Exercise Sensor Networks (due till December 8, 2008)

Lecture 6: Compression

Exercise 6.1: Huffman codes

(a) Encode (compress) the string MISSISSIPPI using Huffman-encoding.

Solution:

The text to be encoded consists of 11 characters. As a result, we get the following relative occurrences:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Relative Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>1/11</td>
</tr>
<tr>
<td>I</td>
<td>4/11</td>
</tr>
<tr>
<td>S</td>
<td>4/11</td>
</tr>
<tr>
<td>P</td>
<td>2/11</td>
</tr>
</tbody>
</table>

First, we combine the rarest two symbols into a new one and sum up the two relative occurrences (1/11+2/11=3/11). They will form the deepest branch in the tree.

We get a new list of symbols and again, we combine the two rarest symbols into another node of the tree. Note that in some cases we might get two not yet connected branches of the tree. However, not in this case:

- M: 1/11
- P: 2/11
- I: 4/11
- S: 4/11
Exercise Sensor Networks

Lecture 6: Compression

Exercise 6.1: Huffman codes

Now, we only need to assign either 1 or 0 to each branch and get the optimal Huffman code:

\[
\begin{align*}
M &: 111 \\
P &: 110 \\
I &: 10 \\
S &: 0
\end{align*}
\]

Note that the Huffman tree always produces a prefix-tree because the symbols are only attached to the leaves. Thus, no symbol can be a (leading) bit pattern of another one.

The message MISSISSIPPI would be coded as: 111 10 0 10 0 10 110 110 10

(b) Produce the table for fast decoding of the Huffman code.

<table>
<thead>
<tr>
<th>bits</th>
<th>symb</th>
<th>no symbol bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>M</td>
<td>3</td>
</tr>
<tr>
<td>110</td>
<td>P</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>I</td>
<td>2</td>
</tr>
<tr>
<td>101</td>
<td>I</td>
<td>2</td>
</tr>
<tr>
<td>000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>001</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>011</td>
<td>S</td>
<td>1</td>
</tr>
</tbody>
</table>
Exercise Sensor Networks

Lecture 6: Compression

Exercise 6.2: Arithmetic encoding

Using the relative occurrences from the last exercise, encode (compress) the string MISP using the arithmetic encoding. P is considered to be the terminal symbol which means that the decoder knows that it can stop, once a P is encountered. So in order to encode the entire message, you will have to pick a (binary) number from the interval for P in the end. Try to go for the shortest number you can find in the P-interval to use as few bits as possible.
Exercise Sensor Networks

Lecture 6: Compression

Exercise 6.2: Arithmetic encoding

M : 1/11
I : 4/11
S : 4/11
P : 2/11

The last final P-interval consists of

10001100
10010101

The shortest bit string between those two intervals is

1001

So 00001001 will code the message MISP.
Lecture 6: Compression

Exercise 6.2: Arithmetic encoding

The arithmetic coding is known to be a generalization of Huffman. Explain why arithmetic coding can be at least as efficient as Huffman but not vice versa. In particular, which advantage does arithmetic encoding have as compared to Huffman?

Solution:

Huffman assigns “full codewords” to symbols by using a fixed number of bits. As a consequence, the probabilities which correspond to the number of bits spent can only be of the kind $1/2^n$.

Remember that $-\log_2(p)$ results in the number of bits that correspond to a given probability $p$. Since Huffman only assigns “entire” bits, $p$ can only be $1/2$, $1/4$, $1/8$, and so on. If the true occurrences of symbols are different, than Huffman can only assign the closest match.

In contrast, arithmetic encoding partitions another domain. In particular, in assigns parts of an interval to a symbol. Rather than the coarse assignments of Huffman, the interval can be partitioned finer. Thus, arithmetic encoding adapts much better to the true occurrences of symbols than Huffman can.