Digital Image Processing

Lecture #12

Ming-Sui (Amy) Lee

Course Information

- **Following Schedule**

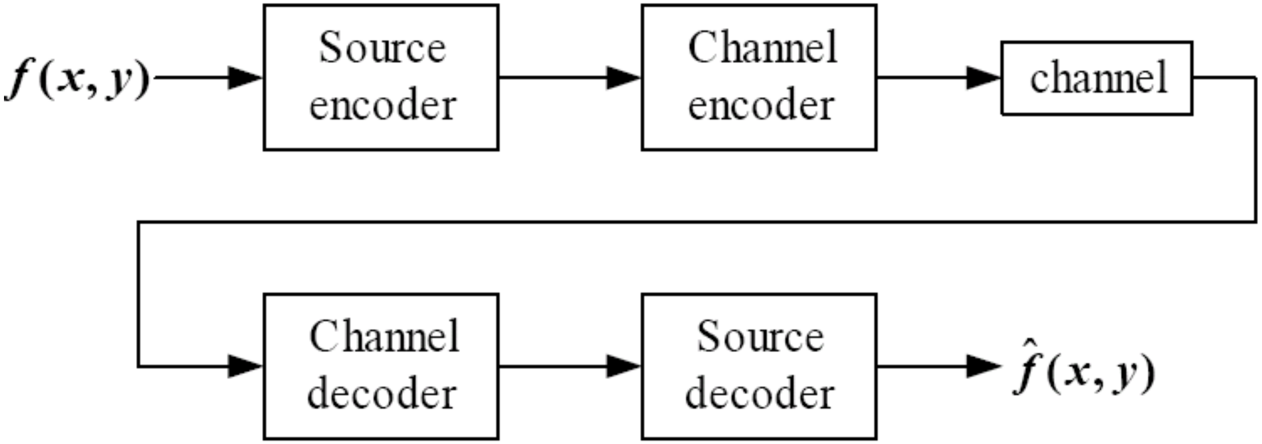
04/01	Lecture 5	05/20	Lecture 10
04/08	Lecture 6	05/27	Lecture 11
04/15	Lecture 7	06/03	Lecture 12
04/22	Midterm	06/10	Demo
04/29	Lecture 8	06/17	Demo
05/06	Proposal	06/24	Final Package Due
05/13	Lecture 9		

Announcement

- Final demo video
  - Due: 11:59 a.m. on Jun. 9, 2021
    - 10~12 minutes for each team
    - Video format: mpg, avi, mp4 or wmv
    - Provide a valid link on NTU COOL for us to download
    - DIP_Teamxx_FinalDemo.mpg/avi/mp4/wmv
  - Remember to include
    - Paper title / Motivation / Problem definition /
    - Algorithm / Experimental results
    - Reference

Compression Model

Compression Model



- **The encoder is composed of a source encoder and a channel encoder**
  - **Source encoder**

    remove input redundancies
  - **Channel encoder**

    increase the noise immunity of the source encoder's output

Compression Model

- **Source encoder**
  - reduce or eliminate any coding, interpixel, or psychovisual redundancies in an input image
- **Channel encoder**
  - If the channel between the encoder and decoder is noise free (not prone to error), the channel encoder and decoder are omitted
  - As the output of the source encoder contains little redundancy, it would be highly sensitive to transmission noise without the addition of the "controlled redundancy"

Compression Model

- Examples of source coding and channel coding
  - Source coding
    - Huffman code
    - Arithmetic code
  - Channel coding
    - Hamming code

Source Coding
- Huffman Code

Entropy

- From information theory, the average number of bits needed to encode the symbols in a source S is always bounded by the entropy of S

$$H = \sum p_i \cdot \log_2 \frac{1}{p_i}$$

  - E.g. $i = 2,\ p_1 = 1,\quad p_2 = 0 \qquad \Rightarrow H = 0$

    $p_1 = 1/2,\ p_2 = 1/2 \qquad \Rightarrow H = 1\ \ bits/symbol$

    $i = 4,\ p_1 = p_2 = p_3 = p_4 = 1/4 \quad \Rightarrow H = 2\ \ bits/symbol$

Source Coding

- **Huffman code**
  - **Given the statistical distribution of the gray levels**
  - **Generate a code that is as close as possible to the minimum bound, entropy**
  - **A variable length code**

Huffman Code

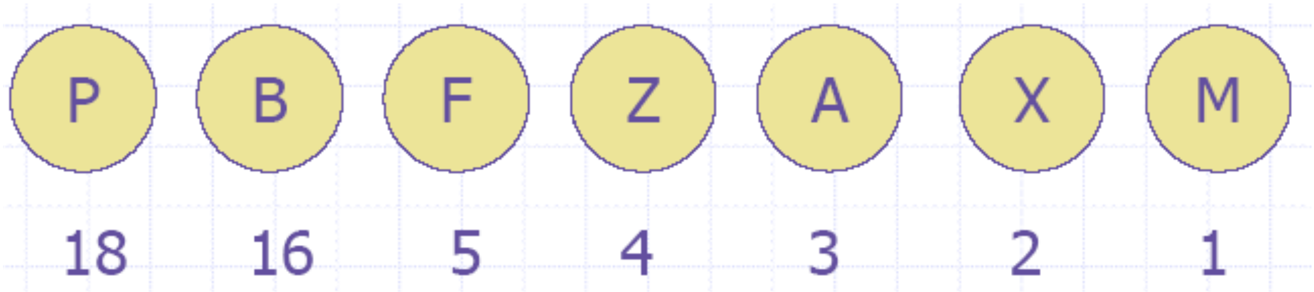
- **Five steps:**
  - Find the gray-level probabilities for the image by finding the histogram
  - Order the input probabilities (histogram magnitudes)from smallest to largest
  - Combine the smallest two by addition

    (if multiple → Not a unique code)

    → Repeat until one tree is formed
  - For each node, assign 0 to the left, 1 to the right
  - Traverse the tree from root to the leaf to generate the code

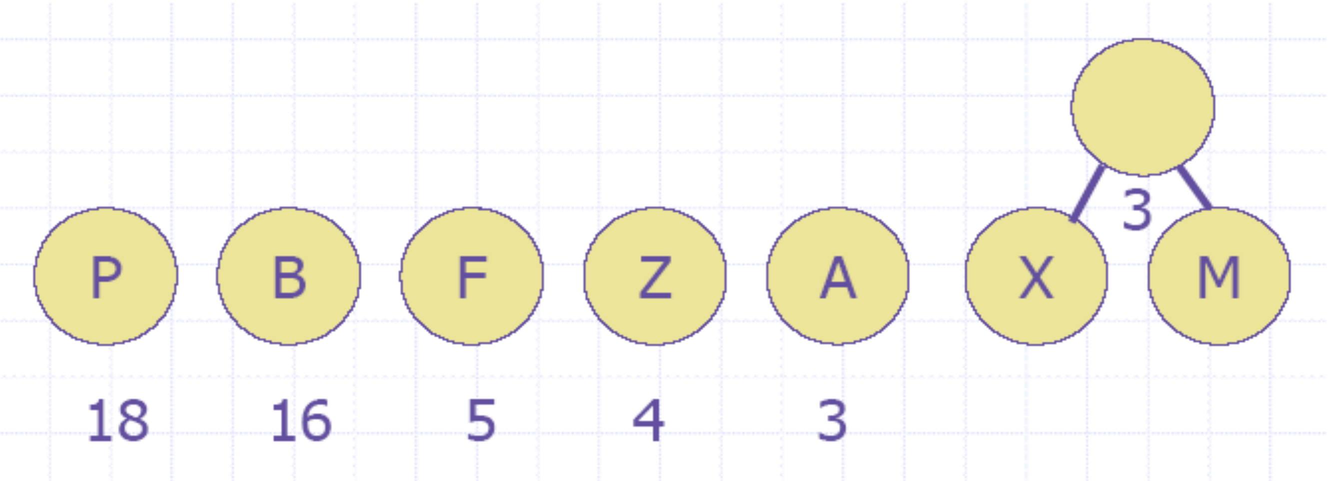
Huffman Code

- Example
  - Relative frequency of characters in a message text
  - A larger number indicates higher frequency of use
    - Higher frequency → shorter code
    - Lower frequency → longer code



