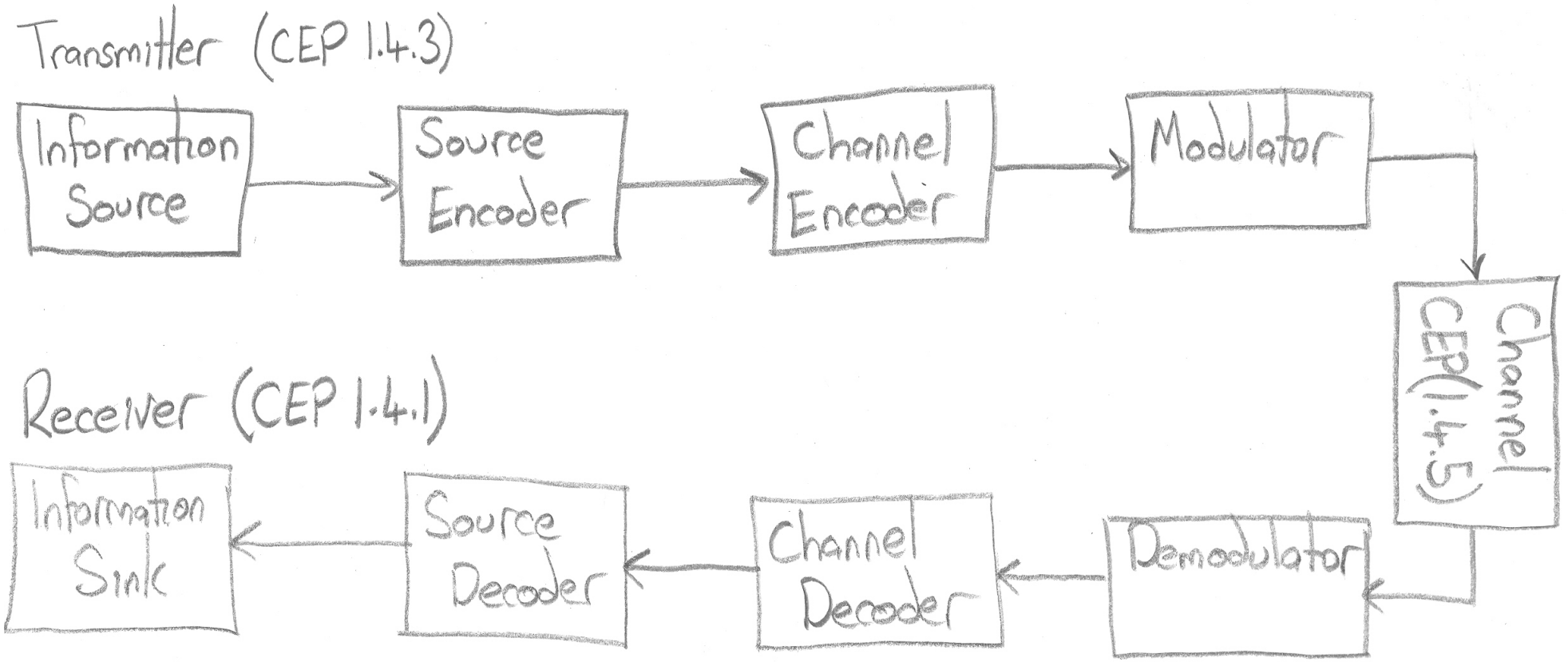
ELEC1323 Communications

9 Source Coding

Communication schemes



Source enCOder and DECoder (codec) (CEP 1.4.3.1 – 1.4.3.2)

- Source encoder converts the information to a format that is suitable for transmission.
- Source decoder converts it back again.
- e.g.
  - Multiplexing combines several signals into one. e.g. the left and right channels of stereo audio, the red, green and blue components of component video or the audio and video of a television signal.
  - Low Pass Filtering (LPF) to limit the bandwidth of the signal in order to avoid aliasing or to reduce the amount of spectrum required.
  - Analogue-to-Digital Conversion (ADC) if we want to use digital modulation to transmit an analogue signal. This uses sampling and quantisation.

