Simple Data Compression

CS 10C Programming Assignment

Huffman coding is used to compress data. The idea is straightforward: represent more common longer strings with shorter ones via a basic translation matrix. The translation matrix is easily computed from the data itself by counting and sorting by frequency.

For example, in a well-known corpus used in Natural Language Processing called the "Brown" corpus (see nltk.org), the top-20 most frequent tokens, which are words or punctuation marks are listed below associated with frequency and code. The word "and" for example requires writing three characters. However, if I encoded it differently, say, using the word "5" (yes, I called "5" a word on purpose), then I save having to write two extra characters! Note, the word "and" is so frequent, I save those two extra characters many times over!

Token	Frequency	Code
the	62713	1
,	58334	2
.	49346	3
of	36080	4
and	27932	5
to	25732	6
a	21881	7
in	19536	8
that	10237	9
is	10011	10
was	9777	11
for	8841	12
``	8837	13
''	8789	14
The	7258	15
with	7012	16
it	6723	17
as	6706	18
he	6566	19
his	6466	20

So the steps of Huffman coding are relatively straightforward:

1. Pass through the data once, collecting a list of token-frequency counts.
2. Sort the token-frequency counts by frequency, in descending order.
3. Assign codes to tokens using a simple counter, for example by incrementing over the integers; this is just to keep things simple.

4. Store the new mapping (token -> code) in a hashtable called “encoder”.
5. Store the reverse mapping (code -> token) in a hashtable called "decoder".
6. Pass through the data a second time. This time, replace all tokens with their codes.

Now, be amazed at how much you've shrunk your data!

Delivery Notes:

(1) Implement your own hashtable from scratch, you are not allowed to use existing hash table libraries.

(2) To be useful, your output should include the coded data as well as the decoder (code -> token) mapping file.

Now GZIP all that and watch it shrink immensely!