes the input file name, which always has a `.txt` extension. The "-s" option prints statistics, such as for compression it prints, how many distinct characters are there, what is the compression ratio, and the wall clock time that it took for performing the compression task. For decompression, it prints how many character were written, and the wall clock time it took for performing the decompression task. The "-o" option precedes the name of an output file. If the output file name is not given, then we will append `.hzip` at the end of the input filename to create the output filename.

Deliverables

Please submit the source files in a tarzipped folder through on-course. The folder should be named as *lastname1_lastname2* and the submission file should be named as *lastname1_lastname2.tgz*. The folder should have a README with instructions to compile and run the program. Also, for the case of group submission, please highlight the contribution of each member of the team in the README file. Program will be graded by automated scripts, so it is important that you follow the submission guideline carefully. You will lose 10% of your score if we cannot process your submission in our automated script.