Source enCOder and DECoder (codec) continued

- More examples of source codec functions
  - Encryption to protect the information from being decoded by an unauthorised receiver. Decoding is performed using a key that is built in to authorised receivers or SIM cards.
  - Watermarking to prove that the information has not been tampered with or sent from an unauthorised transmitter. The source decoder compares the received watermark with one that it has built in.
  - Compression to reduce the amount of information we have to transmit. Video, image and audio codecs typically use lossy compression, which discards information that we (hopefully) won't notice too much. Quantisation and LPF can be considered to be lossy compression techniques. Data such as computer programs must be compressed *losslessly* so that they can be exactly reconstructed by the source decoder. Lossless compression only discards redundant information in the message. e.g. run-length coding, Morse code, Huffman coding, arithmetic coding, zip files. Sampling is lossless so long as the sampling period is short enough.

Information and redundancy

- Any message can be considered to contain two things:
  - Information, which cannot be removed without harming the integrity of the message.
  - Redundancy, which can be removed without harming the integrity of the message.

- For example, suppose you wanted your friend to know when your evening class starts. You could say "My evening class starts at 7pm". Here, the "pm" part is redundant. It wouldn't be an evening class if it started at 7am. The message can be shortened to "My evening class starts at 7" without losing any of the information.

Compression

- Compression is a source coding technique for reducing the length of a message required to convey some information:
  - Lossless compression removes only redundancy from the message. e.g. zip file to compress a computer program.
  - Lossy compression also removes some information, but (hopefully) only the least important information. e.g. jpeg file to compress an image.

Quantifying information

- The amount of information in a message can be quantified in bits.
- For example, a message that conveys the result of flipping a coin contains $k = 1$ bit of information. This is because there are $N = 2$ (equally likely) outcomes of flipping a coin (heads and tails) and $N = 2$ values that $k = \log_2(N) = 1$ bit can have (0 and 1).
- If somebody asked me which season I was born in, my reply would contain $k = 2$ bits of information. This is because there are $N = 4$ (equally likely) replies that I could give (Winter, Spring, Summer, Autumn) and $N = 4$ values that $k = \log_2(N) = 2$ bits can have (00, 01, 10 and 11).
- A message that conveys the result of throwing an $N = 6$-sided dice would contain $k = \log_2(N) = 2.59$ bits of information. A message doesn't have to contain an integer number of bits of information!

Quantifying information cont

- Suppose a message is selected from a set of N possibilities. The number of bits of information k_i that is conveyed by the i^{th} possibility is related to its probability of being selected p_i according to $k_i = \log_2(1/p_i)$

- In the case of the $N = 6$-sided dice, each possibility (1,2,3,4,5,6) has the same probability $p_i = 1/6$. When every possibility has the same probability, we get $1/p_i = N$ and $k_i = \log_2(N)$ as on the previous slide.

- When two dice are rolled, different sums occur with different probabilities...

i	p_i	$k_i = \log_2(1/p_i)$
2	1/36	5.17
3	2/36	4.17
4	3/36	3.59
5	4/36	3.17
6	5/36	2.85
7	6/36	2.59
8	5/36	2.85
9	4/36	3.17
10	3/36	3.59
11	2/36	4.17
12	1/36	5.17

Entropy

The entropy H of a source is equal to the expected (i.e. average) information content of its messages.

$$H = \sum_{i=1}^{N} p_i k_i = \sum_{i=1}^{N} p_i \log_2(1/p_i)$$

The entropy of a six-sided dice is $H = 2.59$ bits.

The entropy of the sum of two-six sided dice is $H = 3.27$ bits.

Note that this is less than the entropy of two six-sided dice, which is $H = 2 \times 2.59 = 5.18$ bits.

It makes sense for the sum of the two dice rolls to contain less information than the dice rolls separately. This is because the sum could be calculated if you knew the two dice rolls, but the two dice rolls couldn't always be determined if you only knew the sum.

It is impossible to losslessly compress messages from a particular source using an average number of bits that is less than the entropy.

Fixed length coding

- Similar to Pulse Coded Modulation (PCM), fixed length coding represents each of the N message possibilities i with a different codeword $\mathbf{c}_i$.
- In order for the codewords to be unique, they need to have a length of $L = \lceil \log_2(N) \rceil$.
- The coding efficiency is given by $R = H/L$ and can never be greater than 1.

