Brotli Compression Algorithm
outline of a specification
Overview

- Structure of backward reference commands
- Encoding of commands
- Encoding of distances
- Encoding of Huffman codes
- Block splitting
- Context modeling
- Meta-block structure
**LZ77 decomposition**

Command:
- L bytes inserted (literals)
- C bytes copied from D bytes distance (C >= 2, D >= 1)

Ring buffer of last 4 distances

Distance short codes 1 - 16 for
- \(d_1, d_2, d_3, d_4, d_1 \pm \{1, 2, 3\}, d_2 \pm \{1, 2, 3\}\)
- other distances are shifted by 16

Example:
- distance sequence: 1, 10, 10, 1, 3, 4, 7, 5
- transformed to: 17, 26, 1, 2, 8, 6, 10, 7
## Encoding of commands

### Literal insertion length alphabet:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6 - 7</th>
<th>8 - 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 - 13</td>
<td>14 - 17</td>
<td>18 - 25</td>
<td>26 - 33</td>
<td>34 - 49</td>
<td>50 - 65</td>
<td>66 - 97</td>
<td>98 - 129</td>
<td></td>
</tr>
<tr>
<td>130 - 193</td>
<td>194 - 321</td>
<td>322 - 577</td>
<td>578 - 1089</td>
<td>1090 - 2113</td>
<td>2114 - 6209</td>
<td>6210 - 22593</td>
<td>22594 - 16799809</td>
<td></td>
</tr>
</tbody>
</table>

### Copy length alphabet:

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 - 11</td>
<td>12 - 13</td>
<td>14 - 17</td>
<td>18 - 21</td>
<td>22 - 29</td>
<td>30 - 37</td>
<td>38 - 53</td>
<td>54 - 69</td>
<td></td>
</tr>
<tr>
<td>70 - 101</td>
<td>102 - 133</td>
<td>134 - 197</td>
<td>198 - 325</td>
<td>326 - 581</td>
<td>582 - 1093</td>
<td>1094 - 2117</td>
<td>2118 - 16779333</td>
<td></td>
</tr>
</tbody>
</table>
## Joint literal- and copy length histogram

<table>
<thead>
<tr>
<th>Short literal seq.</th>
<th>Short copies</th>
<th>Medium copies</th>
<th>Long copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last dist.</td>
<td>0 - 63</td>
<td>64 - 127</td>
<td></td>
</tr>
<tr>
<td>Short literal seq.</td>
<td>128 - 191</td>
<td>192 - 255</td>
<td>384 - 447</td>
</tr>
<tr>
<td>Medium literal seq.</td>
<td>256 - 319</td>
<td>320 - 383</td>
<td>512 - 575</td>
</tr>
<tr>
<td>Long literal seq.</td>
<td>448 - 511</td>
<td>576 - 639</td>
<td>640 - 703</td>
</tr>
</tbody>
</table>

**Examples:**

- **Command (11, 19, 100)**
  - medium literal seq. (bucket 0)
  - medium copy (bucket 3)
  → code 323 + 4 extra bits

- **Command (0, 2, 1)**
  - short literal seq. last distance (bucket 0)
  - short copy (bucket 0)
  → code 0, no extra bits
Encoding of distances

Distance alphabet consists of:

- short codes for previous distances (1 - 16)
- $M$ number of “direct codes” encoded without extra bits
- buckets of length $2^n$ encoded with $n$ extra bits
- optionally buckets can have $k$ last bits fixed
- $2^{k+1}$ bucket for each number of extra bits

Example for $M = 12$ and $k = 1$

<table>
<thead>
<tr>
<th>short codes (1 - 16)</th>
<th>direct codes (17 - 28)</th>
<th>29, 31</th>
<th>30, 32</th>
<th>33, 35</th>
<th>36, 38</th>
</tr>
</thead>
<tbody>
<tr>
<td>39, 41, 43, 45</td>
<td>40, 42, 44, 46</td>
<td>47, 49, 51, 53</td>
<td>48, 50, 52, 54</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Encoding of Huffman codes

Similar to DEFLATE, with the following changes:

- Order of code length codes:
  
  17, 18, 0, 1, 2, 3, 4, 5, 16, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