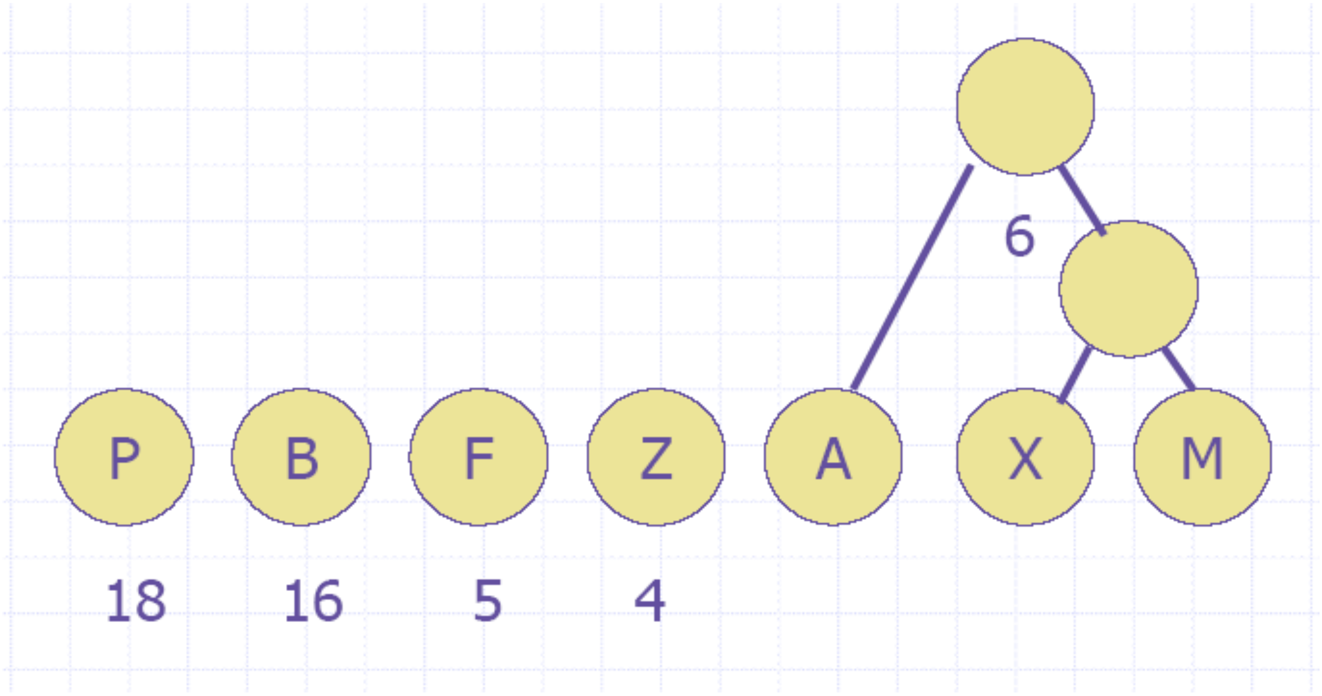
Huffman Code

- Group lowest sum up the frequencies



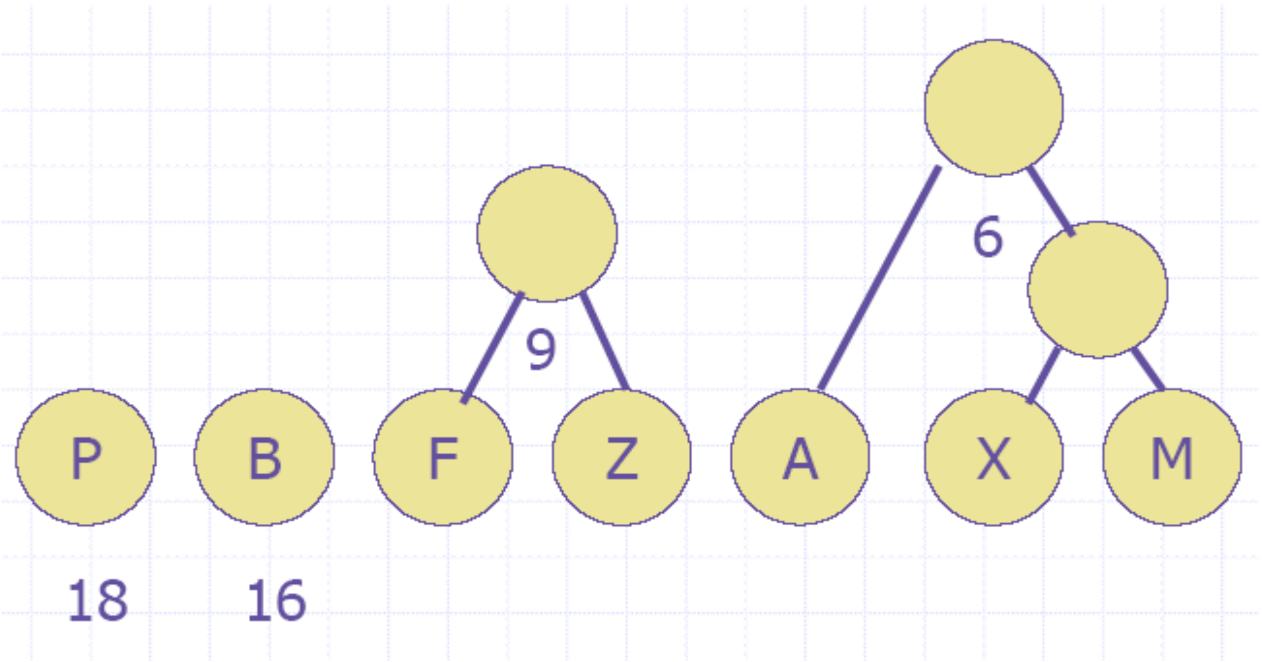
Huffman Code

- Group next two lowest



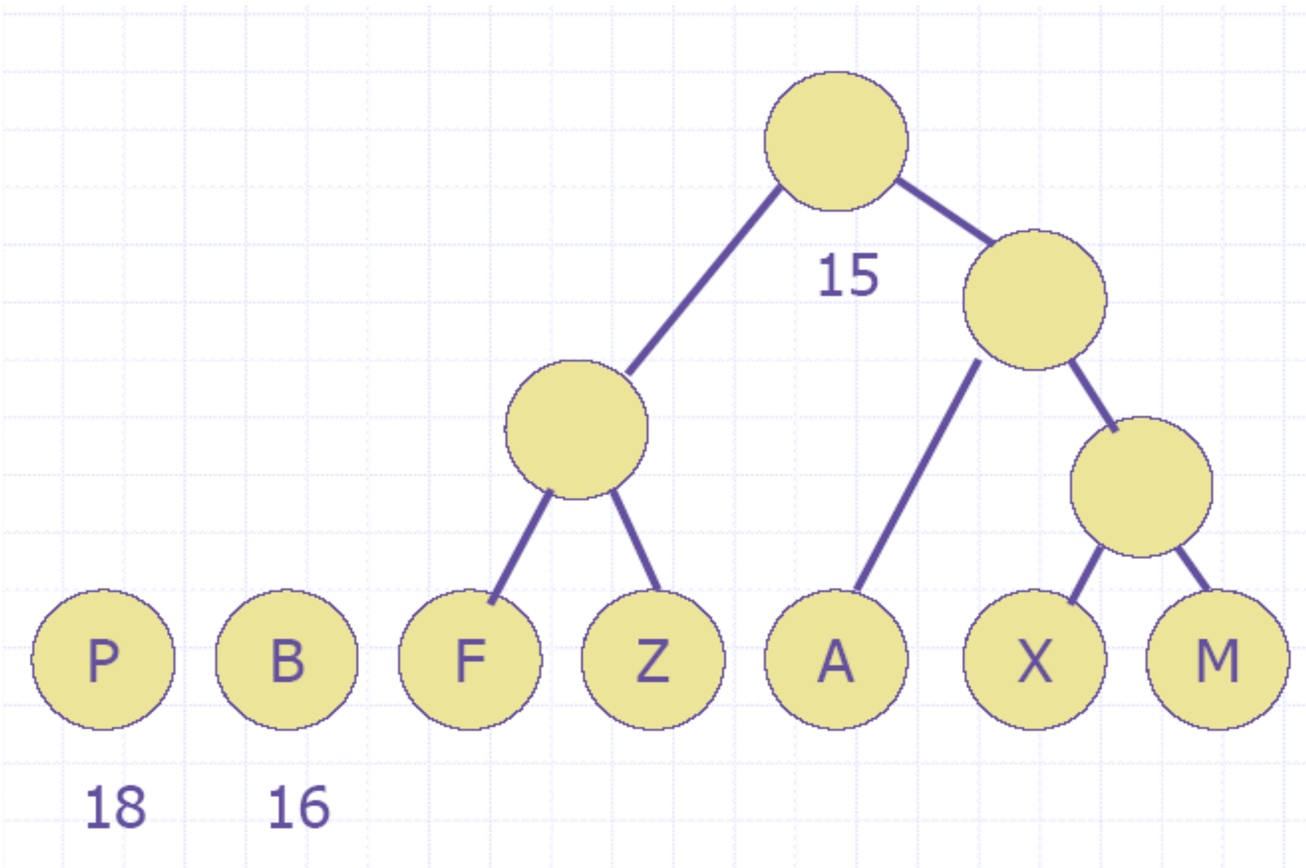
Huffman Code

- Group next two lowest



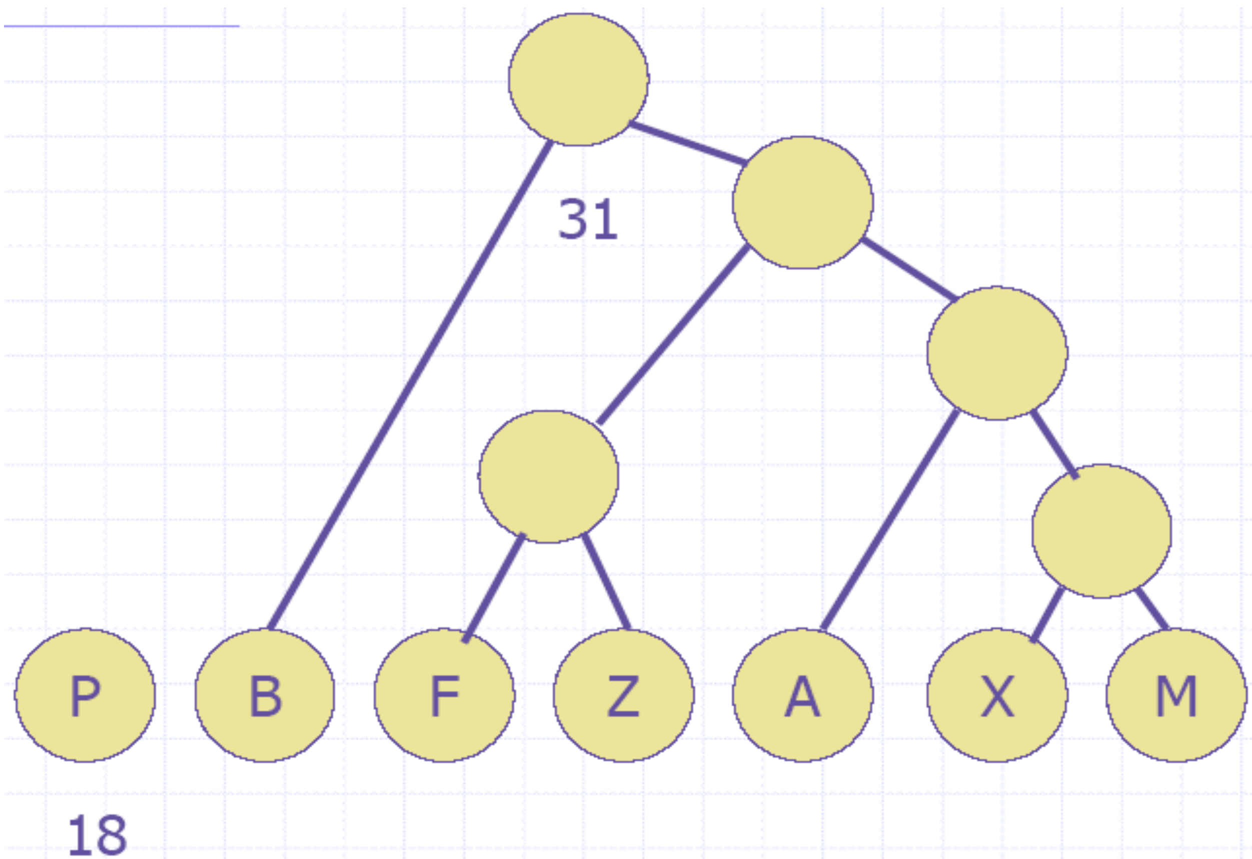
Huffman Code

- Group next two lowest



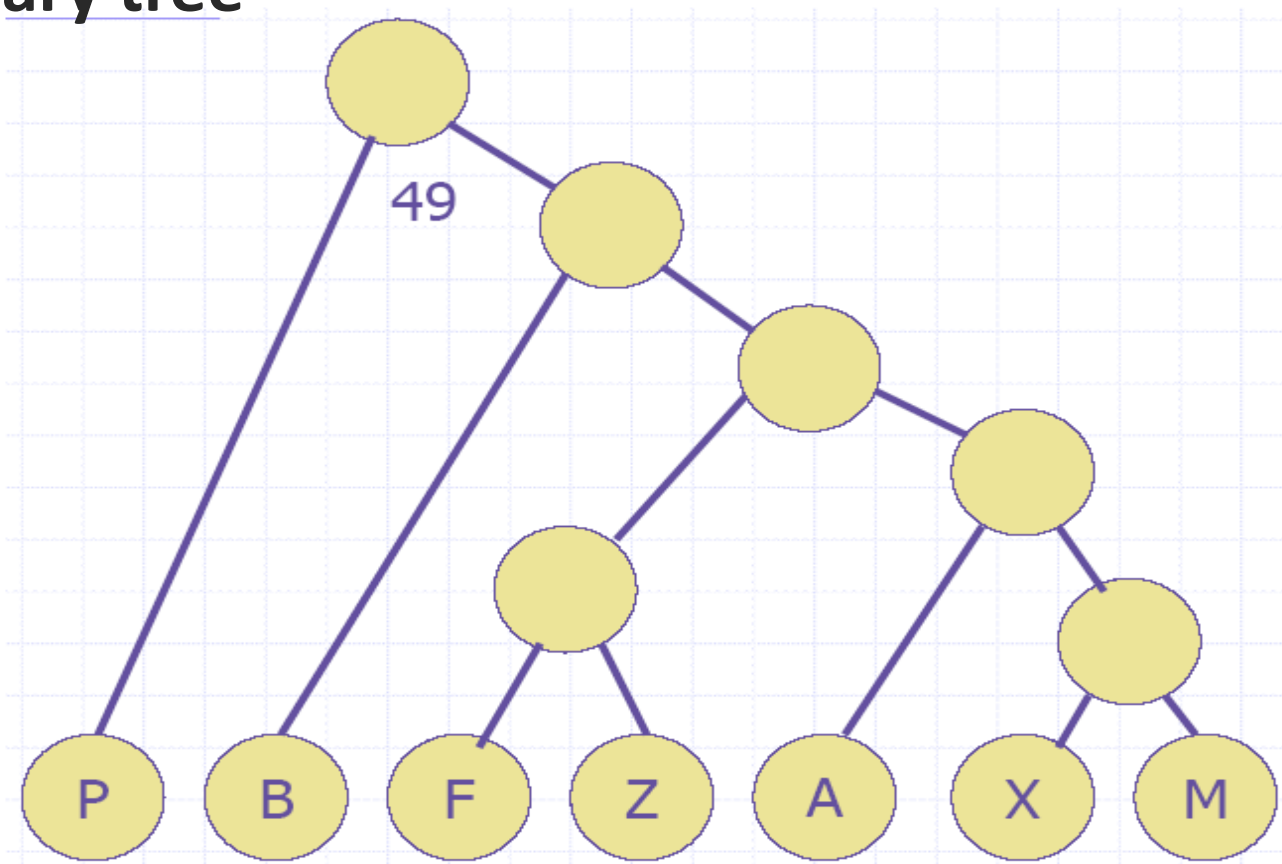
Huffman Code