For the sum of two dice example...

i	p_i	$k_i = \log_2(1/p_i)$	$\mathbf{c}_i$
2	$1/36$	5.17	1110
3	$2/36$	4.17	1101
4	$3/36$	3.59	1100
5	$4/36$	3.17	1000
6	$5/36$	2.85	0100
7	$6/36$	2.59	0000
8	$5/36$	2.85	0010
9	$4/36$	3.17	0001
10	$3/36$	3.59	0011
11	$2/36$	4.17	1011
12	$1/36$	5.17	0111

$H = 3.27$, $L = 4$, $R = 0.82$.

The sequence of sums [2,7,4,8,7,8,3,7,12] is encoded as the sequence of 36 bits 111000001100001000000010110100000111.

Variable length encoding

- Variable length coding represents each of the message possibilities i with different codewords $\mathbf{c}_i$ having various lengths l_i.
- No codeword is allowed to be a prefix of any other.
- The average codeword length is given by $L = \sum_{i=1}^{N} p_i l_i$.
- The coding efficiency is given by $R = H/L$ and can never be greater than 1.

For the sum of two dice example...

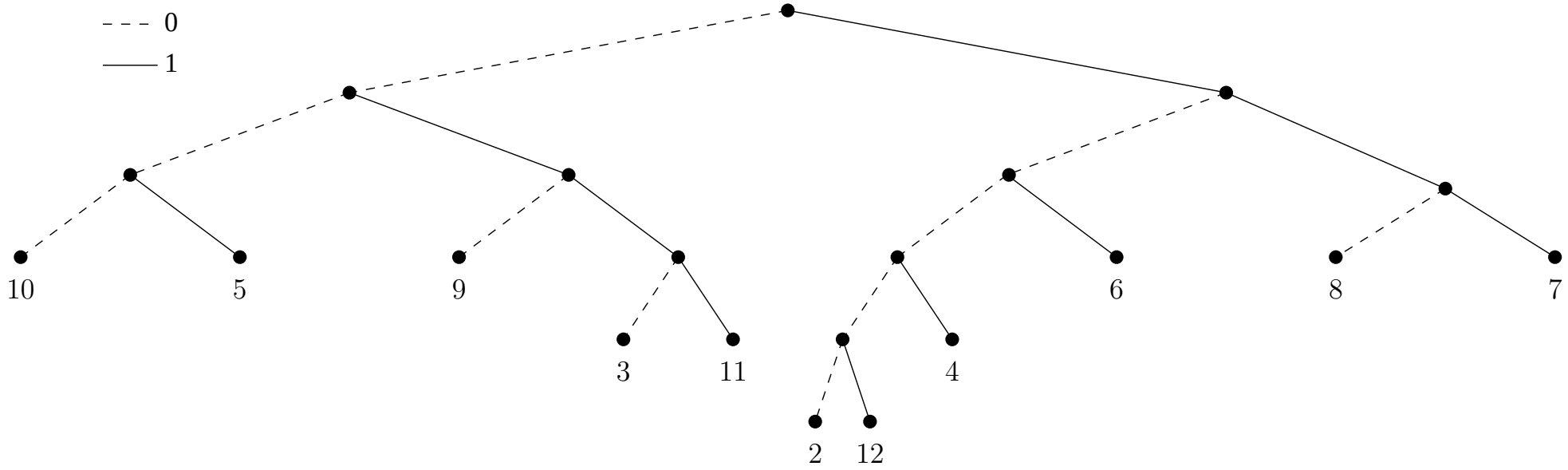
i	p_i	$k_i = \log_2(1/p_i)$	$\mathbf{c}_i$	l_i
2	1/36	5.17	10000	5
3	2/36	4.17	0110	4
4	3/36	3.59	1001	4
5	4/36	3.17	001	3
6	5/36	2.85	101	3
7	6/36	2.59	111	3
8	5/36	2.85	110	3
9	4/36	3.17	010	3
10	3/36	3.59	000	3
11	2/36	4.17	0111	4
12	1/36	5.17	10001	5

$H = 3.27$, $L = 3.31$, $R = 0.99$.

