HUFFMAN TREES AND CODES

Huffman Coding

- Huffman coding is a way of encoding characters into a sequence of 0s and 1s.

- Huffman coding generates a set of codes based on the frequency of occurrence of the characters. When the data is coded using these characters, the resulting encoding has a shorter length compared to an encoding that uses 8 bits to encode the data.

- The Problem

Encoding symbols using bits: Encoding scheme that takes text written in alphabets & converts this text into long string of bits.

Ex: 26 letters in English + space + ! + . + ? +’ = 32 symbols

i.e 2 pow b sequences. So b =5.

Ie each alphabet will be 5 bits long.

We couldn’t ask to encode each symbol using 4 bits.

Letters do not get used equally frequently.

- Letters e, t, a, o, l, n get used much more frequently than q, j, x, z.

- So it's a waste to translate them all into same number of bits. Instead use small number of bits for frequent letters and large number of bits for less frequent ones and end up using fewer than 5 bits per letter when we average over long string of typical text.

- Variable length encoding scheme

- Before internet, radio, telephone, digital computer
- Telegraph was used
- It is capable of transmitting pulses down a wire. So to send message encode the text into sequence of pulses (Morse Code)
- Translate each letter into dots and dashes.
- Encode frequent letters with short strings
- Asked printing press people to get frequency estimates for letters in English.

- Morse code maps e to 0 (DOT) t to 1 (DASH) a to 01.

- In general morse code maps more frequent letters to shorter bit strings.

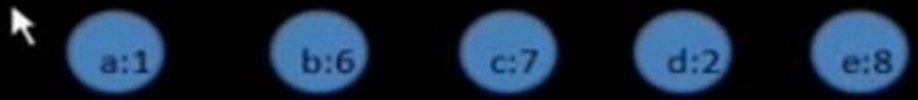
- Morse code is applied on letters/Alphabets and not on Words
- ex eta 0101 aa, eta, etet or ate
- To handle this morse code transmission involved shorter pauses between letters.

- ex aa .- pause .-
- But it actually means that we haven't encoded the everything usinfg only bits 0,1 and pause.

- So if we want to encode everything using only the bits 0 and 1 we need to have some further encoding in which pause got mapped to bits.

Consider this example:

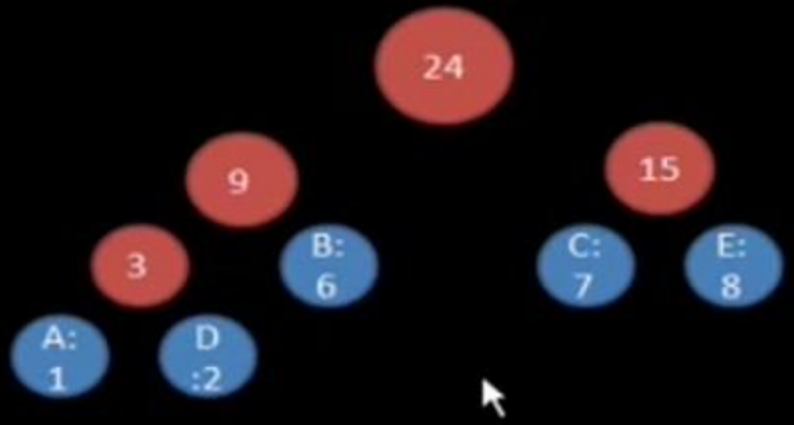
char	frequency
a	1
b	6
c	7
d	2
e	8



(imagine these as 5 separate 'trees'. Combine the two smallest trees in order)

Consider this example:

