Non-Binary Huffman Codes

Sometimes one has to add phantom source symbols with 0 probability in order to make a non-binary Huffman code.

The basic strategy to create a Huffman code where the code words are from an alphabet with \( n \) letters is to:

(a) Order probabilities high to low (perhaps with extra symbols with probability 0)
(b) Combine \( n \) least likely probabilities. Add them and re-order
(c) End up with \( n \) symbols (i.e., probabilities)!!

To be true we may need to add phantom symbols.

Example \( n=3 \) \{A, B, C, D\} \((p_1, p_2, p_3, p_4) = (0.5, 0.3, 0.1, 0.1)\)

Wrong

\[
\begin{array}{c}
0 \quad 0.5 \quad 0 \\
10 \quad 0.3 \quad 10 \\
11 \quad 0.1 \quad 11 \\
12 \quad 0.1 \quad 12 \\
\end{array}
\]

\( L_1 = 1.5 \)

Right

\[
\begin{array}{c}
0 \quad 0.5 \quad 0 \\
1 \quad 0.3 \quad 1 \\
20 \quad 0.1 \quad 20 \\
21 \quad 0.3 \quad 21 \\
X \quad 0 \quad 0 \\
\end{array}
\]

\( L_1 = 1.2 \)

No U.D. ternary code will have smaller \( L_1 \)!!!

Compare with entropy (base 3)

\[
H_3(S) = 0.5 \log_3 0.5 + 0.3 \log_3 0.3 + 0.1 \log_3 0.1 + 0.1 \log_3 0.1
\]

\( = 1.063 \)
If one starts with $M$ source symbols and one combines the $r$ least likely into one symbol, one is left with $M - r(2 - 1)$ symbols.

After doing this $d$ times, one is left with $M - d(2 - 1)$ symbols.

But at the end we must be left with $r$ symbols. If this is not the case, we must add phantom symbols.

Thus we must add "D" phantom symbols to ensure that $M + D - \alpha(2 - 1) = r$ or $(M + D) = \alpha'(2 - 1) + 1$

**Example:** $r = 3$

<table>
<thead>
<tr>
<th>M</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example:** $r = 4$

<table>
<thead>
<tr>
<th>M</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

**Example:** $r = 5$

<table>
<thead>
<tr>
<th>M</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example:** $r = 6$

<table>
<thead>
<tr>
<th>M</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
</tr>
</tbody>
</table>
Run Length Codes for Fax (Black/White)

... www BBwwwBBBBwwwwww...
Encoding of

variable length source sequences

Previously we only considered the situation where we encoded N source symbols into variable length code sequences for a fixed value of N.

We could call this "fixed length to variable length" encoding.

But another possibility exists. We could encode variable length source sequences into fixed or variable length code words.

Example: \( \{A, B\} \) and source \((P_A, P_B) = (.9, .1)\)

**Code Book**

<table>
<thead>
<tr>
<th>Source Sequences</th>
<th>Codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>AA</td>
<td>01</td>
</tr>
<tr>
<td>AAB</td>
<td>10</td>
</tr>
<tr>
<td>AAA</td>
<td>11</td>
</tr>
</tbody>
</table>

Average length of source phrase \(= 1 \times .1 + 2 \times .09 + 3 \times (.01\times .75) = 2.71\)

Average \# of code symbols/source symbol \(= \frac{2}{2.71} = .739\)
Tunstall codes - U.D. variable to fixed encoding - binary code words

Basic idea - to encode into binary code words of fixed length L, make 2^L source phrases that are as nearly equally probable as we can.

We do this by making the source phrases as leaves of a tree and always splitting the leaf with the highest probability.

Example: \((A, B, C, D) \quad (p_1, p_2, p_3, p_4) = (0.5, 0.3, 0.1, 0.1)\)

\[
\begin{array}{c}
A \quad 0.5 \\
B \quad 0.3 \\
C \quad 0.1 \\
D \quad 0.1 \\
\end{array}
\]

\[
\begin{array}{cccc}
| & A & B & C & D \\
|---|---|---|---|
0.25 & 0.15 & 0.05 & 0.05 \\
\end{array}
\]

\[
\begin{array}{cccc}
| & A & B & C & D \\
|---|---|---|---|
0.15 & 0.09 & 0.03 & 0.03 \\
\end{array}
\]

\[
\begin{array}{cccc}
| & A & B & C & D \\
|---|---|---|---|
0.125 & 0.075 & 0.025 & 0.025 \\
\end{array}
\]

\[
\begin{array}{cccc}
| & A & B & C & D \\
|---|---|---|---|
0.075 & 0.045 & 0.015 & 0.015 \\
\end{array}
\]

Source phrases = \{D, C, BB, BC, BD, BAA, BAB, BAC, BAD, AB, AC, AD, AAB, AAC, AAD\}
## Codabook

