Fundamentals of Information Theory

Data Compression

Yayu Gao
School of Electronic Information and Communications
Huazhong University of Science and Technology
Email: yayugao@hust.edu.cn

Outline

- Three key questions about data compression
- What is source coding?
- Get to know some codes
- What do we want from a source code?
- Kraft inequality——constraints on prefix codes
- How to find the optimal code?
- Shannon’s first theorem——Zero-error source coding theorem
- From Theory to Applications: source coding algorithms

本节学习目标

1. 写出Shannon code的算法流程
2. 能够编写Shannon code
3. 说出为什么Shannon code不是compact code
4. 写出Huffman code的算法流程
5. 能够编写Huffman code
6. 说出Huffman code的≥3个特点
7. 证明Huffman code的最优性
8. 能够编写Q-ary Huffman code
9. 说出Huffman code的2个局限性

重难点：
- ➢ Shannon code
- ➢ Huffman code

08

From Theory to Applications: Source Coding Algorithms

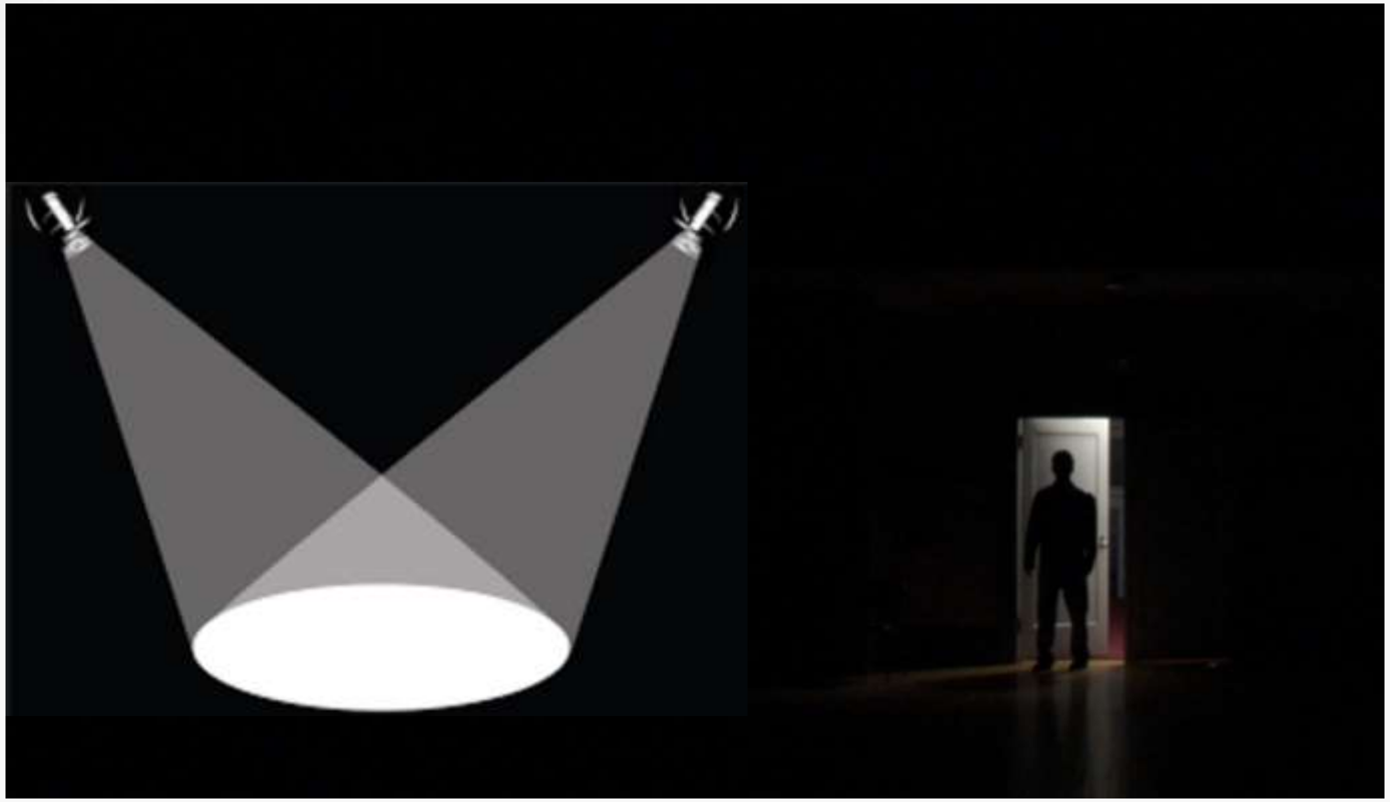
Revisiting: source coding theorem

- Source coding theorem
  - For a binary information source S and arbitrary ε, there exists a binary instantaneous code for which the average code length L per coding symbol satisfies