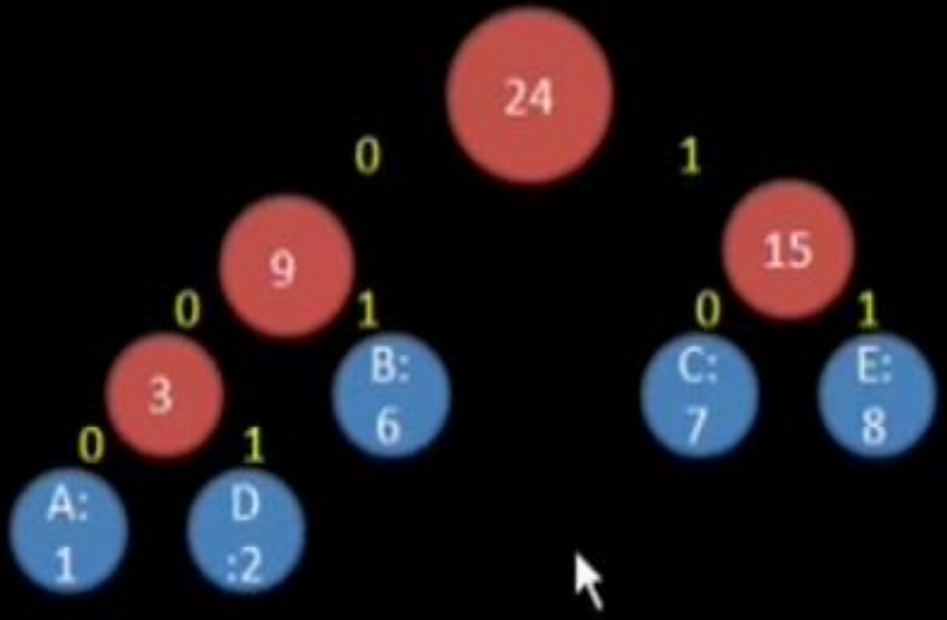
char	frequency
a	1
b	6
c	7
d	2
e	8



Ok, now we have a large tree containing all the characters. We can now assign binary code to each symbol by going down the tree (each left branch receives a '0', each right branch receives a '1')

Consider this example:

char	frequency
a	1
b	6
c	7
d	2
e	8



a = 000, b = 01, c = 10, d = 001, e = 11

These are what a,b,c,d,e will each be converted to.

Examples:

- a = 000, b = 01, c = 10, d = 001, e = 11

Encode 'abcbe' using our results using huffman coding:

abcbe = 000 01 10 01 11 = 00001100111 (simple enough)

Decode '1011001000011101' :

(compare the representations above to the binary code bit by bit to fill the only possible result)

1011001000011101 = c 1001000011101 (only c starts with 1, then 0)
= c e 001000011101 (only e starts with 1, then 1)
= c e d 000011101 (only d starts with 0, then 0, then 1)
= c e d a 011101 (only a starts with 0, then 0, then 0)
= c e d a b 1101 (only b starts with 0, then 1)
= c e d a b e 01 (only e starts with 1, then 1)
= c e d a b e b = 'cedabeb'

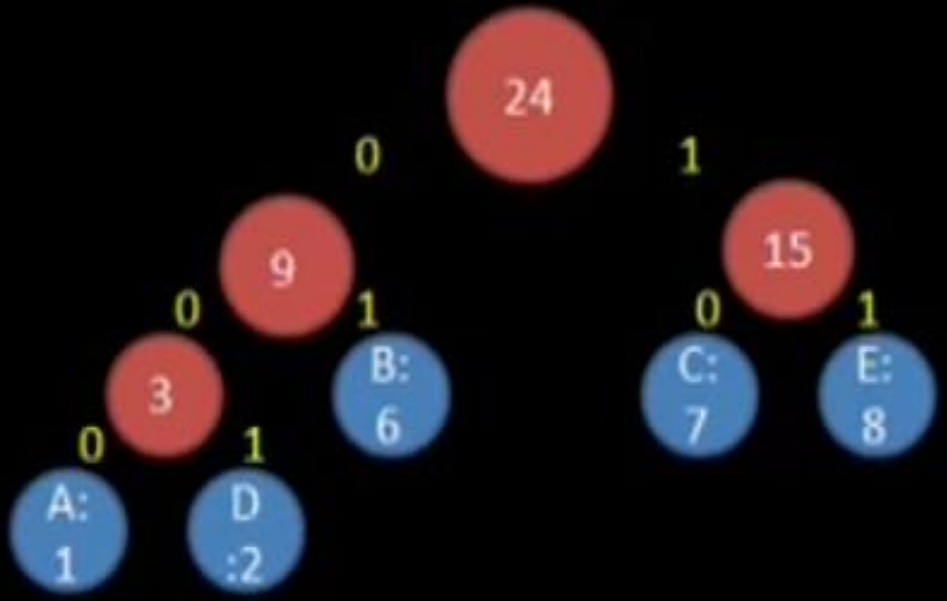
Average Length:

