UNIT IV GREEDY TECHNIQUE

Greedy Technique – Minimum Spanning Tree – Prim‘s Algorithm – Kruskal’s Algorithm – Single-source-shortest-paths Problem – Dijkstra’s Algorithm – Huffman Coding – Fractional Knapsack Problem

Coding Problem

***Coding*: assignment of bit strings to alphabet characters**

E.g. We can code {a,b,c,d} as {00,01,10,11} or {0,10,110,111} or {0,01,10,101}.

***Codewords*: bit strings assigned for characters of alphabet**

Two types of codes:

- ***fixed-length encoding* (e.g., ASCII)**
- ***variable-length encoding* (e,g., Morse code)**

E.g. if P(a) = 0.4, P(b) = 0.3, P(c) = 0.2, P(d) = 0.1, then the average length of code #2 is 0.4 + 2*0.3 + 3*0.2 + 3*0.1 = 1.9 bits

***Prefix-free codes (or prefix-codes)*: no codeword is a prefix of another codeword**

It allows for efficient (online) decoding!

E.g. consider the encoded string (msg) 10010110…

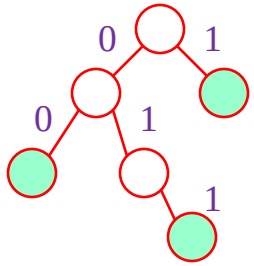
Problem: If frequencies of the character occurrences are known, what is the best binary prefix-free code?

The one with the shortest average code length. The average code length represents on the average how many bits are required to transmit or store a character.

Huffman codes

- Any binary tree with edges labeled with 0's and 1's yields a prefix-free code of characters assigned to its leaves

- Optimal binary tree minimizing the average length of a codeword can be constructed as follows:



represents {00, 011, 1}

Huffman's algorithm

Initialize *n* one-node trees with alphabet characters and the tree weights with their frequencies.

Repeat the following step *n*-1 times: join two binary trees with smallest weights into one (as left and right subtrees) and make its weight equal the sum of the weights of the two trees.

Mark edges leading to left and right subtrees with 0's and 1's, respectively.