$$\boxed{H(S) \le L_n^* < H(S) + \varepsilon.}$$

  - Provide the theoretical limit to achieve the ideal coding
  - Prove the existence of the ideal source code.

Zero-Error Source Coding: From Theory to Applications



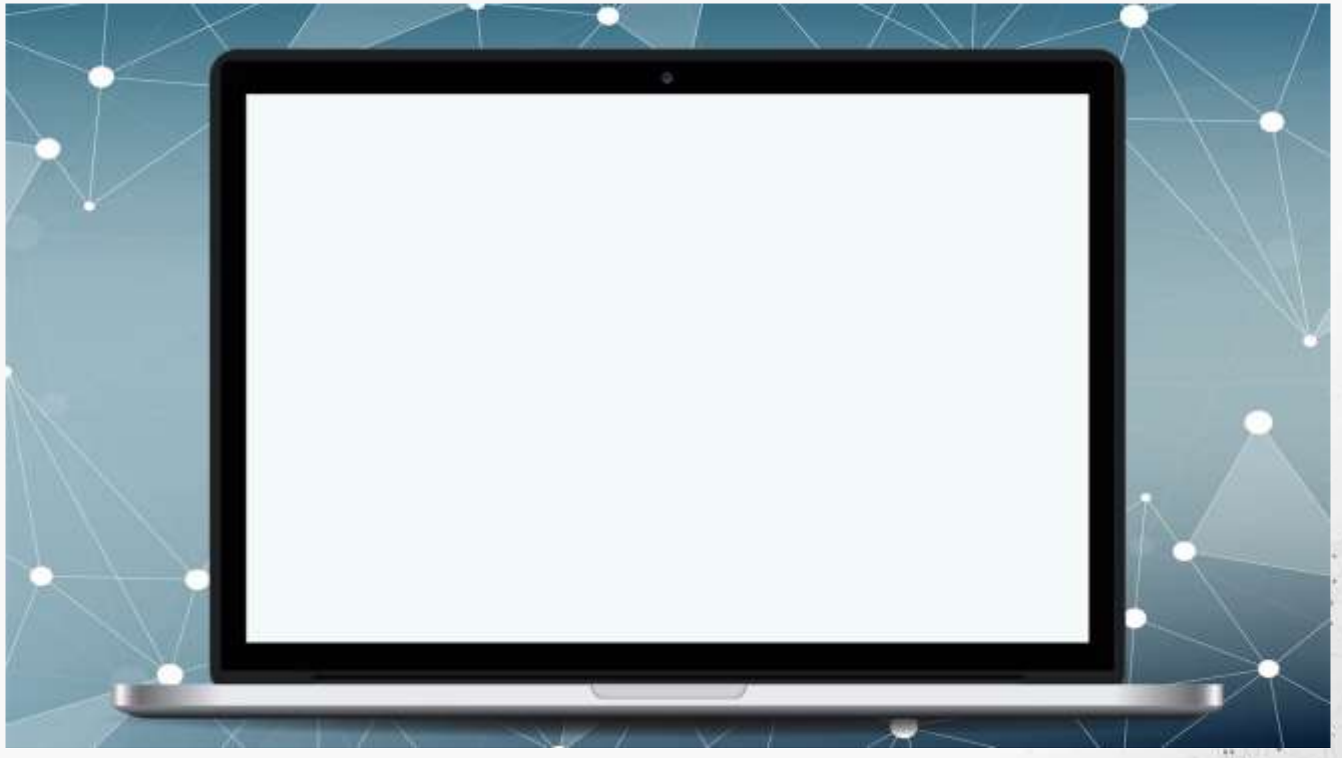
Question: How to design the optimal source codes?