The sequence of sums [2,7,4,8,7,8,3,7,12] is encoded as the sequence of 33 bits 100001111001110111110011011110001.

Variable length decoding

- The codebook on the previous slide can be represented by this binary tree.



- The sequence of bits 10000111100111011111001101111 0001 can be decoded by repeatedly traversing the binary tree from the root node.
- When a leaf node is reached, a sum of two dice is identified and we start at the root node again.
- We get the same sequence of sums [2,7,4,8,7,8,3,7,12] that we started with.

Huffman coding

- The example on the previous slides is a Huffman code, which is a special type of variable length code.
- Huffman codewords are specially allocated in order to minimise the average codeword length L and therefore maximise the coding efficiency R.
- Short codewords are used to represent frequently occurring message possibilities and long codewords are used to represent rare ones.
- More specifically, the message possibility i is allocated a codeword $\mathbf{c}_i$ having an integer-valued length l_i that is typically close to the possibility's information content k_i.

Design of Huffman codes

Start with an empty codeword for each symbol value.

At each step:

- sort the symbol values (or groups of symbol values) in order of decreasing probability
- select the two (groups of) symbol values having the lowest probability
- prepend a 1-valued bit to the codeword(s) of one of these two (groups of) symbol values
- prepend a 0-valued bit to the codeword(s) of the other of these two (groups of) symbol values
- merge these two (groups of) symbol values into a single group of symbol values

Continue until we have a single group containing all symbol values.

Design of Huffman codes (continued)

<---- Decreasing probability

Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9	Step 10
7 (6/36)	7 (6/36)	7 (6/36)	7 (6/36)	5,10 (7/36)	3,9,11 (8/36)	2,4,6,12 (10/36)	7,8 (11/36)	3,5,9,10,11 (15/36)	2,4,6,7,8,12 (21/36)
8 (5/36)	8 (5/36)	8 (5/36)	8 (5/36)	7 (6/36)	5,10 (7/36)	3,9,11 (8/36)	2,4,6,12 (10/36)	7,8 (11/36)	3,5,9,10,11 (15/36)
6 (5/36)	6 (5/36)	6 (5/36)	6 (5/36)	8 (5/36)	7 (6/36)	5,10 (7/36)	3,9,11 (8/36)	2,4,6,12 (10/36)	
9 (4/36)	9 (4/36)	3,11 (4/36)	2,4,12 (5/36)	6 (5/36)	8 (5/36)	7 (6/36)	5,10 (7/36)		
5 (4/36)	5 (4/36)	9 (4/36)	3,11 (4/36)	2,4,12 (5/36)	6 (5/36)	8 (5/36)			
10 (3/36)	10 (3/36)	5 (4/36)	9 (4/36)	3,11 (4/36)	2,4,12 (5/36)				
4 (3/36)	4 (3/36)	10 (3/36)	5 (4/36)	9 (4/36)					
11 (2/36)	2,12 (2/36)	4 (3/36)	10 (3/36)						
3 (2/36)	11 (2/36)	2,12 (2/36)							
12 (1/36)	3 (2/36)								
2 (1/36)									

Symbol value	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9	Step 10
2	0	0	00	00	00	000	000	000	0000	10000
3		0	0	0	10	10	10	110	110	0110
4			1	1	1	01	01	01	001	1001
5				1	1	1	1	01	01	001
6						1	1	1	01	101
7							1	1	11	111
8							0	0	10	110
9					0	0	0	10	10	010
10				0	0	0	0	00	00	000
11		1	1	1	11	11	11	111	111	0111
12	1	1	01	01	01	001	001	001	0001	10001

