Lempel-Zev

Drawbacks of Huffman’s:
- must read entire file before encoding
- encodes only single characters (not strings)

Ex: AAAA BAAA

Huffman’s 0 1 => A = 0  B = 1

encoded: 0001000

Better: AAA = 0  B = 1  encoded: 010

Recall for Huffman’s we did: char → binary
In practice: char → ASCII → binary.

Assume: have ASCII table (code for all 256 characters).

Goals:
- encode text while reading
- encode strings instead of chars.

Address both using a sliding window of length > 1
(all strings of length 1 already have ASCII codes).

Ex: the cat the hat

With sliding window of length 2:
encode: th, he, ec... (th and he already encoded). eh, ha...
But we want to be able to encode longer strings (e.g. "the") so window length should grow!

So, start with window of size 2 (all strings of length 1 already have ASCII codes) and extend the window when should we extend?

\[ \Rightarrow \text{When the current string has already been encoded} \]

ex: the cat the hat

th - encode
he - "
ec - " :

th - already encoded, extend window, encode "the" (since 'th' was already encoded we treat it like a single character, so don't break it up since unlikely that 'he' will appear later) so slide window to 'eh'

eh

ha :

Frequently checking whether a string has a code How to do this quickly?

Store strings, codes in a hash table/dictionary

Recall: All single chars are already in the table
Lempel-Ziv (msg, dictionary)

str = msg getNextChar()  // first char will be in
    // Dictionary

while (there are still chars in msg) {
    c = msg getNextChar()
    if (dictionary contains (str + c))
        str = str + c  // extend window length to
            // build longer string
    else
        output str code  // str will have a code
        dictionary.add (str + c)
        str = c  // slide the window
}

}  // after while

output str code  // last str
example: A B B A B B B B A B B A

newstr  code
AB     256
BB     257
BA     258
ABB    259
BBA    260
ABBA   261

output: 65 66 66 256 257 259 65

Reduction:

No compression : 11 chars x 8 bits = 88 bits
Compressed w/ LZ = 9 chars x 8 bits +

3 strings x 9 bits = 59 bits

need a few additional bits to indicate when a binary string ends.
Effectiveness on Alice in Wonderland

Huffman's: 15%
Lempel-Ziv: 50%

Runtime: n=length of msg, \(|A_1| =\) table size
\(O(n)\) if contains is \(O(1)\)
\(O(|A_1|n)\) if contains searches table (unlikely)

Another example:

A B C A C A D A B R A (\(C = 67, D = 68, R = 114\))

<table>
<thead>
<tr>
<th>newstr</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>256</td>
</tr>
<tr>
<td>BR</td>
<td>257</td>
</tr>
<tr>
<td>RA</td>
<td>258</td>
</tr>
<tr>
<td>AC</td>
<td>259</td>
</tr>
<tr>
<td>CA</td>
<td>260</td>
</tr>
<tr>
<td>AD</td>
<td>261</td>
</tr>
<tr>
<td>DA</td>
<td>262</td>
</tr>
<tr>
<td>ABR</td>
<td>263</td>
</tr>
<tr>
<td>68</td>
<td>256</td>
</tr>
<tr>
<td>258</td>
<td></td>
</tr>
</tbody>
</table>

Encoding: 65 66 114 65 67 65 DA 262 ABR 263

No compression: \(8(11) = 88\) bits
Compressed: \(8(7) + 2(9) = 74\) bits
Yet another example:

<table>
<thead>
<tr>
<th>newstr</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA</td>
<td>256</td>
</tr>
<tr>
<td>AB</td>
<td>257</td>
</tr>
<tr>
<td>BAA</td>
<td>258</td>
</tr>
<tr>
<td>ABA</td>
<td>259</td>
</tr>
<tr>
<td>AA</td>
<td>260</td>
</tr>
</tbody>
</table>

output: 66 65 256 257 65 260

ex (3):

<table>
<thead>
<tr>
<th>newstr</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>256</td>
</tr>
<tr>
<td>BA</td>
<td>257</td>
</tr>
<tr>
<td>ABA</td>
<td>258</td>
</tr>
</tbody>
</table>

65 65 256 258