Applications: How to design the optimal source codes?

- A large amount of source coding algorithms have been proposed after Shannon’s first theorem, aiming to approach the data compression limit.
  - Shannon code (1948)
  - Shannon-Fano code (1949)
  - Huffman code (1952)
  - Run-length code (1966)
  - Universal coding (1975)
  - Arithmetic coding (1976)
  - Lempel-Ziv coding (1977)
  - …

Applications: How to design the optimal source codes?

- A large amount of source coding algorithms have been proposed after Shannon’s first theorem, aiming to approach the data compression limit.
  - Shannon codes (1948)
  - Shannon-Fano codes (1949)
  - Huffman codes (1952)
  - Universal coding
  - Arithmetic coding
  - Lempel-Ziv coding
  - …

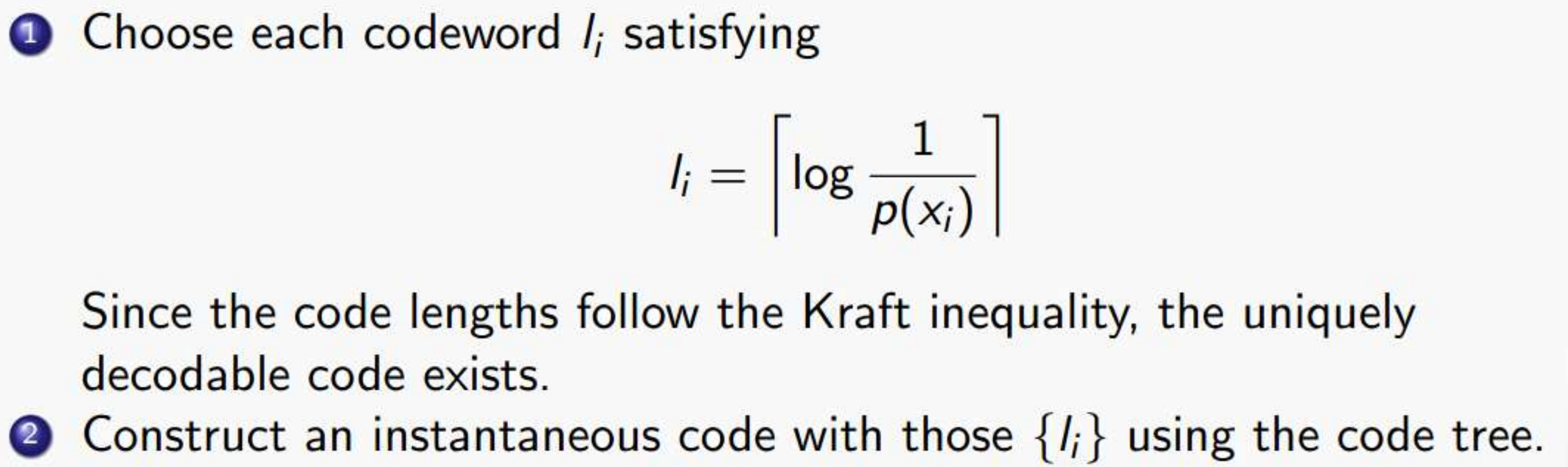


Shannon codes

Shannon codes: Overview

- Idea: deducted from Shannon's first theorem

- Method



1. Choose each codeword l_i satisfying

$$l_i = \left\lceil \log \frac{1}{p(x_i)} \right\rceil$$

Since the code lengths follow the Kraft inequality, the uniquely decodable code exists.

2. Construct an instantaneous code with those $\{l_i\}$ using the code tree.

Shannon codes: Algorithm

Given a discrete memoryless source

$$\begin{bmatrix} X \\ p(x) \end{bmatrix} = \left\{ \begin{array}{cccc} x_1, & x_2, & \cdots & x_n \\ p(x_1), & p(x_2), & \cdots & p(x_n) \end{array} \right\}, \sum_i p(x_i) = 1.$$

For simplicity, $p(x_1) \geq p(x_2) \geq \cdots \geq p(x_n)$.

1. $p(x_0) = 0$.
2. Define the cumulative distribution function

$$p_a(x_i) = \sum_{j=0}^{i-1} p(x_j), i = 1, 2, \ldots, n.$$

3. $l_i = \left\lceil \log \frac{1}{p(x_i)} \right\rceil$ is the code length of i-th code word.