<table>
<thead>
<tr>
<th>Source Phrases</th>
<th>Codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>0000</td>
</tr>
<tr>
<td>C</td>
<td>0001</td>
</tr>
<tr>
<td>BB</td>
<td>0010</td>
</tr>
<tr>
<td>BC</td>
<td>0011</td>
</tr>
<tr>
<td>BD</td>
<td>0100</td>
</tr>
<tr>
<td>BAA</td>
<td>0101</td>
</tr>
<tr>
<td>BAB</td>
<td>0110</td>
</tr>
<tr>
<td>BAC</td>
<td>0111</td>
</tr>
<tr>
<td>BAD</td>
<td>1000</td>
</tr>
<tr>
<td>AB</td>
<td>1001</td>
</tr>
<tr>
<td>AC</td>
<td>1010</td>
</tr>
<tr>
<td>AD</td>
<td>1011</td>
</tr>
<tr>
<td>AAA</td>
<td>1100</td>
</tr>
<tr>
<td>ABB</td>
<td>1101</td>
</tr>
<tr>
<td>AAC</td>
<td>1110</td>
</tr>
<tr>
<td>AAD</td>
<td>1111</td>
</tr>
</tbody>
</table>

**Average length of source phrase = sum of probabilities of internal nodes**

\[
= 1 + .5 + .3 + .25 + .15 = 2.2
\]

**Average number of code symbols/source symbol**

\[
= 4 / 2.2 = 1.82
\]
IMPROVED TUNSTALL CODING

Since the phrases are not equally probable, one can use a Huffman code on the phrases.

The result is encoding a variable number of source symbols into a variable number of code symbols.

Example: \((A, B) \quad (P_1, P_2) = (0.9, 0.1)\)

```
\[ E_{\text{Huffman}} = \frac{2}{2.71} = 0.729 \]

\[ 11 \quad 0.1 \]

\[ 110 \quad 0.09 \]

\[ 111 \quad 0.81 \]
```

Source Phrases | Codewords | Au # of code symbols
---|---|---
B | 11 | \( (1 + 0.271 + 0.171) \)
AB | 110 | \( \frac{1.442}{2.71} = 0.532 \)
AAA | 0 | \( E_{\text{eff}} = \frac{4.69}{5.32} = 0.881 \approx 88.1\% \)
ABB | 111 |
Summary of Results for $(A, B) = (0.9, 0.1)$

All of the following use 4 code words in coding table

1. Huffman Code
   
   \begin{align*}
   AA & \rightarrow 0 \\
   AB & \rightarrow 11 \\
   BA & \rightarrow 100 \\
   BB & \rightarrow 101
   \end{align*}

   Efficiency \geq 72.7\%

2. Shannon-Fano Code

   Efficiency \geq 72.7\%

3. Ternary Code
   
   \begin{align*}
   B & \rightarrow 00 \\
   AB & \rightarrow 01 \\
   AAB & \rightarrow 10 \\
   AAA & \rightarrow 11
   \end{align*}

   ESS \geq 63.5\%

4. Ternary / Huffman
   
   \begin{align*}
   B & \rightarrow 11 \\
   AB & \rightarrow 100 \\
   AAA & \rightarrow 0 \\
   AAB & \rightarrow 101
   \end{align*}

   ESS = 85.1\%
LEMPEL-ZIV SOURCE CODING

THE BASIC IDEA IS THAT IF WE HAVE A
DICTIONARY OF $2^n$ SOURCE PHRASES
(AVAILABLE AT BOTH THE ENCODER AND
THE DECODER) IN ORDER TO ENCODE
ONE OF THESE PHRASES ONE NEEDS ONLY
"A" BINARY DIGITS.

NORMALLY A COMPUTER STORES EACH SYMBOL
AS AN ASCII CHARACTER OF 8 BINARY DIGITS,
(ACTUALLY ONLY 7 ARE NEEDED).

USING L-Z ENCODING, FAR LESS THAN 7 BINARY
DIGITS PER SYMBOL ARE NEEDED. TYPICALLY THE
COMPRESSIION IS ABOUT 2:1 OR 3:1

THERE ARE TWO VERSIONS OF L-Z CODES. WE
WILL ONLY DISCUSS THE "WINDOW" VERSION. IN
THIS VERSION, SYMBOLS THAT HAVE ALREADY BEEN
ENCODED ARE STORED IN A WINDOW. THE ENCODER
THEN LOOKS AT THE NEXT SYMBOLS TO BE
ENCODER TO FIND THE LONGEST STRING THAT
IS IN THE WINDOW THAT MATCHES THE SOURCE
SYMBOLS TO BE ENCODED,
If it can't find the next symbol in the window, it sends a "0" followed by the 8 (or 7) bits of the ASCII character.

If it finds a sequence of ONE OR MORE symbols in the window, it sends a "1" followed by the bit position of the first symbol in the match followed by the length of the match. These latter two quantities are encoded into binary.

Then the sequence that was just encoded is put into the window.