Arithmetic coding

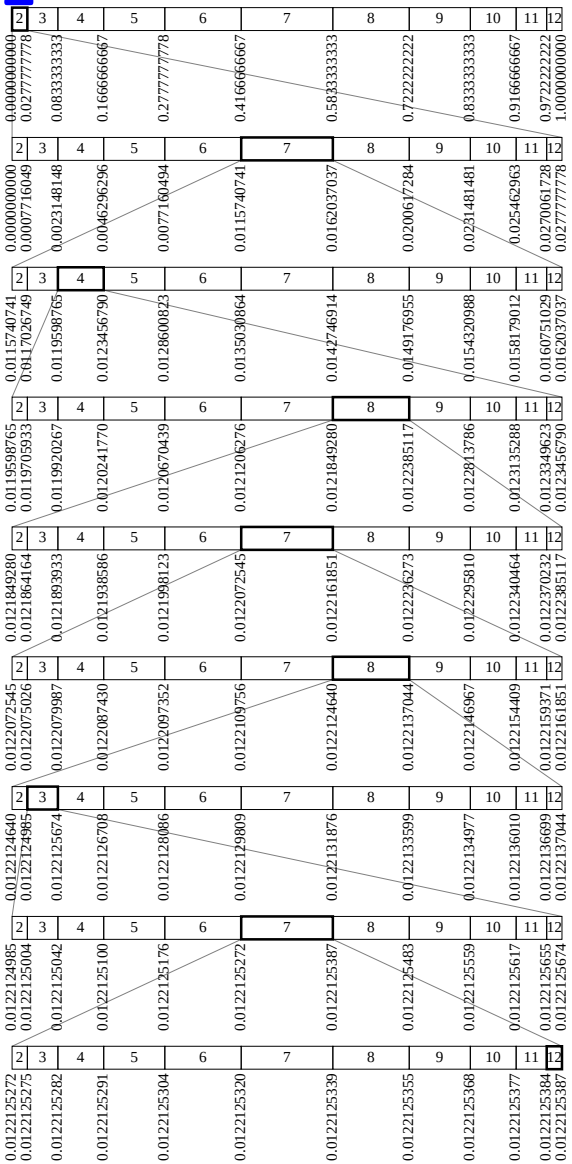
- The coding efficiency of Huffman coding is limited because it has to use an integer number of bits l_i to represent the message possibility i.
- It would have an efficiency of $R = 1$ if it could use k_i number of bits to represent the message possibility i.
- This motivates arithmetic coding, which represents a sequence of messages together, rather than individually.

Arithmetic encoding step 1

- Draw a number line that goes from 0 to 1.
- Divide the number line into N portions, where the i^{th} portion has a width equal to the probability of the i^{th} message possibility p_i.

Arithmetic encoding step 2

- Repeatedly select the portion of the number line that represents the next message in the sequence and divide that portion using the message probabilities.
- Once a portion has been selected for the last message in the sequence, a numerical range has been identified.
- The sequence of messages [2,7,4,8,7,8,3,7,12] gives the number range 0.0122125384 to 0.0122125387.