- Special encoding of Huffman codes with <= 2 non-zero lengths
  
  ○ 1 bit for small tree marker
  ○ 1 bit for the number of symbols (1 or 2)
  ○ 1 bit to indicate if the first symbol is 0 or 1
  ○ 1 or 8 bits for the first symbol
  ○ 8 bits for the second symbol, if present

- No bits emitted if Huffman code has only one symbol
Block splitting

Literal stream
Command stream
Distance stream

Independent block boundaries for literals, commands and distances:
## Encoding of block switch symbols

<table>
<thead>
<tr>
<th>Type1, Len1</th>
<th>Type2, Len2</th>
<th>Type3, Len3</th>
<th>Type4, Len4</th>
</tr>
</thead>
</table>

### Block length alphabet:

<table>
<thead>
<tr>
<th>1 - 4</th>
<th>5 - 8</th>
<th>9 - 12</th>
<th>13 - 16</th>
<th>17 - 24</th>
<th>25 - 32</th>
<th>33 - 40</th>
<th>41 - 48</th>
</tr>
</thead>
<tbody>
<tr>
<td>49 - 64</td>
<td>65 - 80</td>
<td>81 - 96</td>
<td>97 - 112</td>
<td>113 - 144</td>
<td>145 - 176</td>
<td>177 - 208</td>
<td>209 - 240</td>
</tr>
<tr>
<td>241 - 304</td>
<td>305 - 368</td>
<td>369 - 496</td>
<td>497 - 752</td>
<td>753 - 1264</td>
<td>1265 - 2288</td>
<td>2289 - 4336</td>
<td>4337 - 8432</td>
</tr>
<tr>
<td>8433 - 16624</td>
<td>16624 - 16793839</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Max 254 block types
- Special block type codes for
  - second last block type
  - last block type + 1
- Block switch is encoded with
  - block type code
  - block length code + extra bits
Encoding of the block split

- 1 bit to indicate if we have more than one block type
- 8 bits for number of block types - 1
- Huffman code on block type alphabet
- Huffman code on block length alphabet
- Length of first block
  - codeword based on Huffman code
  - extra bits
Context modeling for literals

<table>
<thead>
<tr>
<th>literal block type</th>
<th>B_{-3}</th>
<th>B_{-2}</th>
<th>B_{-1}</th>
<th>B_0</th>
</tr>
</thead>
</table>

- Context id
- Context map
- Entropy code index
- Entropy codes
Context modeling for literals

<table>
<thead>
<tr>
<th>literal block type</th>
<th>...</th>
<th>B_3</th>
<th>B_2</th>
<th>B_1</th>
<th>B_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 - 15</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16 - 63</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>64 - 127</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>128 - 191</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>192 - 239</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>240 - 254</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>255</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- 0: 3 bits
- 1 - 63: 2 bits
- 64 - 254: 2 bits
- 255: 1 bit

6 bit context id
Context modeling for distances

- Distance
- Block type
- Copy length
- Distance

- Context map
- Context id
- Entropy code index
- Entropy codes
Encoding of the context map

- 1 bit to enable/disable context modeling
- 4 bits for the context mode id (only in the literal context map)
- 8 bits for the number of entropy codes
- 1 bit to enable/disable run-length coding of zero sequences
- 4 bits for max zero-run-length bucket
- 1 bit to enable/disable Move-to-Front transformation
- Huffman code on entropy code ids + run length codes
- row-by-row encoding of the entropy code ids and run length codes using the above Huffman code
Format specification

<compressed file>:
  <uncompressed size>
  (<meta-block size><meta-block>)*

<uncompressed size>:
  number_of_bytes  3 bits
  size              number_of_bytes bytes

<meta-block size>:
  is_last  1 bit
  size     log2(uncompressed size) * is_last bits

Example:
  empty file is encoded with 3 zero bits
Meta-block format

<meta-block>:
  <literal block split>
  <command block split>
  <distance block split>
  <distance postfix bits and num direct codes>
  <literal context map>
  <distance context map>
  <literal Huffman code>*
  <command Huffman code>+ 
  <distance Huffman code>*
  [<command block switch>??<command>
   ( <literal block switch>??<literal> )]*
  <distance block switch>??<distance>??]+