Programming Assignment 4

CMPE 250, Data Structures and Algorithms, Fall 2011

Instructor: A. T. Cemgil
TA's: Barış Kurt, Murat Arar, Zeynep Gözen Sarıbatur

Due: 14 Jan 2012, 23:59 Sharp

Problem Definition

In this project you are going to compress a file, using *Huffman Coding.* It's a lossless encoding algorithm with variable length codes for source symbols. The basic idea is to assign short codes for the frequently seen symbols. You can find the details of the algorithm in your course book. Here is a quick summary:

You are going to calculate the frequency of letters in a text document, and build a Huffman tree, according to it. For example, for the input string *AAAAAABCCCCCCDDEEEEE*, the frequencies, and codes are given at Table 1, and the corresponding Huffman tree is given at Figure 1. Instead of 8 bits per character, this coding scheme needs 2 bits for A,C,E and 3 bits for B and D. Therefore, overall size reduces from 160 bits to 43 bits. However, you have to include the Huffman tree into the encoded file in order to be able to restore it.

symbol	frequency	code
A	6	10
C	6	11
E	5	01
D	2	001
B	1	000

Table 1: Frequencies in the input sequence *AAAAAABCCCCCCDDEEEEE.*

Storing The Huffman Tree

One way of decoding the tree is to traverse it recusively and printing the symbols with the following rule:

1. If you're at a leaf node, output 1 and 8-bits for the character
2. If you're at a non-leaf node, output 0, visit left and right childs.

For the tree in Figure 1, the output sequence will be 0001B1D1E01A1C, where the letter A,B,C,D and E are replaced by 8-bits for their ASCII code.

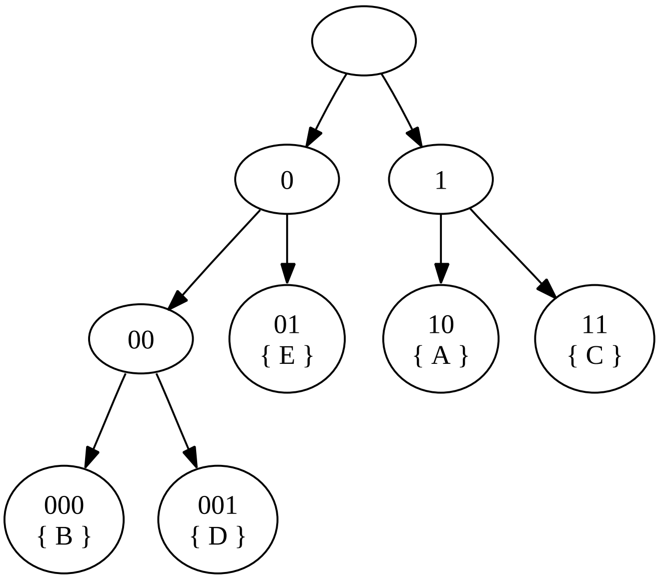

Figure 1: Huffman Tree for the input sequence *AAAAAABCCCCCCCDDEEEEE*.

Storing The Encoded Bitstream

Finally, you have to write your bitstream to a binary file. The problem here is that you can only read/write bytes. Therefore, you are going to write your bits in byte length blocks. We will use a simple convention here. The number of bits (Huffman tree + encoded text) is going to the written as an **unsigned integer** (4 bytes) in the beginning of the binary file. Then you will write your stream byte by byte. You have to be careful for reading and writing the last byte, since not all of the bits of the last byte may be valid. For the given above example, the Huffman tree, and the encoded text takes 92 bits (+4 bits to complete the last 8-bit block) The binary file will be like this:

920001B1D1E01A1C1010101010100001111111111111100100101010101010000

Test Cases

You can find 2 test cases in the course web site. Each test case includes the text, encoded version, and a graph file for the Huffman tree used in encoding. We are going to use ASCII alphabet with 128 distinct symbols.

How to run the program

Your program will be tested automatically with the following commands:

```
./your_program -e hamlet.txt hamlet.bin
./your_program -d hamlet.bin hamlet_decoded.txt
```

The first argument is -e for encoding, -d for decoding. The second and third arguments are the input and output file names respectively.

Submission & Grading

- **What, How and When to submit**
  - You should submit your project in electronic form.
  - You should compress your source code (.cpp and .h files) in a zip file, name it as [pr]_[#]_[Student ID] (e.g. `pr_4_200700803.zip`), and email to `bucmpe250@gmail.com`.
  - The subject of your e-mail must also be the same as your zip file (e.g. `pr_4_200700803`)
  - Your zip file should NOT contain any executable file (.exe) or any folder. If you sent multiple e-mails, only the last one will be taken into account.
  - The deadline is 14 Jan 2012, 23:59 Sharp. Emails tagged later won't be considered.
- **Grading**
  - Warning: All source codes are checked automatically for similarity with other submissions and exercises from previous years. Make sure you write and submit your own code.
  - Your program will be graded based the correctness of your output and the clarity of the source code. Correctness of your output will be tested automatically so make sure you stick with the format described above.
  - There are several issues that makes a code piece 'quality'. In our case, you are expected to use C++ as powerful, steady and flexible as possible. Use mechanisms that affects these issues positively.
  - Make sure you document your code with necessary inline comments, and use meaningful variable names. Do not over-comment, or make your variable names unnecessarily long.
  - Try to write as efficient (both in terms of space and time) as possible. Informally speaking, try to make sure that your program completes under 20 seconds.