- Group next two lowest



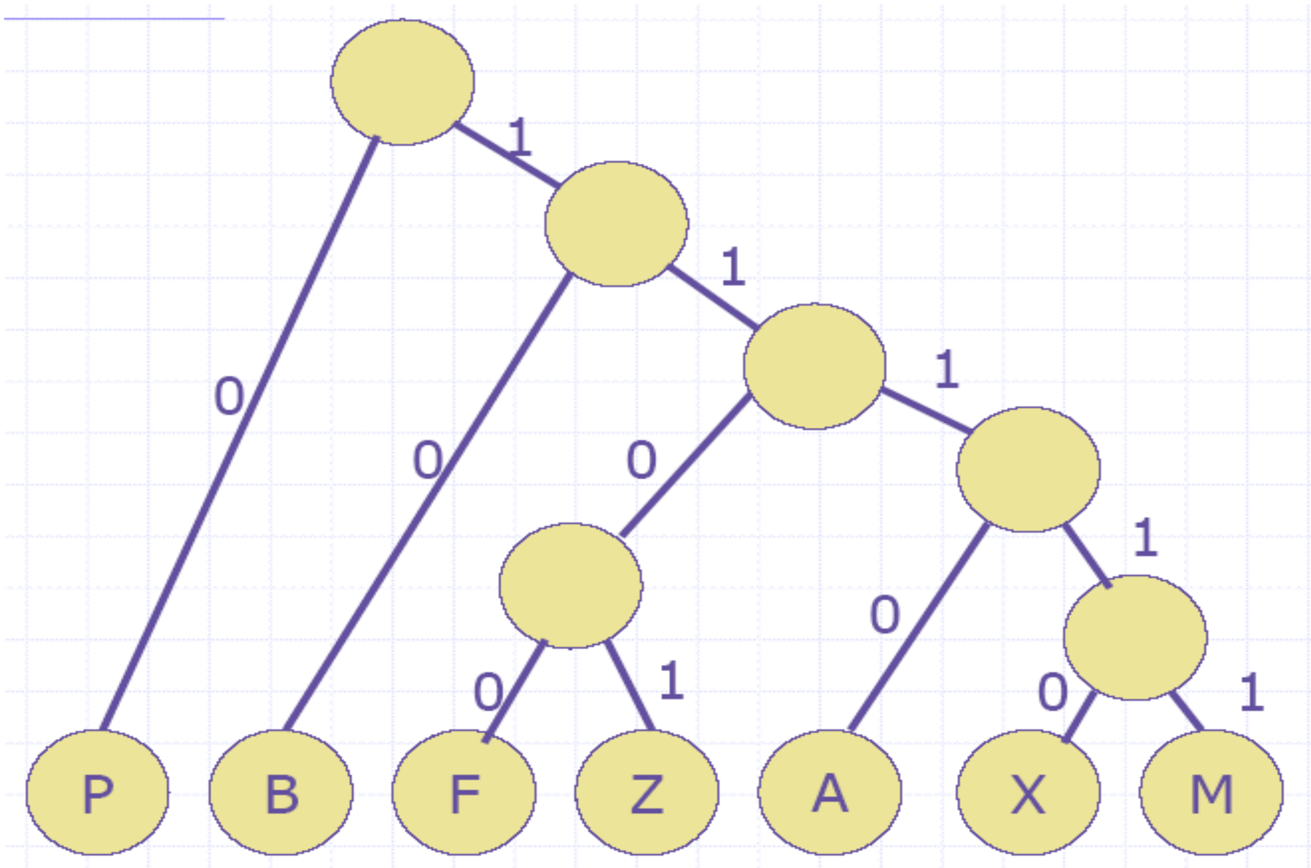
Huffman Code

- Finally all characters are leaf nodes of a single binary tree



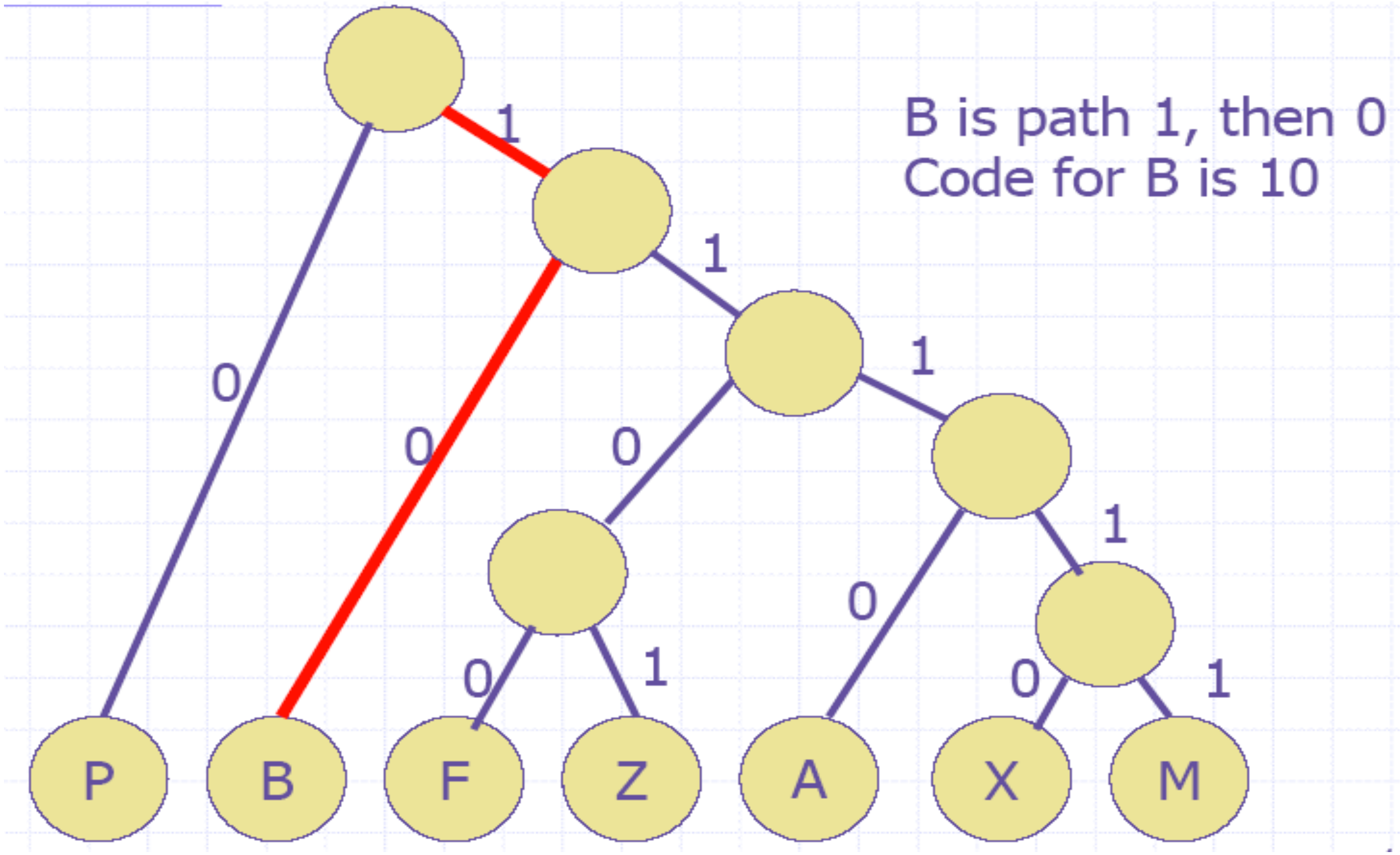
Huffman Code

- Assign 0 to left and 1 to right



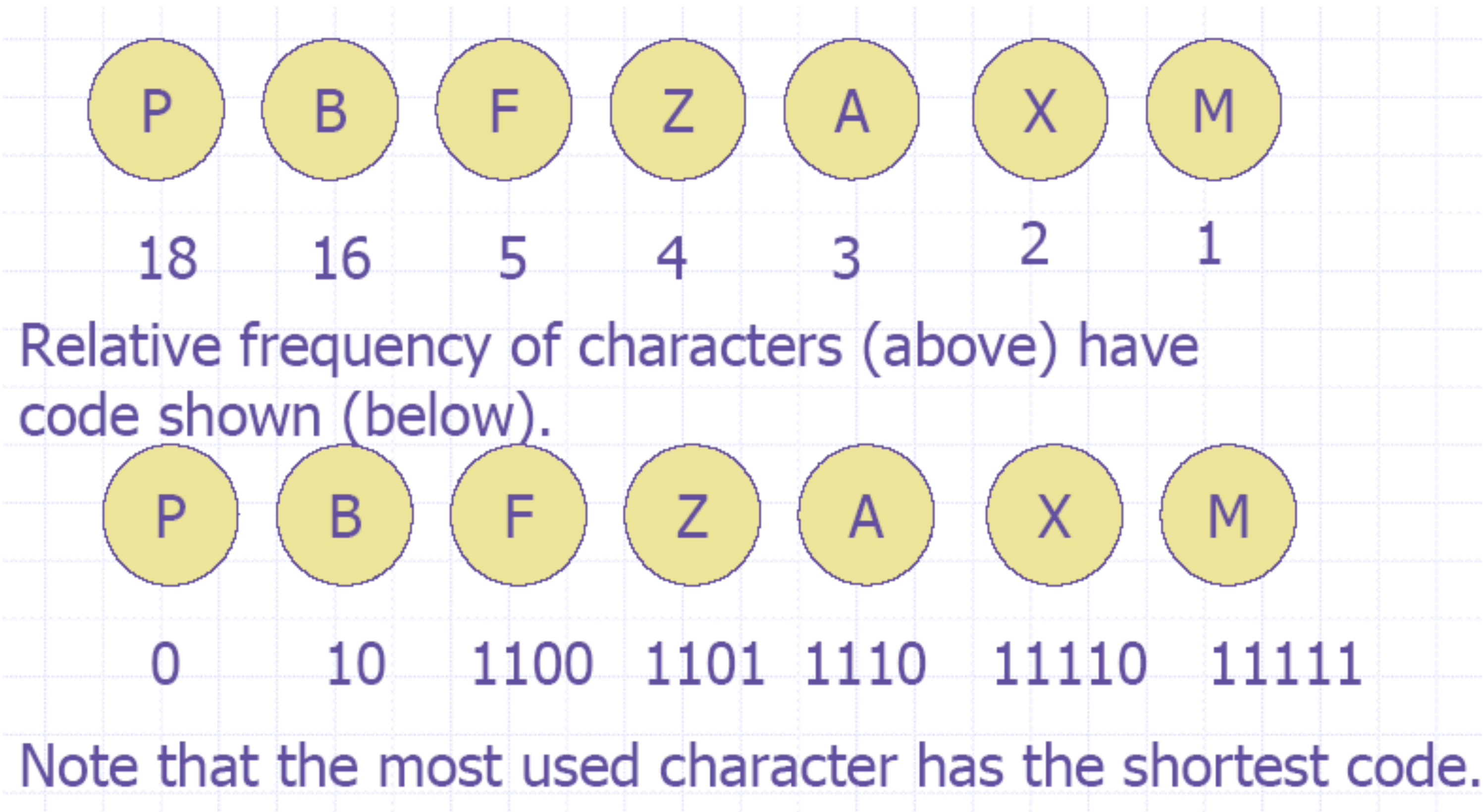
Huffman Code

- Code for a character is the path to that character node from the root



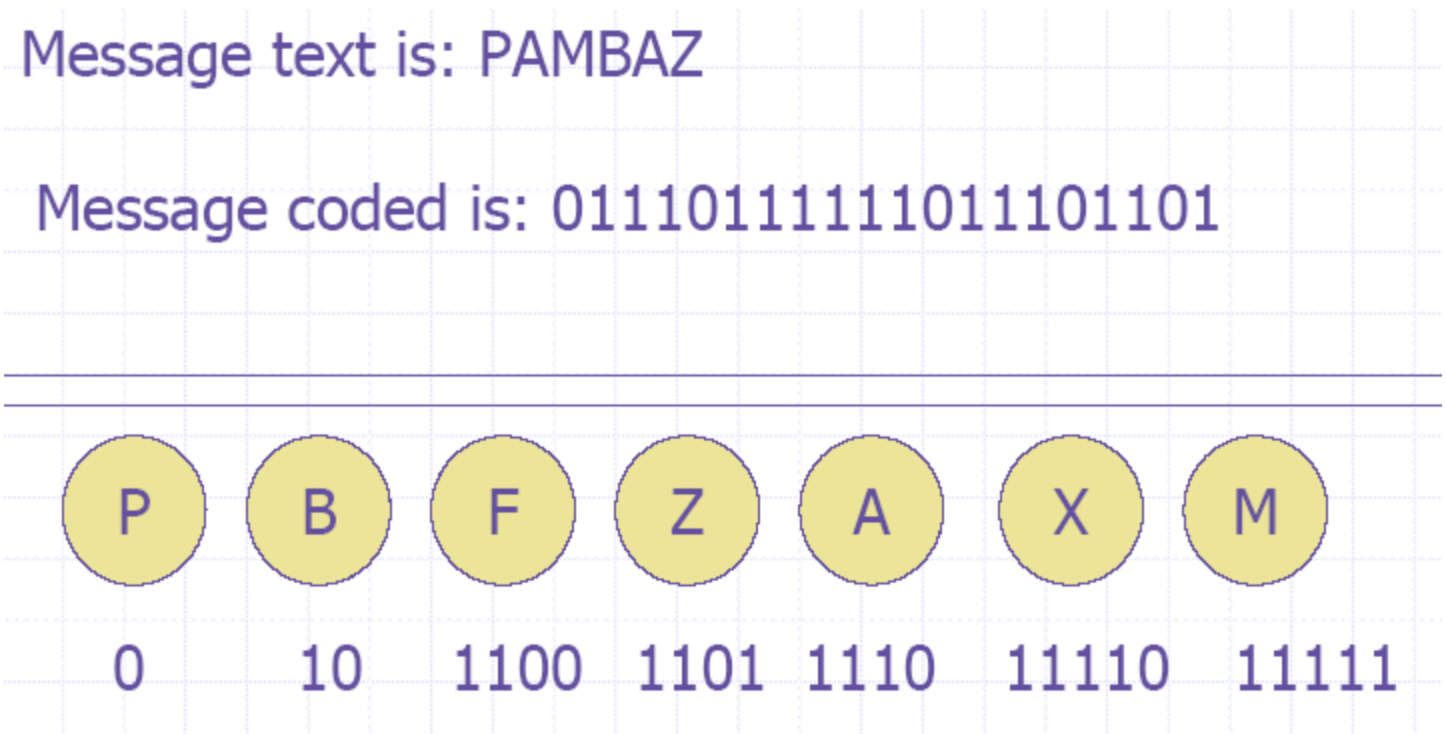