**Example**

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

"THE" is encoded as (1, "15", "4")

```
\uparrow \uparrow
```

4 bits 7 bits

And then the window contains

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

"THREE ARE IN THE"
<table>
<thead>
<tr>
<th>Example of LEMPEL-ZIV Encoding</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>240.0310</td>
<td>38</td>
</tr>
<tr>
<td>280.0310</td>
<td>39</td>
</tr>
<tr>
<td>350.0310</td>
<td>40</td>
</tr>
<tr>
<td>350.0310</td>
<td>41</td>
</tr>
<tr>
<td>350.0310</td>
<td>42</td>
</tr>
<tr>
<td>350.0310</td>
<td>43</td>
</tr>
<tr>
<td>350.0310</td>
<td>44</td>
</tr>
<tr>
<td>350.0310</td>
<td>45</td>
</tr>
<tr>
<td>350.0310</td>
<td>46</td>
</tr>
<tr>
<td>350.0310</td>
<td>47</td>
</tr>
<tr>
<td>350.0310</td>
<td>48</td>
</tr>
<tr>
<td>350.0310</td>
<td>49</td>
</tr>
<tr>
<td>350.0310</td>
<td>50</td>
</tr>
<tr>
<td>350.0310</td>
<td>51</td>
</tr>
<tr>
<td>350.0310</td>
<td>52</td>
</tr>
<tr>
<td>350.0310</td>
<td>53</td>
</tr>
<tr>
<td>350.0310</td>
<td>54</td>
</tr>
<tr>
<td>350.0310</td>
<td>55</td>
</tr>
<tr>
<td>350.0310</td>
<td>56</td>
</tr>
<tr>
<td>350.0310</td>
<td>57</td>
</tr>
<tr>
<td>350.0310</td>
<td>58</td>
</tr>
<tr>
<td>350.0310</td>
<td>59</td>
</tr>
<tr>
<td>350.0310</td>
<td>60</td>
</tr>
<tr>
<td>350.0310</td>
<td>61</td>
</tr>
<tr>
<td>350.0310</td>
<td>62</td>
</tr>
<tr>
<td>350.0310</td>
<td>63</td>
</tr>
<tr>
<td>350.0310</td>
<td>64</td>
</tr>
<tr>
<td>350.0310</td>
<td>65</td>
</tr>
<tr>
<td>350.0310</td>
<td>66</td>
</tr>
<tr>
<td>350.0310</td>
<td>67</td>
</tr>
<tr>
<td>350.0310</td>
<td>68</td>
</tr>
<tr>
<td>350.0310</td>
<td>69</td>
</tr>
<tr>
<td>350.0310</td>
<td>70</td>
</tr>
<tr>
<td>350.0310</td>
<td>71</td>
</tr>
<tr>
<td>350.0310</td>
<td>72</td>
</tr>
<tr>
<td>350.0310</td>
<td>73</td>
</tr>
<tr>
<td>350.0310</td>
<td>74</td>
</tr>
<tr>
<td>350.0310</td>
<td>75</td>
</tr>
<tr>
<td>350.0310</td>
<td>76</td>
</tr>
<tr>
<td>350.0310</td>
<td>77</td>
</tr>
<tr>
<td>350.0310</td>
<td>78</td>
</tr>
<tr>
<td>350.0310</td>
<td>79</td>
</tr>
<tr>
<td>350.0310</td>
<td>80</td>
</tr>
<tr>
<td>350.0310</td>
<td>81</td>
</tr>
<tr>
<td>350.0310</td>
<td>82</td>
</tr>
<tr>
<td>350.0310</td>
<td>83</td>
</tr>
<tr>
<td>350.0310</td>
<td>84</td>
</tr>
<tr>
<td>350.0310</td>
<td>85</td>
</tr>
<tr>
<td>350.0310</td>
<td>86</td>
</tr>
<tr>
<td>350.0310</td>
<td>87</td>
</tr>
<tr>
<td>350.0310</td>
<td>88</td>
</tr>
<tr>
<td>350.0310</td>
<td>89</td>
</tr>
<tr>
<td>350.0310</td>
<td>90</td>
</tr>
<tr>
<td>350.0310</td>
<td>91</td>
</tr>
<tr>
<td>350.0310</td>
<td>92</td>
</tr>
<tr>
<td>350.0310</td>
<td>93</td>
</tr>
<tr>
<td>350.0310</td>
<td>94</td>
</tr>
<tr>
<td>350.0310</td>
<td>95</td>
</tr>
<tr>
<td>350.0310</td>
<td>96</td>
</tr>
<tr>
<td>350.0310</td>
<td>97</td>
</tr>
<tr>
<td>350.0310</td>
<td>98</td>
</tr>
<tr>
<td>350.0310</td>
<td>99</td>
</tr>
<tr>
<td>350.0310</td>
<td>100</td>
</tr>
</tbody>
</table>
EXAMPLE: ENCODE THE TEXT
"MY_MY_MY_WHAT_A_HAT-IS_THAT"

16 BIT WINDOW
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

MY MY MY MY MY MY MY MY MY MY MY MY MY MY
MY MY MY MY MY MY MY MY MY MY MY MY MY
MY MY MY MY MY MY MY MY MY WHAT-
MY MY MY MY WHAT-
MY MY MY WHAT-
MY MY WHAT-
MY WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-
WHAT-