- Average code word length = 1/f(T) * sum of d(i) * f(i), for i = 0 to i = n,

  where n = number of characters, f(T) = total frequency, f(i) = frequency of that character, d(i) = length of that symbol.

Consider this example:

char	frequency
a	1
b	6
c	7
d	2
e	8



a = 000, b = 01, c = 10, d = 001, e = 11
These are what a,b,c,d,e will each be converted to.

AvgLength = (1/(1+6+7+2+8)) * (3*1 + 2*6+ 2*7 + 3*2 + 2*8)
= (1/24) * (3 + 12 + 14 + 6 + 16)
= (1/24) * (51)
= 2.125 digits long (the average letter in the code is encoded to 2.125 binary digits)

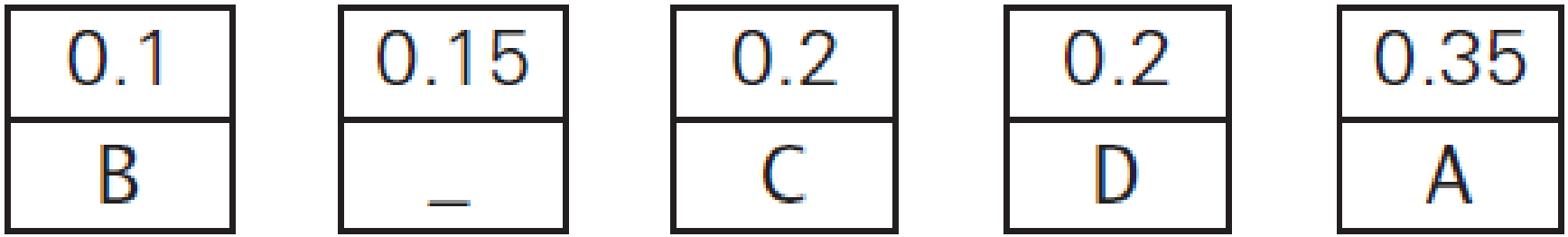
HUFFMAN TREES AND CODES

- Consider the five-symbol alphabet {A, B, C, D, _} with the following occurrence frequencies in a text made up of these symbols:

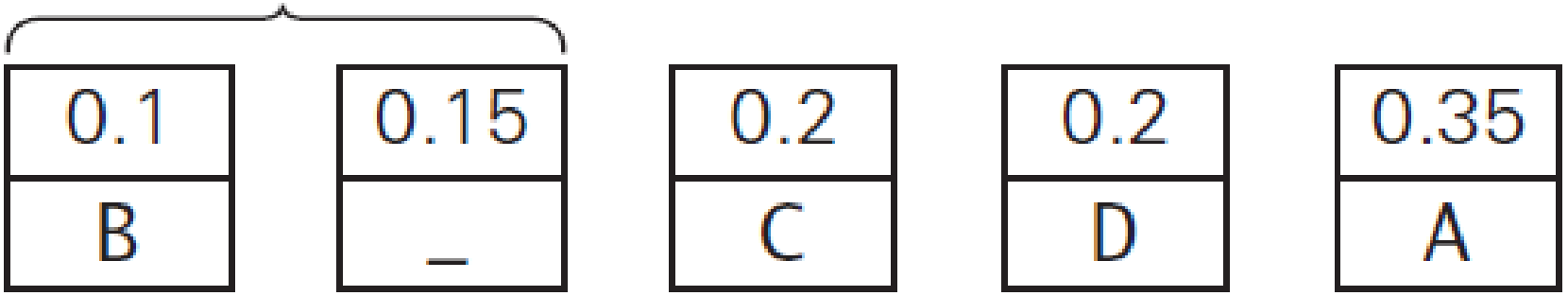
symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15