Huffman Code

- **Code for a character is the path to that character node from the root**



Huffman Code

- Encode the following message



Huffman Code

- **How to decode the coded message?**

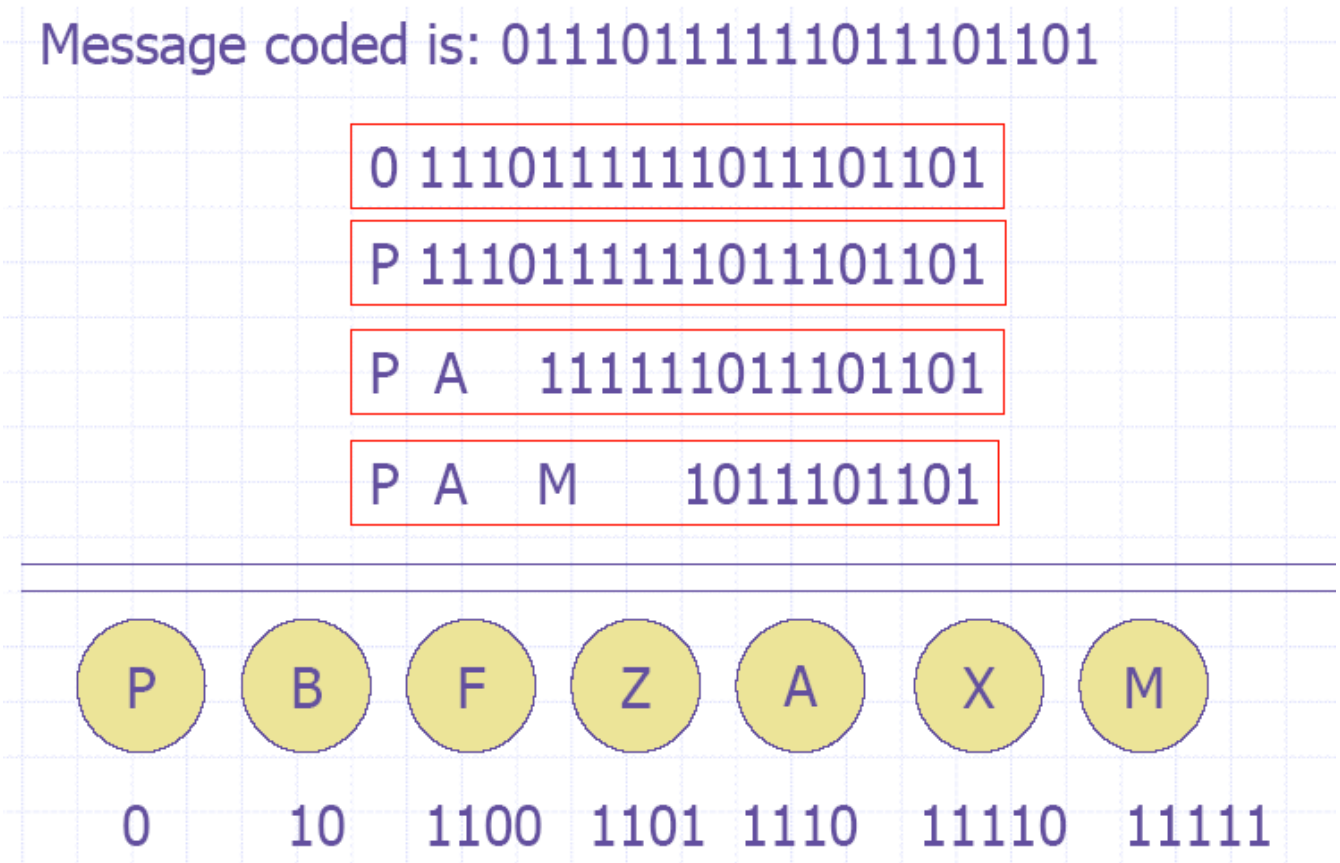
Message coded is: 01110111111011101101

Read the code bit by bit, as the bits correspond to some Character, accept that series of bits as that character