Example: Huffman Coding

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

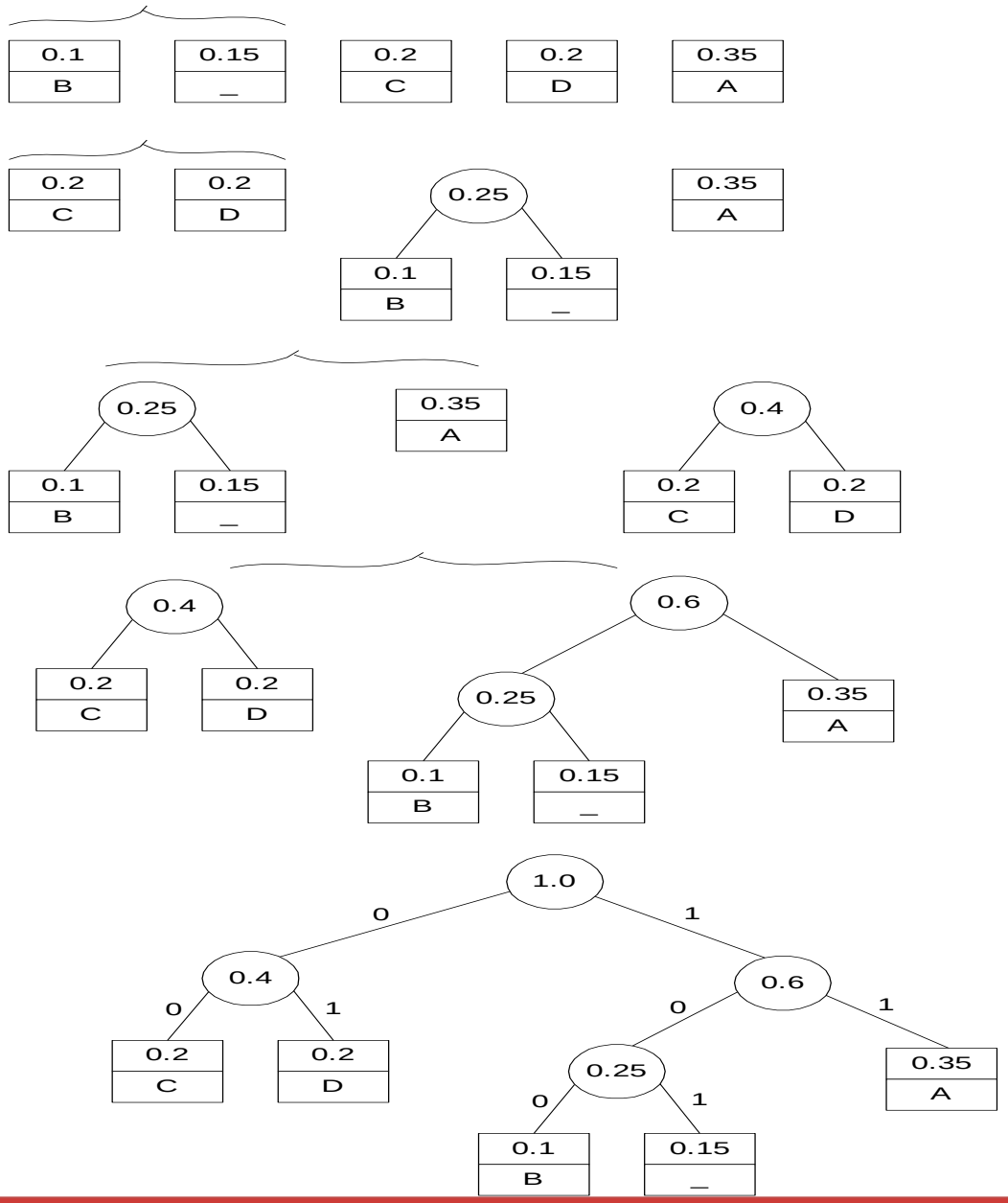
Codeword: **11 100 00 01 101**

average bits per character: 2.25

for fixed-length encoding: 3

compression ratio:

