X = (X, p)

\[ X = \{ x_1, \ldots, x_n \} \]

\[ p(x) = 2^x, \quad x = 1, \ldots, n-1, \quad p(n) = 2^{(n-1)} \]

Huffman code:

1. \[ \frac{1}{2} \rightarrow \frac{1}{2} \rightarrow \frac{1}{2} \rightarrow \cdots \]
2. \[ \frac{1}{2} \rightarrow \frac{1}{2} \rightarrow \frac{1}{2} \rightarrow \cdots \]
3. \[ \frac{1}{2} \rightarrow \frac{1}{2} \rightarrow \frac{1}{2} \rightarrow \cdots \]
4. \[ \frac{1}{2} \rightarrow \frac{1}{2} \rightarrow \frac{1}{2} \rightarrow \cdots \]

\[ \vdots \]

n-2 \[ \frac{1}{2^{(n-1)}} \rightarrow \frac{1}{2^{(n-2)}} \rightarrow \cdots \]

n-1 \[ \frac{1}{2^{(n-1)}} \rightarrow \frac{1}{2^{(n-2)}} \rightarrow \cdots \]

n \[ \frac{1}{2^{(n-1)}} \rightarrow \frac{1}{2^{(n-2)}} \rightarrow \cdots \]

\[ \vdots \]

Observe that:

\[ l(1) = 1, \quad l(2) = 2, \quad l(3) = 3, \ldots, \quad l(n-1) = n-1, \quad l(n) = n-1 \]

The average codeword length

\[ L(C, p) = \sum_{i=1}^{n-1} i \cdot 2^i + (n-1) \cdot 2^{(n-1)} \]

On the other hand,

the entropy

\[ H(p) = -\sum_{i=1}^{n-1} 2^i \log_2 2^i - 2^{(n-1)} \log_2 2^{(n-1)} \]

\[ = \sum_{i=1}^{n-1} i \cdot 2^i + (n-1) \cdot 2^{(n-1)} \]

\[ = L(C, p) \quad \text{(From 1)} \]

\[ \therefore \text{the entropy bound is indeed achieved!} \]
Codebook:
\[ \{01, 100, 101, 1110, 1111, 0011, 0001 \} \]

Tree representation:

Observe that the codewords are not optimally assigned in the highlighted regions of the tree, hence, this cannot be a Huffman code.

Let us first take the case \( k = 3 \).

Obviously, 3 symbols cannot be represented by just 1 bit.

Thus, any fixed-length code has an average codeword length of at least 2 bits.

On the other hand, consider the variable-length code \( \{0, 10, 11 \} \). The average codeword length is
\[
\frac{1 + 2 + 2}{3} = 1.67 \text{ bits}
\]

and hence is more efficient than the fixed-length code. Thus, the assertion is not true in this case.
Next, take $K$ to be a power of 2, i.e.
\[ K = 2^n, \quad n > 0 \text{ integer}, \]

Since the source is equiprobable, the optimal code have average codeword length
\[ L(C, P) \geq H_2(X) = \log_2 K = n \text{ bits}. \]

We will now construct a fixed-length code that has length $n$ bits and hence is as efficient as any code can be. We take the binary tree of depth $n$ and label each of the $2^n$ leaves as the codewords. For e.g. when $n = 3$,

This fixed-length code is 100\% efficient.

Thus, the assertion in the problem is true if
\[ K = 2^n, \quad n > 0 \text{ integer}. \]
(a) code I:

[Diagram of a binary tree]

PREFIX!

code II:

[Diagram of a binary tree]

Not prefix.

code III:

[Diagram of a binary tree]

Not prefix

code IV:

[Diagram of a binary tree]

PREFIX!

(b) Kraft inequality:

Code I: \[ 2^{-1} + 2^{-2} + 2^{-3} + (2 \times 2^{-4}) = 1 \] satisfied!

Code IV: \[ (3 \times 2^{-2}) + (2 \times 2^{-3}) = 1 \] satisfied!

Since code I and code IV are prefix, Kraft inequality is satisfied as expected.
Huffman code 1: (move combined symbol as high as possible).

* Here, a different way of representing the combining of symbols is demonstrated. You may choose to use either the one in the book, or this, whichever you find convenient.