P	B	F	Z	A	X	M
0	10	1100	1101	1110	11110	11111

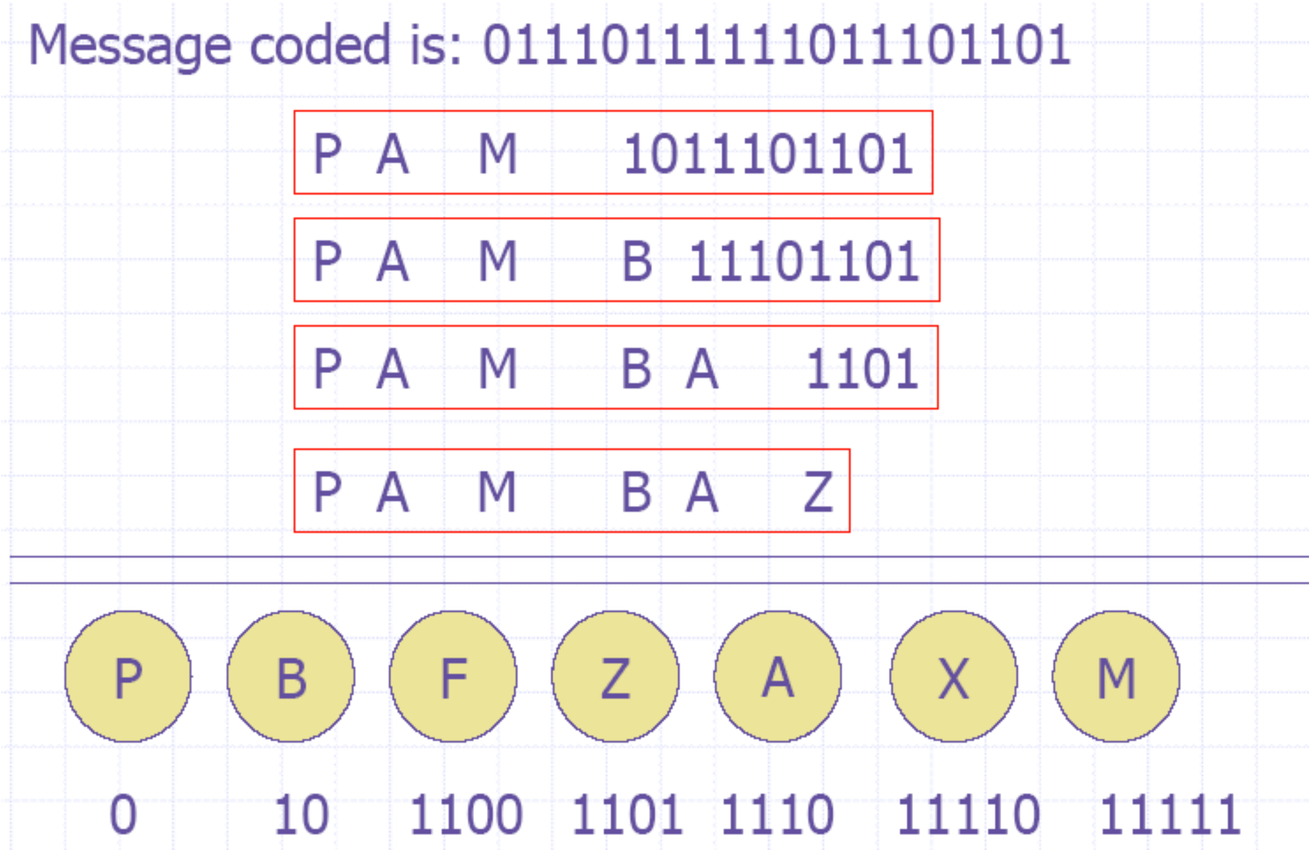
Huffman Code

- **How to decode the coded message?**



Huffman Code

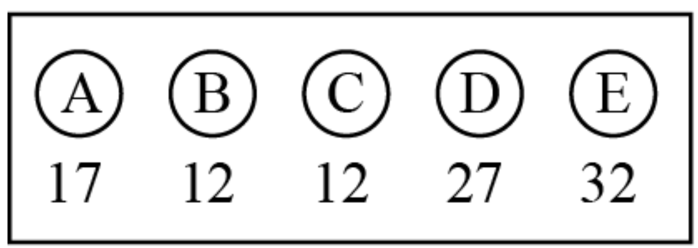
- How to decode the coded message?



Huffman Code

- **Another example**
  - **A character's code is found by starting at the root and following the branches that lead to that character. The code itself is the bit value of each branch on the path, taken in sequence**

Frequency of characters

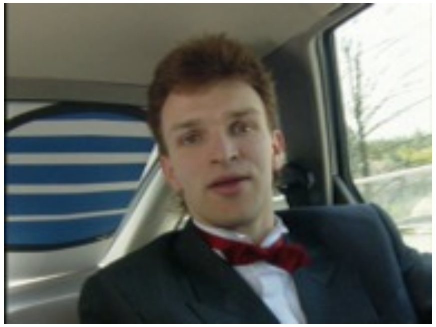


Huffman Code

- Shorter codes are assigned to higher probability data items
- No code contains any other codes as a prefix
- Reading from the left-hand bit, each code is uniquely decodable

Huffman Code

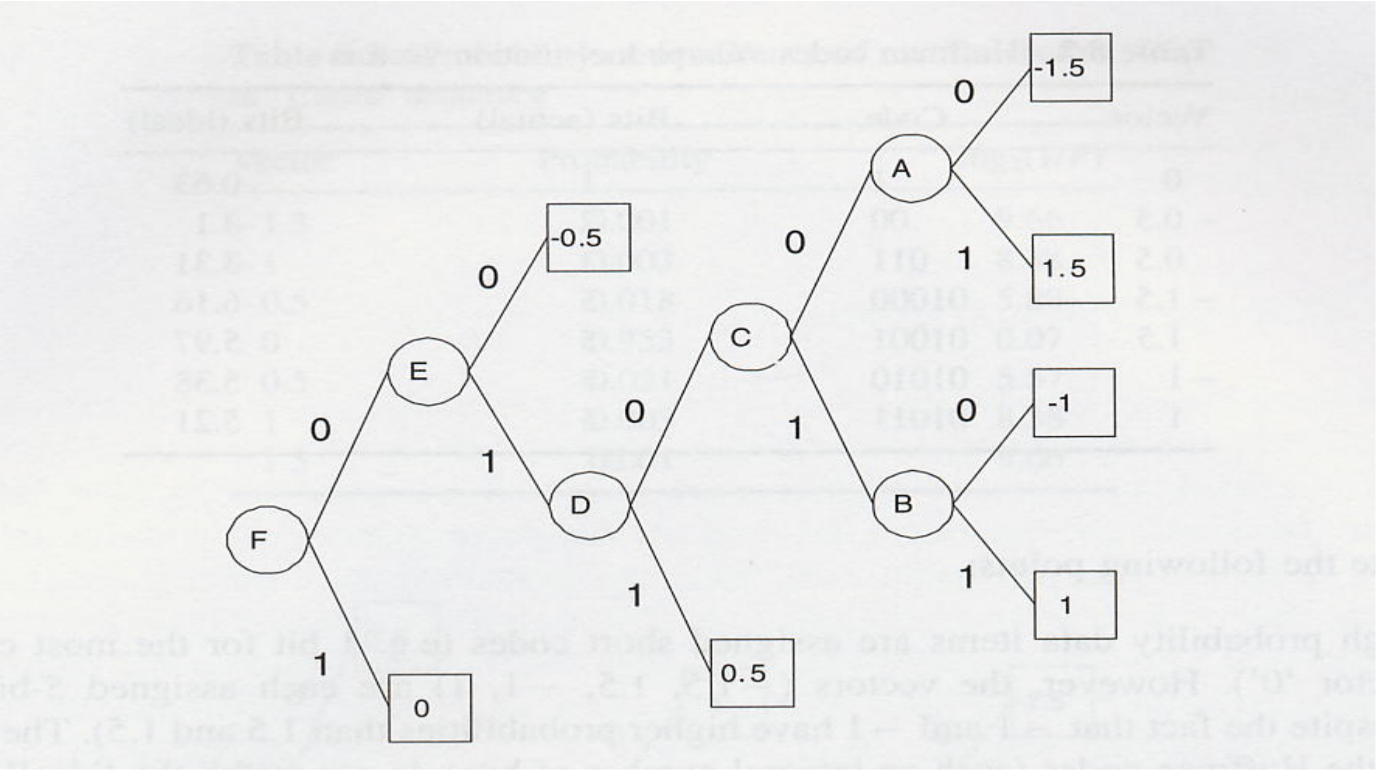
- **Another example**
  - **Carphone sequence**
    - **Probability of motion vectors**



Vector	Probability P	$\log_2(1/P)$
-1.5	0.014	6.16
-1	0.024	5.38
-0.5	0.117	3.10
0	0.646	0.63
0.5	0.101	3.31
1	0.027	5.21
1.5	0.016	5.97

Huffman Code

- **Another example**
  - **Huffman tree for Carphone sequence**



Huffman Code

- Another example
  - Huffman code for motion vectors of Carphone sequence

Vector	Code	Bits (actual)
0	1	1
−0.5	00	2
0.5	011	3
−1.5	01000	5
1.5	01001	5
−1	01010	5
1	01011	5