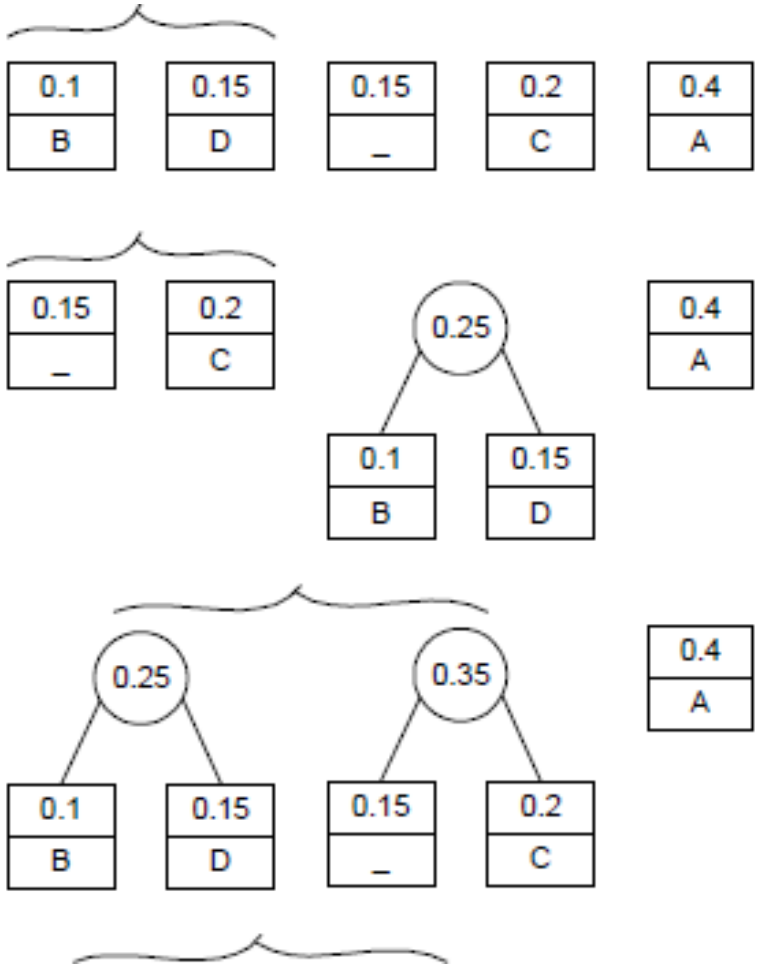
(3-2.25)/3*100% = 25%

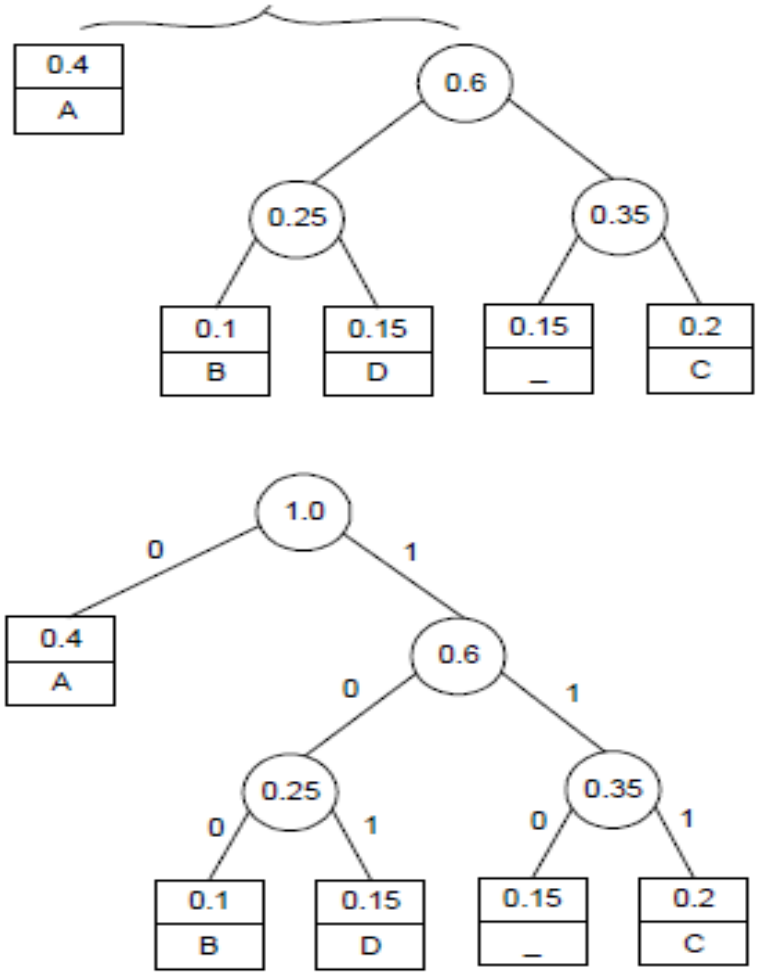


Construct a Huffman code for the following data:

character	A	B	C	D	_
probability	0.4	0.1	0.2	0.15	0.15

character	A	B	C	D	_
probability	0.4	0.1	0.2	0.15	0.15
codeword	0	100	111	101	110





character	A	B	C	D	_
probability	0.4	0.1	0.2	0.15	0.15
codeword	0	100	111	101	110

Encode the text ABACABAD using the code

The text ABACABAD will be encoded as 0100011101000101.