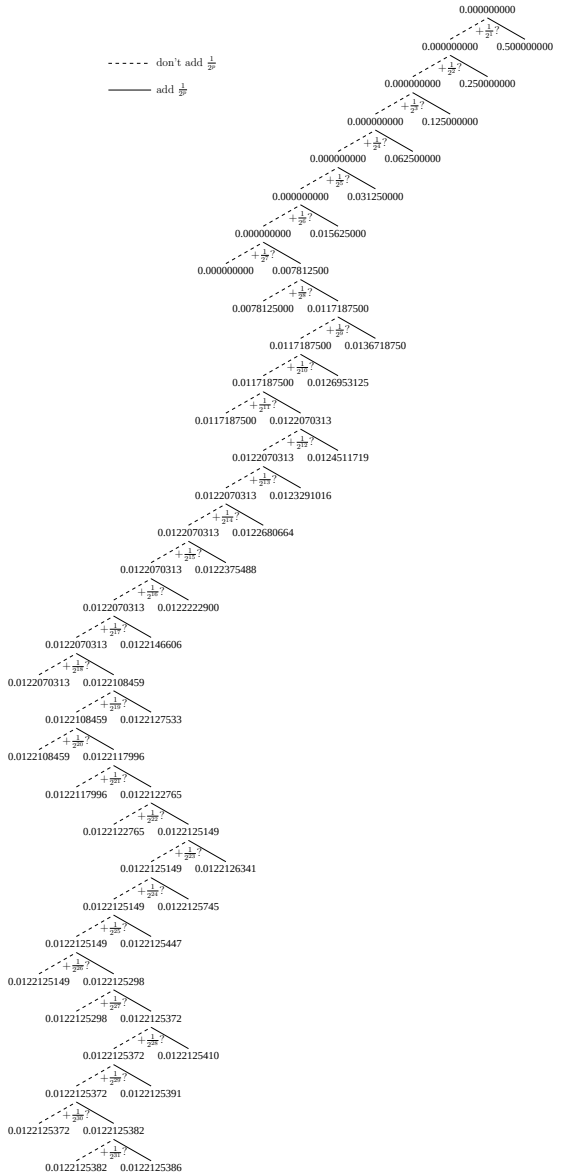
Arithmetic encoding step 3

- Find the shortest binary fraction that represents a number in the identified decimal range.
- `0.00000011001000000101111000110011` is the shortest binary fraction in the decimal range 0.0122125384 to 0.0122125387.
- This gives $\frac{1}{2^7}+\frac{1}{2^8}+\frac{1}{2^{11}}+\frac{1}{2^{18}}+\frac{1}{2^{20}}+\frac{1}{2^{21}}+\frac{1}{2^{22}}+\frac{1}{2^{26}}+\frac{1}{2^{27}}+\frac{1}{2^{30}}+\frac{1}{2^{31}} = 0.0122125386.$
- Since the binary fraction will always start with "`0.`" we only transmit the bits after the binary point.
- The sequence of messages [2,7,4,8,7,8,3,7,12] is represented by 31 bits using an arithmetic code, 33 bits using a Huffman code and 36 bits using a fixed length code.

Arithmetic encoding step 3 cont

We can determine the shortest binary fraction within the desired range by building a binary tree as follows.
Starting at the root node:

- if the result on the right-hand branch is within the desired range then output a bit value of 1 and stop,
- else if the result on the right-hand branch is below the desired range then output a bit value of 1 and follow the right-hand branch,
- else output a bit value of 0 and follow the left-hand branch.



Arithmetic decoding

Step 1 Convert the received binary sequence into a decimal fraction.

Step 2 Divide a number line from 0 to 1 into portions according to the message probabilities.

Step 3 Repeatedly select the portion having the range into which the decimal fraction falls and divide the portion according to the message probabilities. Output the messages that correspond to the selected portions and stop once the required number of messages have been output (this assumes that the required number is known to the receiver).

Exercise

Four friends, Hamilton, Button, Schumacher and Alonso have a race every fortnight. Hamilton tends to win most often and Alonso tends to win least often, as specified by the probability p_i provided for each racer i in the table below. Some source coding schemes have been devised to transmit the victor of a sequence of races.

i	p_i	$\mathbf{c}_i^{\mathrm{FLC}}$	$\mathbf{c}_i^{\mathrm{Huff}}$
Hamilton	0.5	00	0
Button	0.25	01	10
Schumacher	0.125	10	110
Alonso	0.125	11	111

1. Determine the amount of information k_i that is conveyed by messages saying that each racer i has won.

2. Determine the entropy H of messages saying who has won.

3. Determine the coding efficiencies R associated with the fixed length codewords $\mathbf{c}_i^{\mathrm{FLC}}$ and the Huffman codewords $\mathbf{c}_i^{\mathrm{Huff}}$.

Exercise continued

4. Why does the Huffman code perform so well in this case?

5. Determine the bit sequences that result when the sequence of victors [Schumacher,Hamilton,Button,Hamilton] is represented using the fixed length codewords $\mathbf{c}_i^{\mathrm{FLC}}$, the Huffman codewords $\mathbf{c}_i^{\mathrm{Huff}}$ and an arithmetic code.

6. Determine the sequence of victors that is represented by the bit sequence 00110001, which was obtained using the fixed length codewords $\mathbf{c}_i^{\mathrm{FLC}}$.

7. Draw a binary tree for the Huffman codewords $\mathbf{c}_i^{\mathrm{Huff}}$ and use it to determine the sequence of victors that is represented by the bit sequence 11111000.

8. Determine the sequence of four victors that is represented by the bit sequence 010111, which was obtained using an arithmetic code.