** moving a combined symbol as high as possible essentially means combined symbols take the lowest precedence in combining further (when there are ties).

codebook:

a: 010
i: 011
l: 000
m: 100
n: 101
o: 001
p: 110
y: 111

Avg. codeword length = 3

Variance = 0
Huffman code 2: (move combined symbol as low as possible i.e. combined symbols take highest precedence in combining further).

Codebook

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>100</td>
</tr>
<tr>
<td>i</td>
<td>101</td>
</tr>
<tr>
<td>e</td>
<td>00</td>
</tr>
<tr>
<td>m</td>
<td>110</td>
</tr>
<tr>
<td>n</td>
<td>111</td>
</tr>
<tr>
<td>o</td>
<td>010</td>
</tr>
<tr>
<td>p</td>
<td>0110</td>
</tr>
<tr>
<td>y</td>
<td>0111</td>
</tr>
</tbody>
</table>

Avg. codeword length = \[0.2 \times (2 + 3) + 0.1 \times (4 + 4 + 3 + 3 + 3 + 3)\]

= 3

Variance = \[(4 - 3)^2 \times 0.2 + (2 - 3)^2 \times 0.2\]

= 0.4
codebook:

\[s_0: 00\]
\[s_1: 01\]
\[s_2: 100\]
\[s_3: 110\]
\[s_4: 111\]
\[s_5: 1010\]
\[s_6: 1011\]

Avg. codeword length =

\[6 \times 0.25 \times 2 + (3 \times 0.125) \times 3 + (4 \times 0.0625) \times 2\]

= 2.625

\[H_2(S) = -(0.25 \log_2 0.25) \times 2 - (0.125 \log_2 0.125) \times 3 - (0.0625 \log_2 0.0625) \times 2\]

= \((0.25 \times 2) \times 2 + (0.125 \times 3) \times 3 + (0.0625 \times 4) \times 2\)

= 2.625.

Thus, we have 100% efficiency, i.e., entropy bound is achieved.

This is because the pmf of source is of the form

\[p(s_i) = 2^{-n_i}\]

with \(n_i \in \mathbb{N}_0\), \(i = 0, 1, \ldots, 6\).
8

8 = \{ A, B, C, D \}

Take \( p = \left( \frac{1}{3}, \frac{1}{3}, \frac{1}{6}, \frac{1}{6} \right) \)

Codebook:

\begin{align*}
A & : 00 \\
B & : 01 \\
C & : 10 \\
D & : 11
\end{align*}

Codebook:

\begin{align*}
A & : 0 \\
B & : 10 \\
C & : 110 \\
D & : 111
\end{align*}

\begin{align*}
\text{Symbol} & \quad \text{codeword} & \text{Length} \\
A & : 1 & 1 \\
B & : 011 & 3 \\
C & : 010 & 3 \\
D & : 001 & 3 \\
E & : 0001 & 4 \\
F & : 00001 & 5 \\
G & : 00000 & 5
\end{align*}
Recall that the expected codeword length of the optimal code satisfies

$$H_2(X) \leq L(C, p) \leq H_2(X) + 1.$$  

Hence, this is also true for the Huffman code.

The difference satisfies

$$0 \leq L(C, p) - H_2(X) \leq 1.$$  

Thus, we will now try to find a source for which the above difference is close to 1.

Let us take a 2-symbol source, say $X = 0, 1$.

Let $P(X=0) = 1-p$, $P(X=1) = p$.  \(0 < p < \frac{1}{2}\).

$$H_2(X) = -p \log_2 p - (1-p) \log_2 (1-p).$$

The optimal Huffman code uses 1 bit for each symbol and hence $L(C, p) = 1$.

In order to make the difference $L(C, p) - H_2(X)$ close to 1, we make the entropy $H_2(X)$ close to 0 by choosing $p$ to be a small positive constant.

By taking $p$ to be arbitrarily close to 0, we can make the difference $L(C, p) - H_2(X)$ to be arbitrarily close to 1. This gives the required source!
already stored

0 1 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 0 ...

1 2 3 4 5 6 7 8 9

↓ ↓ ↓ ↓ ↓ ↓ ↓

Compressed: 00101 00100 01000 00110 00010 01001 10000

4 bits for location

not enough information.