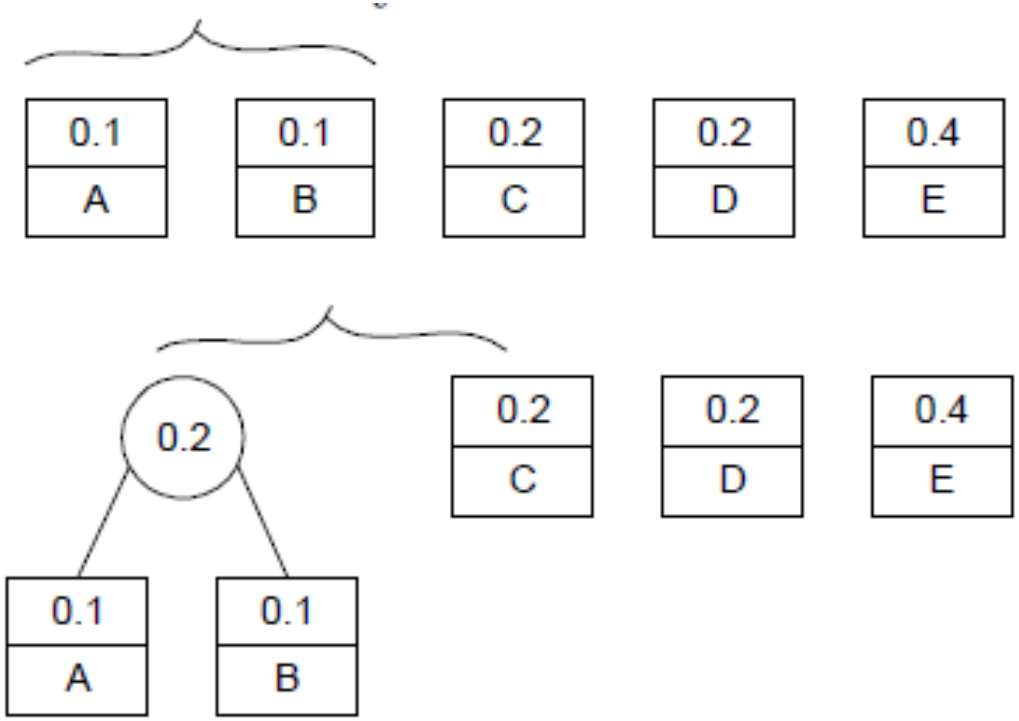
Decode the text whose encoding is 100010111001010

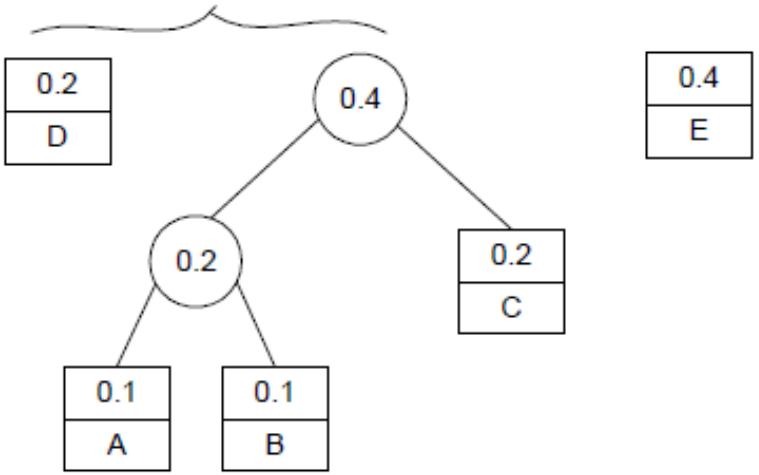
With the code of part a, 100010111001010 will be decoded as

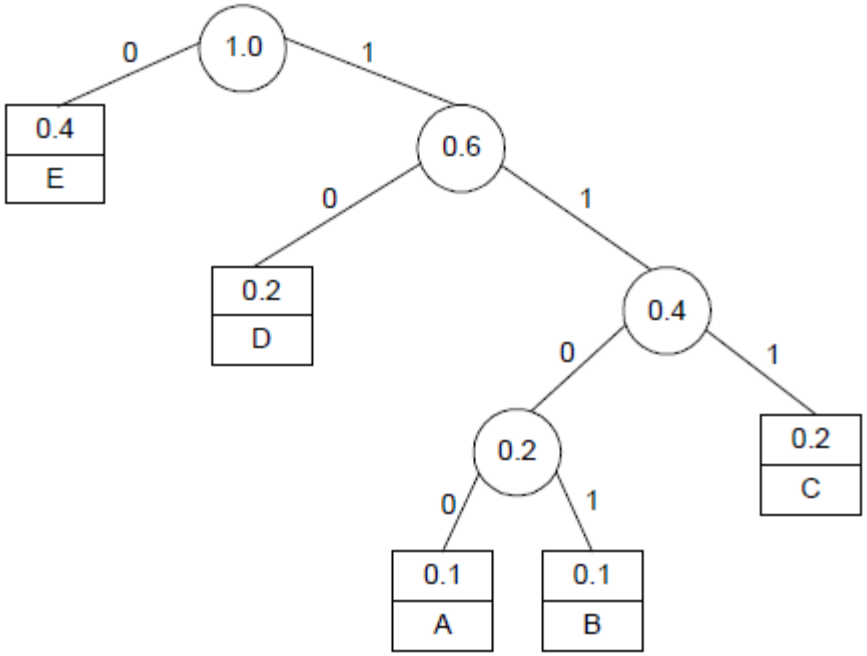
$$\underset{B}{100}|\underset{A}{0}|\underset{D}{101}|\underset{\_}{110}|\underset{A}{0}|\underset{D}{101}|\underset{A}{0}$$