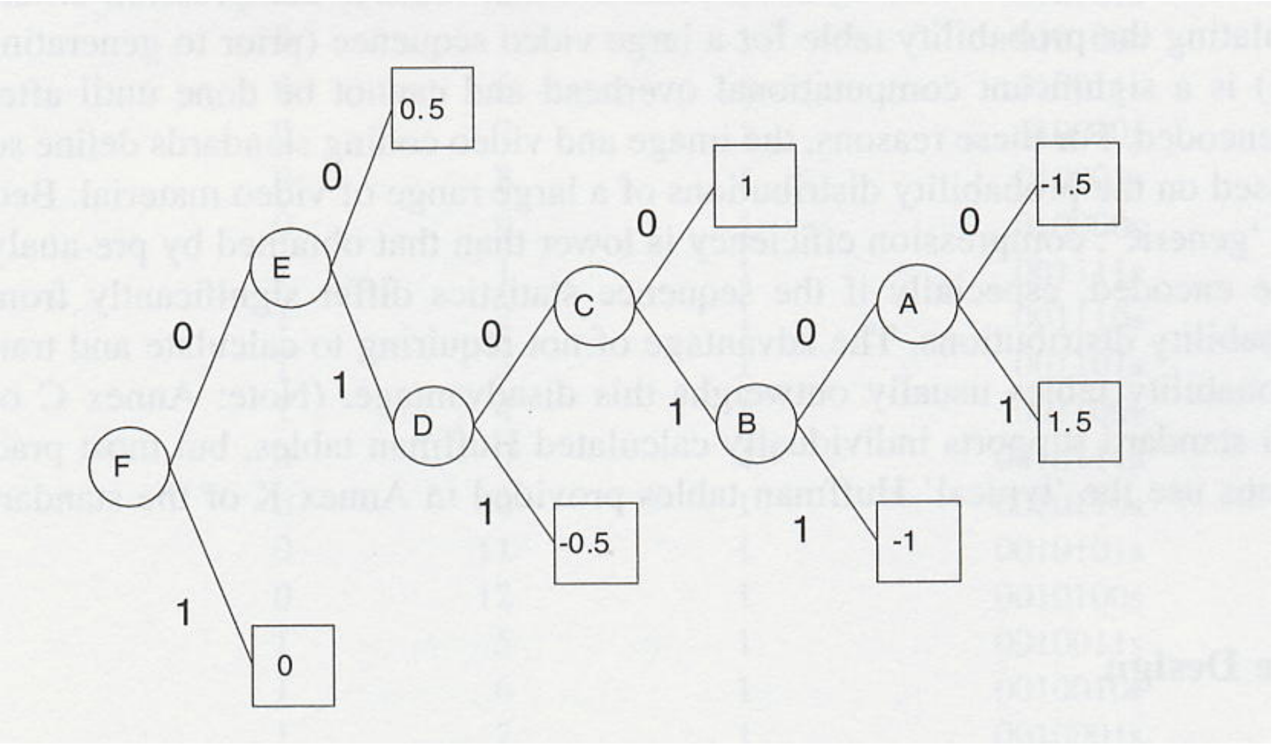
Huffman Code

- Another example
  - Claire sequence
    - Probability of motion vectors

Vector	Probability	$\log_2(1/P)$
– 1.5	0.001	9.66
– 1	0.003	8.38
– 0.5	0.018	5.80
0	0.953	0.07
0.5	0.021	5.57
1	0.003	8.38
1.5	0.001	9.66

Huffman Code

- **Another example**
  - **Huffman tree for Claire sequence**



Huffman Code

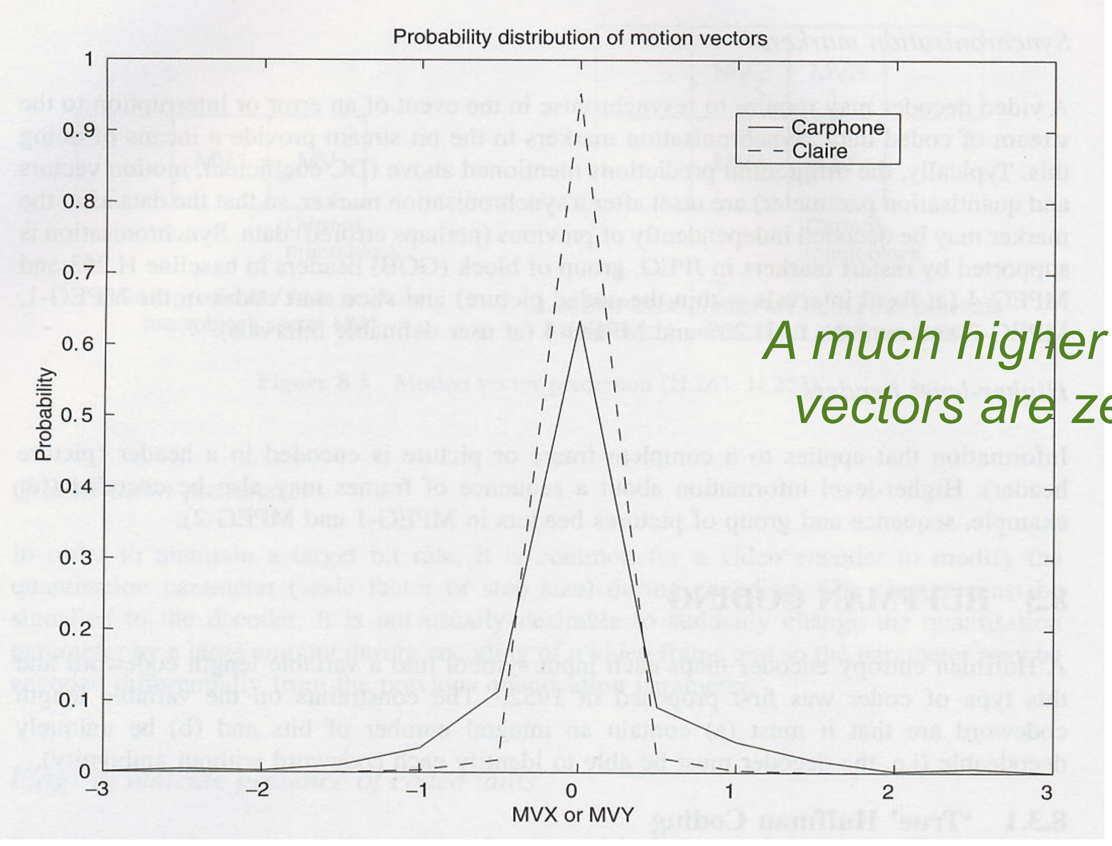
- **Another example**
  - **Huffman code for motion vectors in Claire sequence**

Vector	Code	Bits (actual)
0	1	1
0.5	00	2
−0.5	011	3
1	0100	4
−1	01011	5
−1.5	010100	6
1.5	010101	6

  - **To achieve optimum compression, a separate code table is required for each of the two sequences, *Carphone* and *Claire***

Huffman Code

- **Distribution of motion vectors**
  - **Carphone and Claire sequences**



A much higher proportion of vectors are zeros for Claire

Source Coding

- Arithmetic Code

Arithmetic Code

- Use the probability distribution of the data
- Convert sequence of data symbols into a single fractional/floating point number between 0 and 1
- The longer the sequence of symbols, the greater the precision required to represent the fractional number
- No direct correspondence between the code and the individual pixel values
- Theoretically achieves the maximum compression
- Quite different from Huffman coding, arithmetic coding is stream-based. It overcomes the drawbacks of Huffman coding

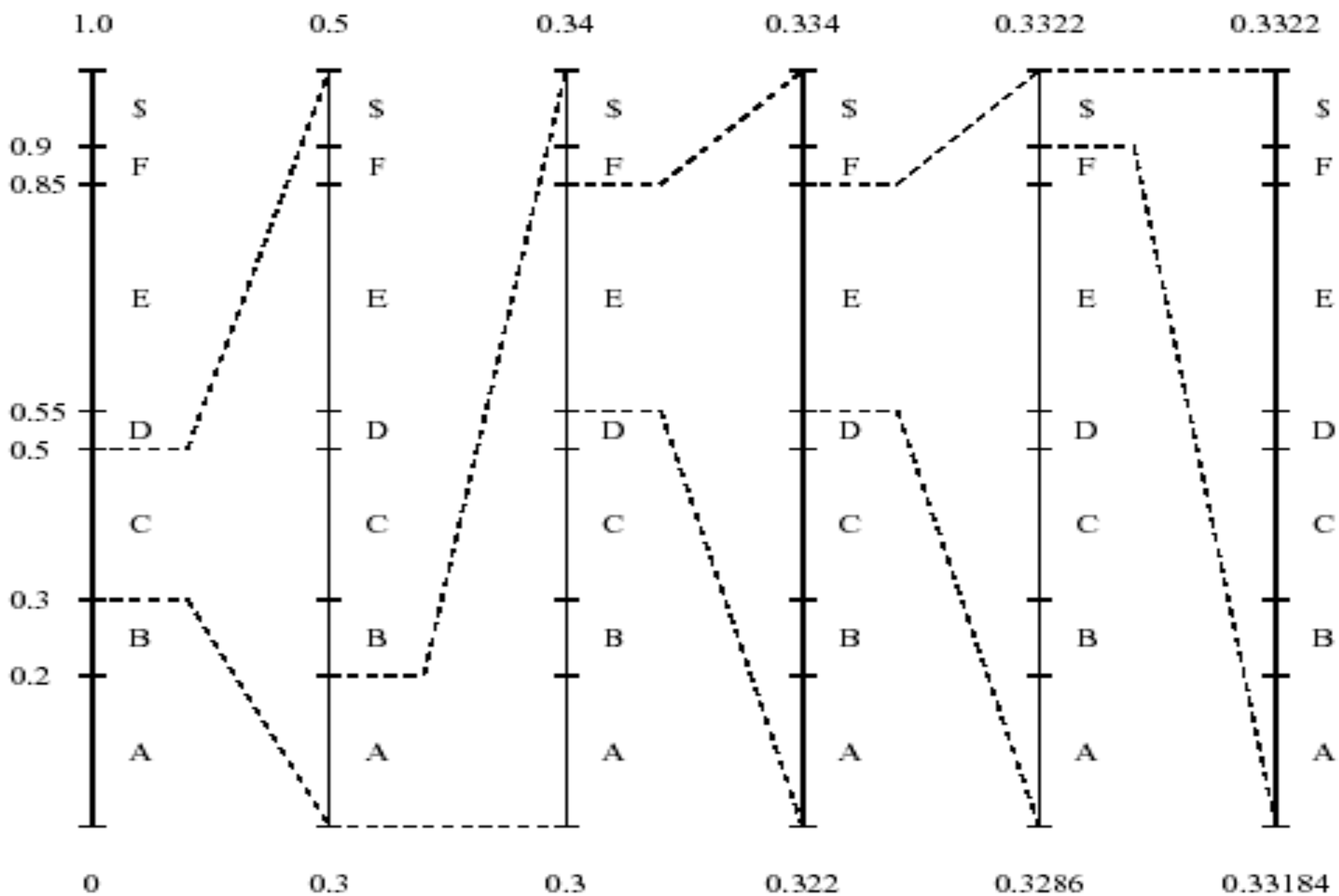
Arithmetic Code

- **Arithmetic Coding Encoder**

```
BEGIN
    low=0.0;        high=1.0;        range=1.0;
    while (symbol != terminator)
        {
            get (symbol);
            low=low+range*Range_low(symbol);
            high=low+range*Range_high(symbol);
            range=high-low;
        }
    output a code so that low<= code<high;
END
```

Arithmetic Code

- **Graphical display of shrinking ranges for "CAEE$"**



Arithmetic Code

- **Encode symbols "CAEE$"**
  - **New low, high and range generated**

Symbol	low	high	range
	0	1.0	1.0
C	0.3	0.5	0.2
A	0.30	0.34	0.04
E	0.322	0.334	0.012
E	0.3286	0.3322	0.0036
$	0.33184	0.33220	0.00036