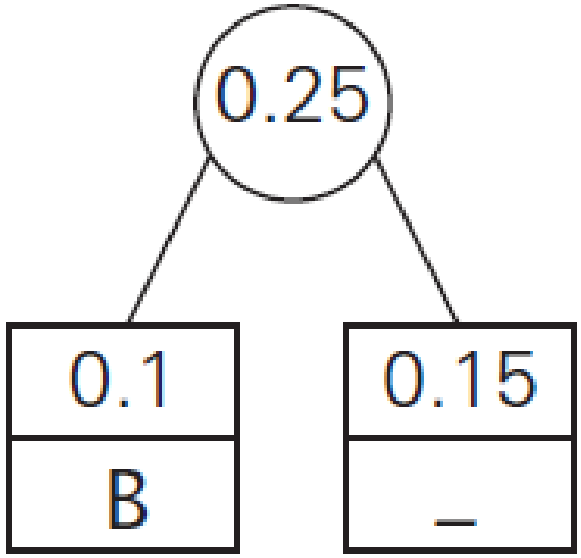
HUFFMAN TREES AND CODES

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15

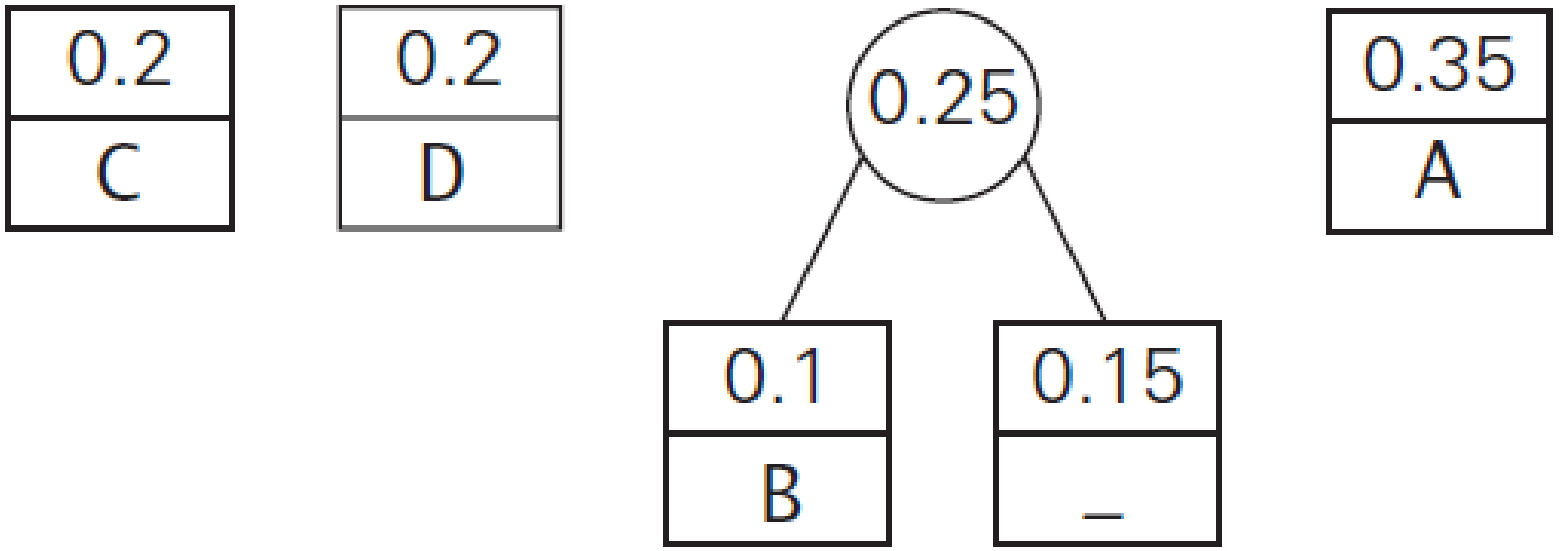


HUFFMAN TREES AND CODES

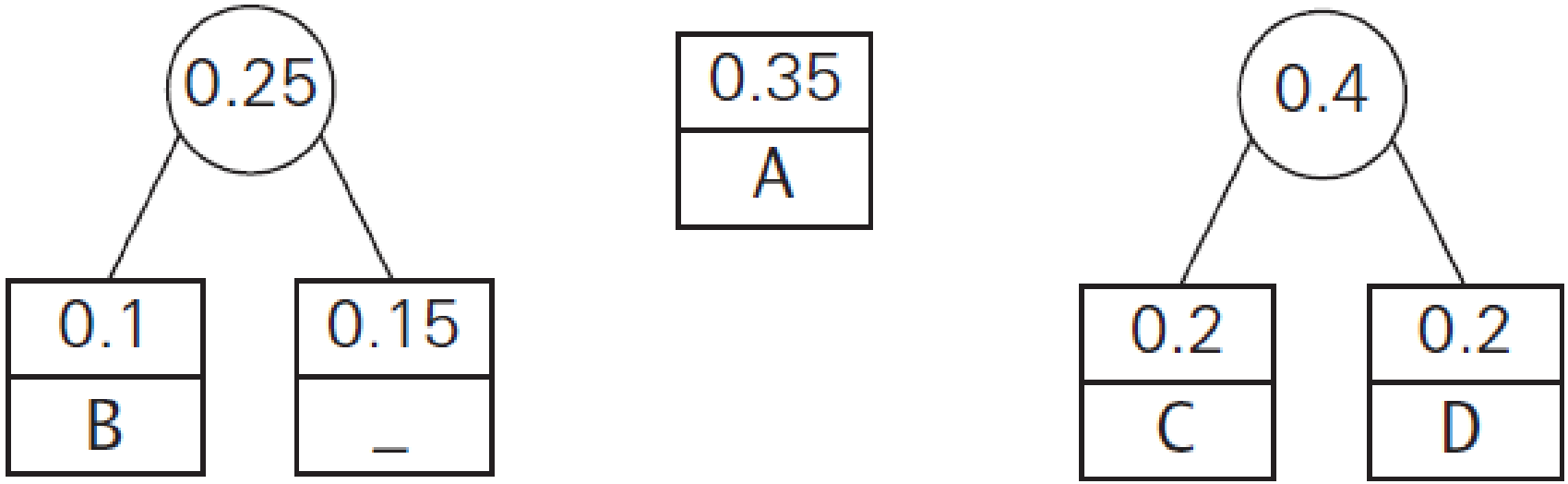




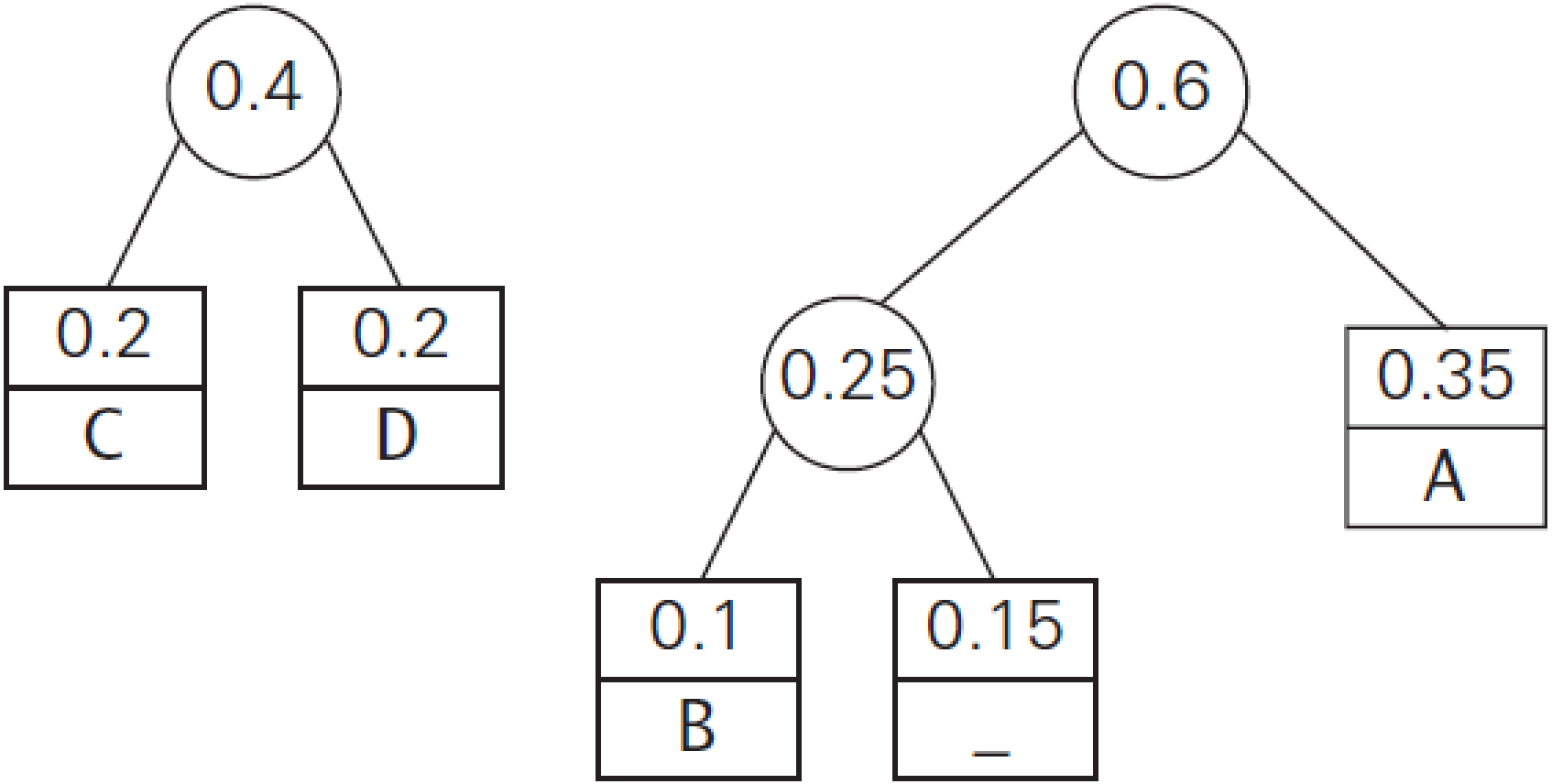
HUFFMAN TREES AND CODES



