CS380 Algorithm Design & Analysis

Assignment 6: Huffman Encoding

Date Assigned: Thursday, April 30, 2009
Date Due: Tuesday, May 12, 2009
Total Points: 50pts

Introduction

Computing power has increased enormously over the past few decades, but the demand for information has increased even more, so efficient data storage continues to be extremely important in computer science. Huffman coding is a statistical encoding and data compression scheme that uses unequal length codes to store information. Characters that occur most frequently are encoded with fewer bits; less frequent characters are encoded with more bits. When the distribution of characters is highly skewed, Huffman coding can result in significant compression rates.

Huffman Coding

Data compression programs consist of an encoder and a decoder. The encoder takes an input file and encodes it to an output file (presumably with lesser storage requirements). The decoder takes the encoded file as input and outputs a decoded version; with lossless coding schemes, the decoded file is identical to the original.

The Huffman encoder makes two passes through an input data file. In the first pass, the frequency of occurrence of each character in the file is determined, and a Huffman coding tree (a type of binary tree) is constructed. In the second pass through the input file, the encoder uses Huffman codes to compress the input data. For each character in the input file, the corresponding variable-length Huffman code is written to the output file. This results in an optimal encoding of the data (optimal in the sense of entropy).

For decoding, the same Huffman codes must be used. Bits are read from the encoded input file, and decoded into characters. These decoded characters are written to the decoded output file. Perhaps the simplest way to store the Huffman codes is to store the histogram at the start of the encoded file, and regenerate the tree inside the decoder. This adds 1K overhead to the encoded file size, which is insignificant for large files.

Problem

Write a Huffman encoder-decoder program in C++. The program should output a menu asking if the users wishes to encode or decode, then ask for the name of the file to be encoded or decoded.

Program Output

After encoding the following file:

```
ABCDEABCDEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCDDDDDDEEEEEE
```

Your program must output the following:

```
Huffman Encoder Pass 1
----------------------
Read 129 bytes from test.txt, found 5 code words

Huffman Code Table
------------------
ASCII Code Probability (%) Huffman Code
65 (A) 48.84 0
66 (B) 26.36 11
67 (C) 12.40 100
68 (D) 6.20 1010
69 (E) 6.20 1011

Huffman Encoder Pass 2
----------------------
Wrote 31 encoded bytes to test.z (1055 bytes including histogram)

Huffman Coding Statistics
-------------------------
Compression ratio = 24.03%
Entropy = 1.88
```

Notes

- For the min-priority queue, you can either use your own implementation of priority queues or the STL implementation of priority queues.
- For decompression to work with Huffman coding, information must be stored in the compressed file that allows the Huffman tree to be re-created so that decompression can take place. There are many options here. You can store all codes and lengths as normal ints or you can try to be inventive and save space. For example, it's possible to store just character counts and recreate the codes from the counts. It's also possible to store code-lengths and codes using bit-at-a-time operations. Any solution to storing information in the compressed file is acceptable. If you use a successful space-saving technique you can earn several points of extra credit (up to 5). Space-saving techniques are defined as those using less space than simply storing 257 counts as 32 bit ints. One useful technique is to write the tree to the file.
- Finally, because Huffman coding requires two passes over the input file you will need to rewind the underlying stream. The effect of rewinding the input stream is to allow it to be read again. You can do so as follows:

  ```
  in.clear();
  in.seekg(0, ios::beg); // Rewind the stream
  ```

What to Submit

- Submit an electronic copy of your project by 9:40am on the day that it is due. Name your project “06PUNETHuffman”, replacing PUNET with your PU Net ID (i.e. khoj0332).
- A summary of the time that you spent working on this assignment (give me a number!!!!), and what slowed you down the most. Submit this document electronically as a Google Document called “06PUNetPart1” for example “06khoj0332Part1”. Create the Google document and share it with me at ShereenKhoja@gmail.com.