Arithmetic Code

- Generating codeword for encoder

```
BEGIN
      code=0;
      k=1;
      while (value(code)<low)
            { assign 1 to the kth binary fraction bit
              if (value(code) > high)
                   replace the kth bit by 0
              k=k+1;
            }
END
```

- The final step in Arithmetic encoding calls for the generation of a number that falls within the range [low, high). The above algorithm will ensure that the shortest binary codeword is found

Arithmetic Code

- **Arithmetic coding decoder**

```
BEGIN
    get the binary code and convert to decimal value =
    value(code);
    Do
        { find a symbol s so that
          Range_low(s) <= value < Range_high(s);
          output s;
          low=Range_low(s);
          high=Range_high(s);
          range =high-low;
          value=[value-low]/range;
        }
    Until symbol s is a terminator
END
```

Arithmetic Code

- **Decode symbols "CAEE$"**

```
…
low=Range_low(s);
high=Range_high(s);
range =high-low;
value=[value-low]/range;
…
```

value	Output Symbol	low	high	range
0.33203125	C	0.3	0.5	0.2
0.16015625	A	0.0	0.2	0.2
0.80078125	E	0.55	0.85	0.3
0.8359375	E	0.55	0.85	0.3
0.953125	$	0.9	1.0	0.1

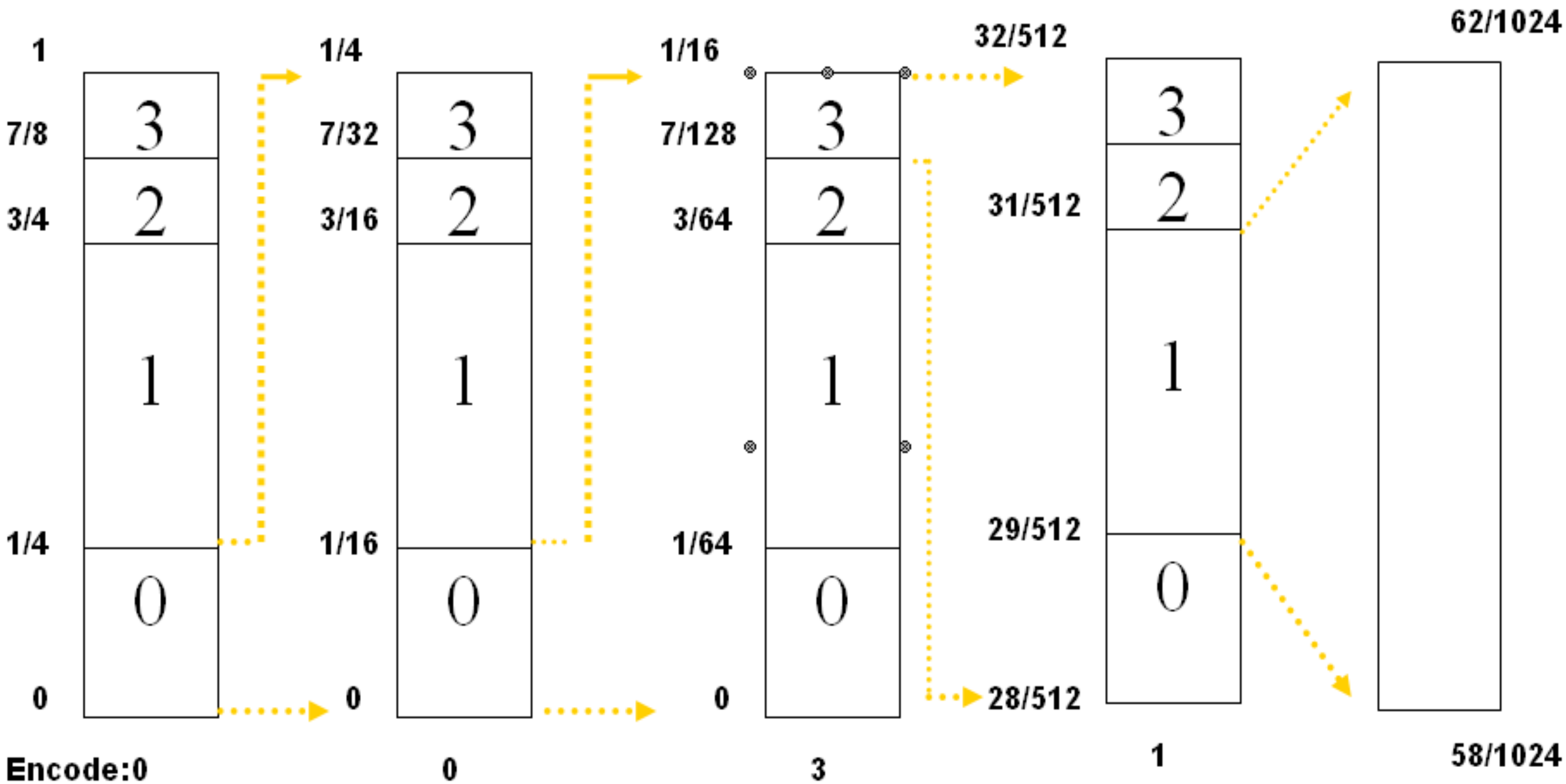
Arithmetic Code

- Another example

Probability Table

Pixel value	probability	Initial Subinterval
0	64/256=1/4	0 - 1/4
1	128/256=1/2	1/4 - 3/4
2	32/256=1/8	3/4 - 7/8
3	32/256=1/8	7/8 - 1

Arithmetic Code



Coding process for 0,0,3,1

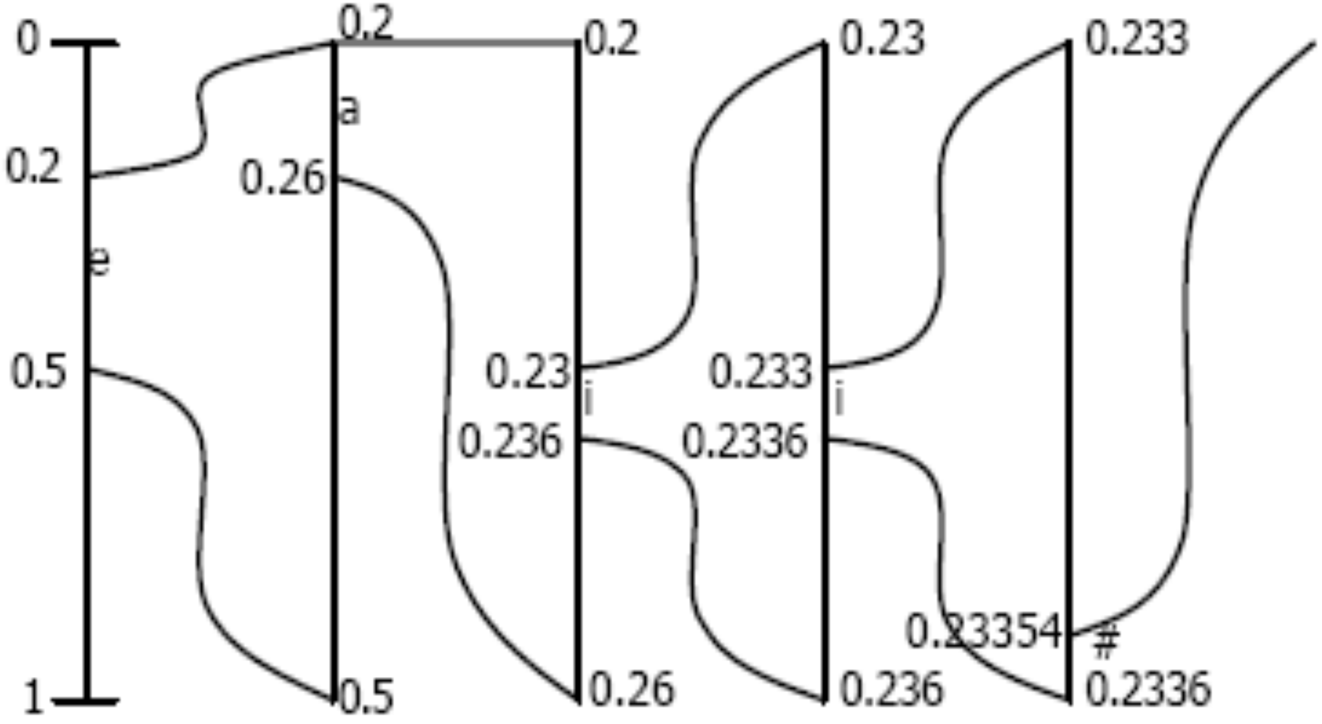
Arithmetic Code

- **Example**

Letter	Probability	Interval
a	0.2	[0,0.2)
e	0.3	[0.2,0.5)
i	0.1	[0.5,0.6)
o	0.2	[0.6,0.8)
u	0.1	[0.8,0.9)
#	0.1	[0.9,1)

Arithmetic Code

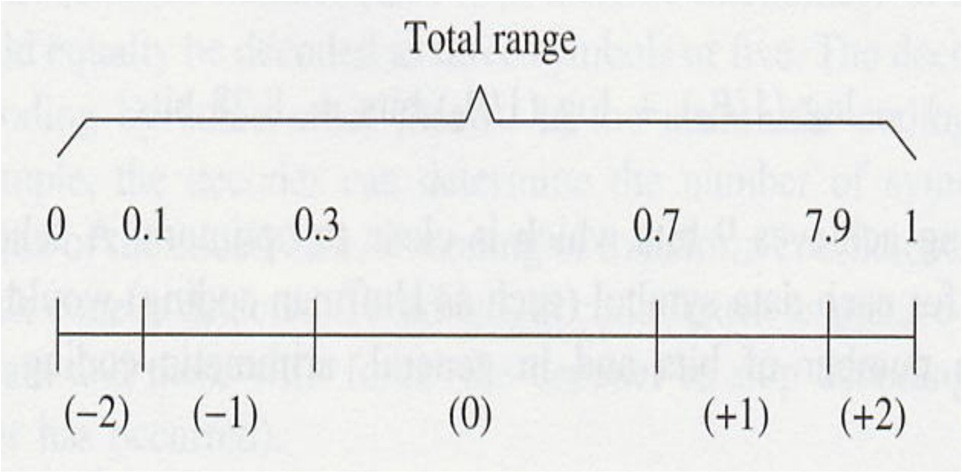
- **Message can be any** $x \in [0.23354, 0.2336)$



Arithmetic Code

- **Example**
  - **Each vector is assigned a subrange within the range 0-1 depending on its probability of occurrence**