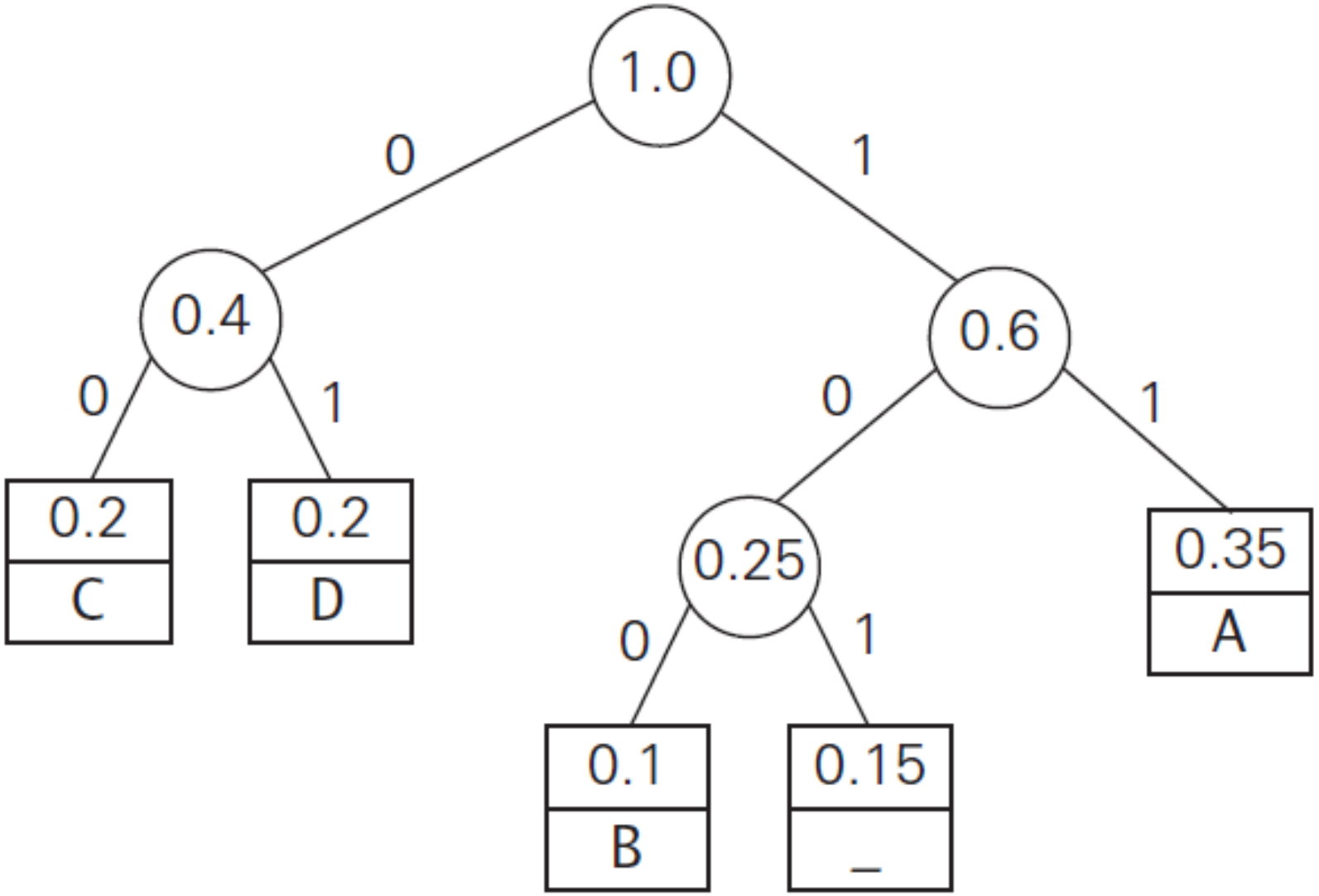
HUFFMAN TREES AND CODES



HUFFMAN TREES AND CODES



HUFFMAN TREES AND CODES



HUFFMAN TREES AND CODES

- The resulting codewords are as follows :

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15
codeword	11	100	00	01	101

HUFFMAN TREES AND CODES

- DAD is encoded as 011101
- 100110110111101 is decoded as BAD_AD

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15
codeword	11	100	00	01	101

HUFFMAN TREES AND CODES

- DAD is encoded as 011101
- 100110110111101 is decoded as BAD_AD

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15
codeword	11	100	00	01	101

HUFFMAN TREES AND CODES

- With the occurrence frequencies given and the codeword lengths obtained, the average number of bits per symbol in this code is
- 2*0.35 + 3*0.1+ 2*0.2 + 2*0.2 + 3*0.15
- = 2.25.

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15
codeword	11	100	00	01	101

THANK YOU