Data Structure and Algorithm II

Homework #4-2

Problem 1 Due: 10am, Monday, May 30, 2011

Problem 2 Due: 10am, Monday, May 23, 2011

=== Homework submission instructions ===

- See the submission method and requirements for the programming assignment in the description of problem 1.
- Submit the answers for writing problems through the CEIBA system (electronic copy) or to the TA in R204 (hard copy). Please write down your name and school ID in the header of your documents.
- Each student may only choose to submit the homework in one way; either all as hard copies or all through CEIBA except the programming assignment. If you submit your homework partially in one way and partially in the other way, you might only get the score of the part submitted as hard copies or the part submitted through CEIBA (the part that the TA chooses).
- If you choose to submit the answers of the writing problems through CEIBA, please combine the answers of all writing problems into only one file in the doc/docx or pdf format, with the file name in the format of "hw4-2_[student ID].{pdf,docx,doc}" (e.g. "hw4-2_b98902010.pdf"); otherwise, you might only get the score of one of the files (the one that the TA chooses).
- For each problem, please list your references (they can be the names of the classmates you discussed the problem with, the URL of the information you found on the Internet, or the names of the books you read). The TA can deduct up to 100% of the score assigned to the problems where you don't list your references.

Problem 1. **Huffman code file compressor/decompressor**: In the class, we introduced **Huffman codes**, which can be generated by a greedy algorithm. Huffman coding is actually often used for lossless data compression. In this assignment, we ask you to

implement a set of two programs, one file compressor and one file decompressor, utilizing the Huffman coding scheme. This assignment will be completed by teams of 2-3 students. As the deadline is tight, this is also a good opportunity for you to learn how to work in a team, how to split a big task into smaller ones, and how each team member can work on different parts of a program simultaneously in order to save time, utilizing a version control system.

The file compressor is executed with the following command:

```
./hc_compress <the name of the file to be compressed>
```

Your file compressor then builds a Huffman code based on the content of the input file, and utilizes that code to compress the input file and write to an output file. **You can assume that each symbol, which will be mapped to a codeword, is a 4-byte binary string in the file**. You program will then examine the input file and determine the frequencies of each symbol in order to create the Huffman code. Note that you have to include the mapping between the code and the symbol it represents in the output file so that your decompressor knows how to decompress the file. You need to design this mapping to be represented in a efficient way in the output file so that the size of it is minimized (obviously, as the goal is to minimize the size of the output file). Your file compressor should print the compression ratio (the size of the output file divided by the size of the input file) at the end of execution,

```
<the name of the input file>.hcc: <compression ratio>%
```

, and writes the output file with the original file name with a postfix ".hcc".

There are cases where using a Huffman code cannot compress the given input file (especially when the representation of the codes in the output file is inefficient), i.e., the compression ratio is larger than 1. If that is the case, your compressor should output

```
<the name of the input file>.hcc: cannot compress.
```

and exit without creating the output file.

Your file decompressor is executed with the following command:

```
./hc_decompress <the name of the original input file>.hcc
```

Your file decompressor then restore the compressed file to its original form, writing to the file with the original filename (without the .hcc postfix). Of course, please make sure that this file is identical to the original input file. And please note that the input file could be **several GB in size**. Note that your file decompressor should be able to tell when the specified file is not valid (not compressed by your file compressor) and in that case should output

```
Not a valid compressed file. Exiting.
```

and exit.
Please read the following requirements carefully as many requirements are different from those of other programming assignments.

1. Please form teams of 2-3 students. Each team only needs to develop and submit one set of compressor and decompressor. Teams with 3 students will have to complete additional features, which will be specified below. If you cannot find suitable team member, **please send us an e-mail as soon as possible and we will try to find one for you**.

2. Each team must utilize Subversion system we setup (the TA will send out the account/password. **Please send an e-mail to dsa2@csie.ntu.edu.tw to give us a list of your team members as soon as you form your team.**). As each team member will have his/her own Subversion account, we will know the contribution of each team member by looking at all commits, i.e., you use your own account to commit the part you develop, and the grade you get for this assignment will be largely based on the parts you commit. Write in the messages for the commits about what you did for these commits so that we know your contribution (the TA will look at them). **Very important: please start using the subversion system the moment you start developing**. (If you don't know what we are talking about, google and read about Subversion or ask us)

3. Please do not submit this assignment to the judgegirl system. This assignment will be graded by the TA, not by the judgegirl system. **At the deadline of this**

assignment, the TA will checkout the HEAD version (the latest version) from your repository and use that version for grading. There is no need to "submit" the assignment to anywhere else.

4. (**Requirements for teams with 3 students or bonus for teams with 2 students**) Instead of assuming that each symbol is a *4-byte binary string*, each symbol can actually be k byte in length, $k = 1, 2, ..., n$, where n is the length of the input file. The choice of k can affect the compression ratio. Please come up with a way to determine the right k value and implement it in your file compressor so that the compression ratio is minimized. Please explain how you determine k in your report. (Hint: Huffman coding is optimal when the probability of each input symbol is a negative power of two)

5. In the Subversion repository, you must have at least the following files:

   - **Makefile**. The TA should be able to execute "make all" in that directory and the GNU make system will compile and generate two executables "hc_compress" and "hc_decompress" as instructed by this makefile. You should make sure that this will work on linux*.csie.ntu.edu.tw. (If you do not know what this is all about, ask us or google for "makefile".)
   - **README**. This plain text file should explain what each file in the directory is for.
   - **report.pdf**. This file should explain the format of your output file (compressed file) and how your algorithm works. The TA should be able to read this report and understand how your code works.
   - All the source files (.c) and header files (.h) should also be in the repository.

6. The grade of this assignment will contribute 30% of HW4. It will be determined by the following:

   - Correctness (20%). We will use several test files to test against your compressor/decompressor. The input file and the file after compression and decompression should be identical.

- Report (10%).
- Optimization (bonus: 5%). The teams which develop compressors with a smaller compression ratio (ranked the first 25% among all teams) will get this bonus. Teams with 3 students will not get this bonus as it is part of the requirement.
- Students who do not contribute much to the development will only get a certain percentage of the score, determined by the TA by the commits made to the subversion system.

Problem 2. Please spend some time to read the NP-Completeness course material we covered in the class so far (section 34.1 and 34.2 in the textbook). Then solve the following to problems:

a. Show that the class P, viewed as a set of languages, is closed under union, intersection, concatenation, complement, and Kleene star. That is, if $L_1, L_2 \in P$, then $L_1 \cup L_2 \in P$, $L_1 \cap L_2 \in P$, $L_1 L_2 \in P$, $\bar{L_1} \in P$, and $L_1^* \in P$ (problem 34.1-6 in the textbook) (10%).

b. Prove that $P \subseteq$ *co-*NP (problem 34.2-9 in the textbook). *co-*NP is the set of languages L such that $\bar{L} \in NP$ (10%).