Vector	Probability	Subrange
−2	0.1	0–0.1
−1	0.2	0.1–0.3
0	0.4	0.3–0.7
1	0.2	0.7–0.9
2	0.1	0.9–1.0

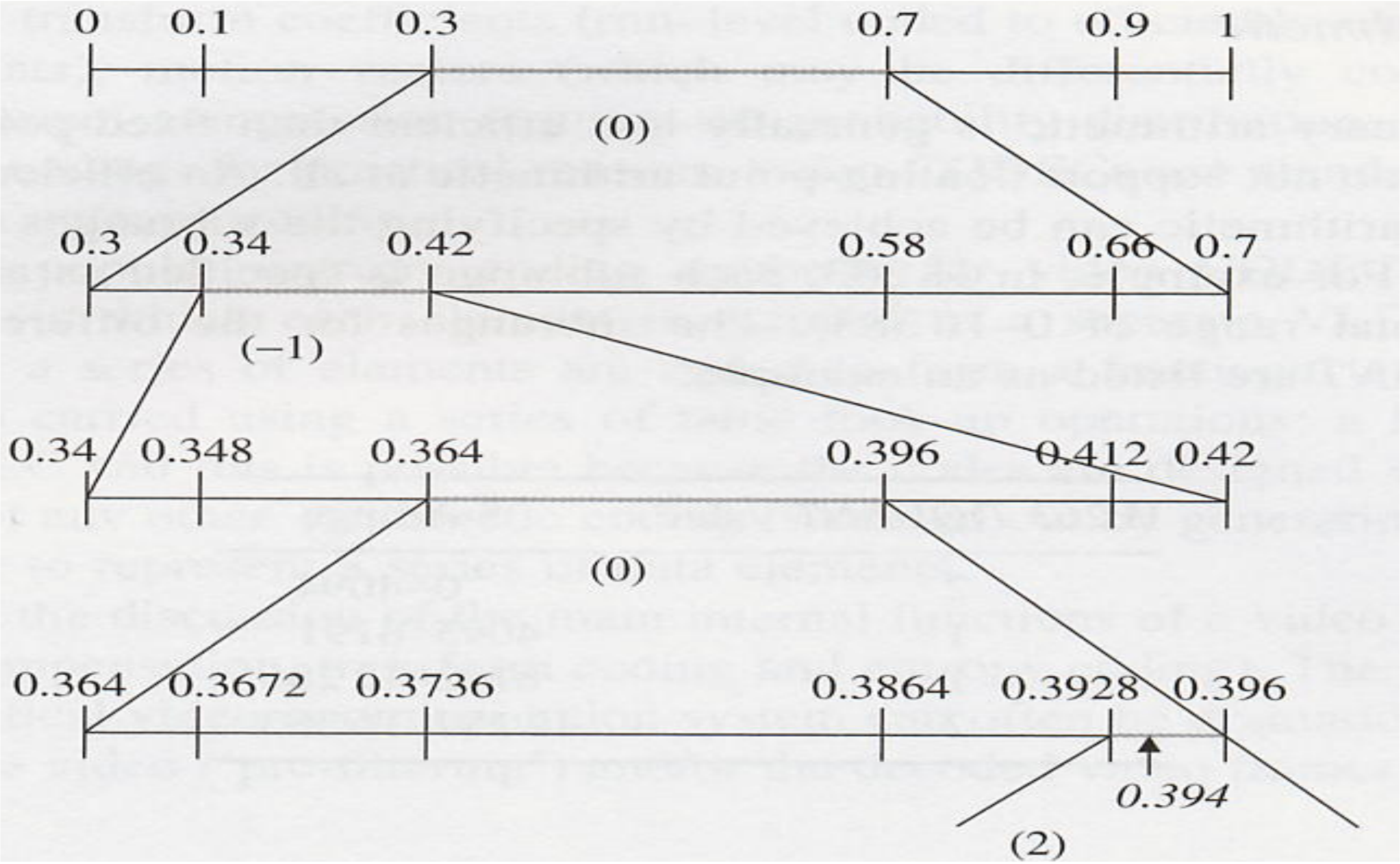


Encoding procedure

Encoding procedure	Range (L → H)	Symbol	Subrange (L → H)	Notes
1. Set the initial range	0 → 1.0			
2. For the first data symbol, find the corresponding subrange (low to high).		(0)	0.3 → 0.7	
3. Set the new range (1) to this subrange	0.3 → 0.7			
4. For the next data symbol, find the subrange L to H		(– 1)	0.1 → 0.3	This is the subrange within the interval 0–1
5. Set the new range (2) to this subrange within the previous range	0.34 → 0.42			0.34 is 10% of the range; 0.42 is 30% of the range
6. Find the next subrange		(0)	0.3 → 0.7	
7. Set the new range (3) within the previous range	0.364 → 0.396			0.364 is 30% of the range; 0.396 is 70% of the range
8. Find the next subrange		(2)	0.9 → 1.0	
9. Set the new range (4) within the previous range	0.3928 → 0.396			0.3928 is 90% of the range; 0.396 is 100% of the range

Arithmetic Code

- **Example**



Decoding procedure

The sequence of subranges (and hence the sequence of data symbols) can be decoded from this number as follows.

Decoding procedure	Range	Subrange	Decoded symbol
1. Set the initial range	$0 \rightarrow 1$		
2. Find the subrange in which the received number falls. This indicates the first data symbol		$0.3 \rightarrow 0.7$	(0)
3. Set the new range (1) to this subrange	$0.3 \rightarrow 0.7$		
4. Find the subrange *of the new range* in which the received number falls. This indicates the second data symbol		$0.34 \rightarrow 0.42$	(−1)
5. Set the new range (2) to this subrange within the previous range	$0.34 \rightarrow 0.42$		
6. Find the subrange in which the received number falls and decode the third data symbol		$0.364 \rightarrow 0.396$	(0)
7. Set the new range (3) to this subrange within the previous range	$0.364 \rightarrow 0.396$		
8. Find the subrange in which the received number falls and decode the fourth data symbol		$0.3928 \rightarrow 0.396$	(2)

Arithmetic Code

- An optional alternative to Huffman coding in several video coding standards(e.g. H.263, MPEG-4,H.26L)
- Pre-calculated subranges are defined by the standard (based on typical probability distributions)
  - Advantage: avoiding the need to calculate and transmit probability distributions
  - Disadvantage: suboptimal compression
- Syntax-based Arithmetic coding (SAC) in H.263 (about 5% decrease in bit-rate)

Arithmetic Code

- Patent Issues
  - IBM’s Q-coder
  - Q-Coder adaptive binary arithmetic coder
  - Some developers of commercial video coding systems have avoided the use of arithmetic coding because of concerns about potential patent infringement
  - H.264/AVC
    - Arithmetic coding is optional

Channel Coding

- Hamming Code

Channel Coding

- Hamming code
  - Popular error correcting code in RAM
  - Richard Hamming 1950

- In Hamming code
  - k bits parity code (check bit) and
  - n data code (data bit)
  - → form a (n+k) data word

  - Allocate parity codes to the positions of power of 2
  - May be applied to any length of code

Channel Coding

- **Hamming code**
  - **Example: 8 bits data 11000100**

1	2	3	4	5	6	7	8	9	10	11	12
P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

- **P1=the XOR of bit(3, 5, 7, 9, 11) =$1\oplus1\oplus0\oplus0\oplus0$=0**
- **P2=the XOR of bit(3, 6, 7, 10, 11) =$1\oplus0\oplus0\oplus1\oplus0$=0**
- **P4=the XOR of bit(5, 6, 7, 12) =$1\oplus0\oplus0\oplus0$=1**
- **P8=the XOR of bit(9, 10, 11, 12) =$0\oplus1\oplus0\oplus0$=1**

The exclusive-OR operation perform an odd function

Channel Coding

- **Hamming code**
  - **Example: 12 bits data word**

1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	1	0	0	1	0	1	0	0

  - **Check:**
    - **C1 = the XOR of bits (1,3,5,7,9,11)**
    - **C2 = the XOR of bits (2,3,6,7,10,11)**
    - **C4 = the XOR of bits (4,5,6,7,12)**
    - **C8 = the XOR of bits (8,9,10,11,12)**
    - **C = C8C4C2C1 = 0 0 0 0 (no error occur)**

Channel Coding

- **Check (cont'd)**
  - **If C ≠0, then the 4-bit binary number formed by the check bits gives the position of the erroneous bit.**
    For example :

1	2	3	4	5	6	7	8	9	10	11	12	Bit position
0	0	1	1	1	0	0	1	0	1	0	0	No error
1	0	1	1	1	0	0	1	0	1	0	0	Error in bit 1
0	0	1	1	0	0	0	1	0	1	0	0	Error in bit 5

	C_8	C_4	C_2	C_1
For no error:	0	0	0	0
With error in bit 1:	0	0	0	1
With error in bit 5:	0	1	0	1

C1 = the XOR of bits (1,3,5,7,9,11)
C2 = the XOR of bits (2,3,6,7,10,11)
C4 = the XOR of bits (4,5,6,7,12)
C8 = the XOR of bits (8,9,10,11,12)

Channel Coding

- If a nonzero value is found, the decoder simply complements the codeword bit position indicated by the parity word

- k bits v.s. n data bits

Number of check nits, k	Range of data bits, n
3	2-4
4	5-11
5	12-26
6	27-57
7	58-120

Channel Coding

- Hamming code
  - Linear code
  - Can be computed in the form of matrices

- Example: 4 data bits: 1011
  - Code generator matrix

$$G=\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  - Parity-check matrix

$$C=\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Channel Coding

- **Hamming code**
  - **Example (cont'd)**

$$h=\begin{pmatrix}1&1&0&1\\1&0&1&1\\1&0&0&0\\0&1&1&1\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}\begin{pmatrix}1\\0\\1\\1\end{pmatrix}=\begin{pmatrix}2\\3\\1\\2\\0\\1\\1\end{pmatrix}=\begin{pmatrix}0\\1\\1\\0\\0\\1\\1\end{pmatrix}$$

Case1:

$$p=\begin{pmatrix}1&0&1&0&1&0&1\\0&1&1&0&0&1&1\\0&0&0&1&1&1&1\end{pmatrix}\begin{pmatrix}0\\1\\1\\0\\0\\1\\1\end{pmatrix}=\begin{pmatrix}2\\4\\2\end{pmatrix}=\begin{pmatrix}0\\0\\0\end{pmatrix}$$

Case2:

$$p=\begin{pmatrix}1&0&1&0&1&0&1\\0&1&1&0&0&1&1\\0&0&0&1&1&1&1\end{pmatrix}\begin{pmatrix}0\\1\\1\\0\\1\\1\\1\end{pmatrix}=\begin{pmatrix}3\\4\\3\end{pmatrix}=\begin{pmatrix}1\\0\\1\end{pmatrix}$$