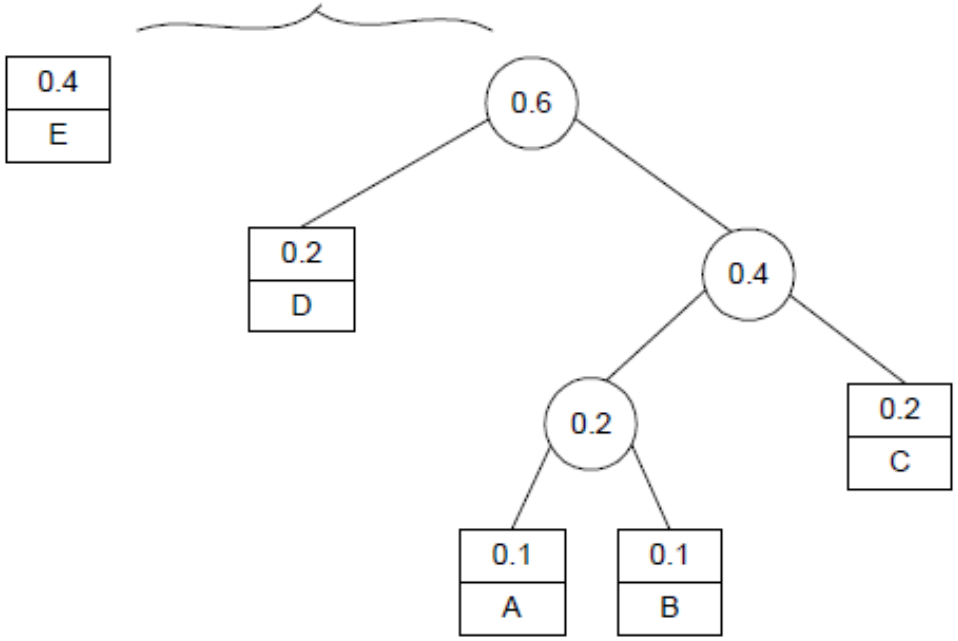
For data transmission purposes, it is often desirable to have a code with a minimum variance of the codeword lengths (among codes of the same average length). Compute the average and variance of the codeword length in two Huffman codes that result from a different tie breaking during a Huffman code construction for the following data:

character	A	B	C	D	E
probability	0.1	0.1	0.2	0.2	0.4









character	A	B	C	D	E
probability	0.1	0.1	0.2	0.2	0.4
codeword	1100	1101	111	10	0
length	4	4	3	2	1

character	A	B	C	D	E
probability	0.1	0.1	0.2	0.2	0.4
codeword	1100	1101	111	10	0
length	4	4	3	2	1

Thus, the mean and variance of the codeword's length are, respectively,

$$\bar{l} = \sum_{i=1}^{5} l_i p_i = 4 \cdot 0.1 + 4 \cdot 0.1 + 3 \cdot 0.2 + 2 \cdot 0.2 + 1 \cdot 0.4 = 2.2 \quad \text{and}$$

$$Var = \sum_{i=1}^{5} (l_i - \bar{l})^2 p_i = (4\text{-}2.2)^2 0.1 + (4\text{-}2.2)^2 0.1 + (3\text{-}2.2)^2 0.2 + (2\text{-}2.2)^2 0.2 + (1\text{-}2.2)^2 0.4 = 1.36.$$


Thank You