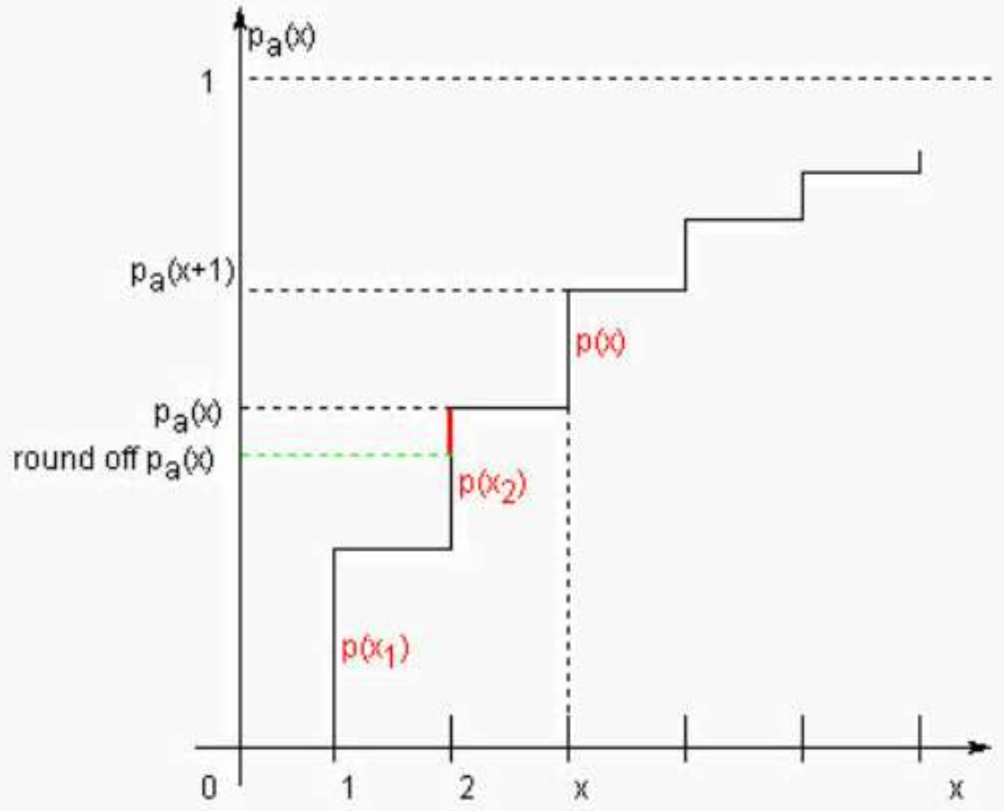
$$\log_2 \frac{1}{p(x_i)} \leq l_i < \log_2 \frac{1}{p(x_i)} + 1$$

4. Code $p_a(x_i)$ using binary, and take l_i digits after the dot as the code for x_i.

Shannon codes: Reflection

- $p(x_0) = 0$
- $p(x_1) \geq p(x_2) \geq \cdots \geq p(x_n)$
- $p_a(x_i) = \sum_{j=0}^{i-1} p(x_j)$

  $(i = 1, 2, \ldots, n)$
- Round-off the cumulative distribution function to l_i bits: $\lfloor p_a(x_i) \rfloor_{l_i}$
- Use the first l_i bits as a code for x_i.



- x_i and $p_a(x_i)$ is one-to-one mapping, such that coding for $p_a(x_i)$ can be seen as coding for x_i.
- $p_a(x_i) - \lfloor p_a(x_i) \rfloor_{l_i} \leq \frac{1}{2^{l_i}} \leq p(x_i)$, such that the round-off CDF $\lfloor p_a(x_i) \rfloor_{l_i}$ lies in the corresponding interval of x_i.

Shannon codes: Example

Assume

$$\begin{bmatrix} X \\ p(x) \end{bmatrix} = \left\{ \begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ 0.25 & 0.25 & 0.2 & 0.15 & 0.1 & 0.05 \end{array} \right\}, \sum_i p(x_i) = 1.$$

- Please design a Shannon code for this source.

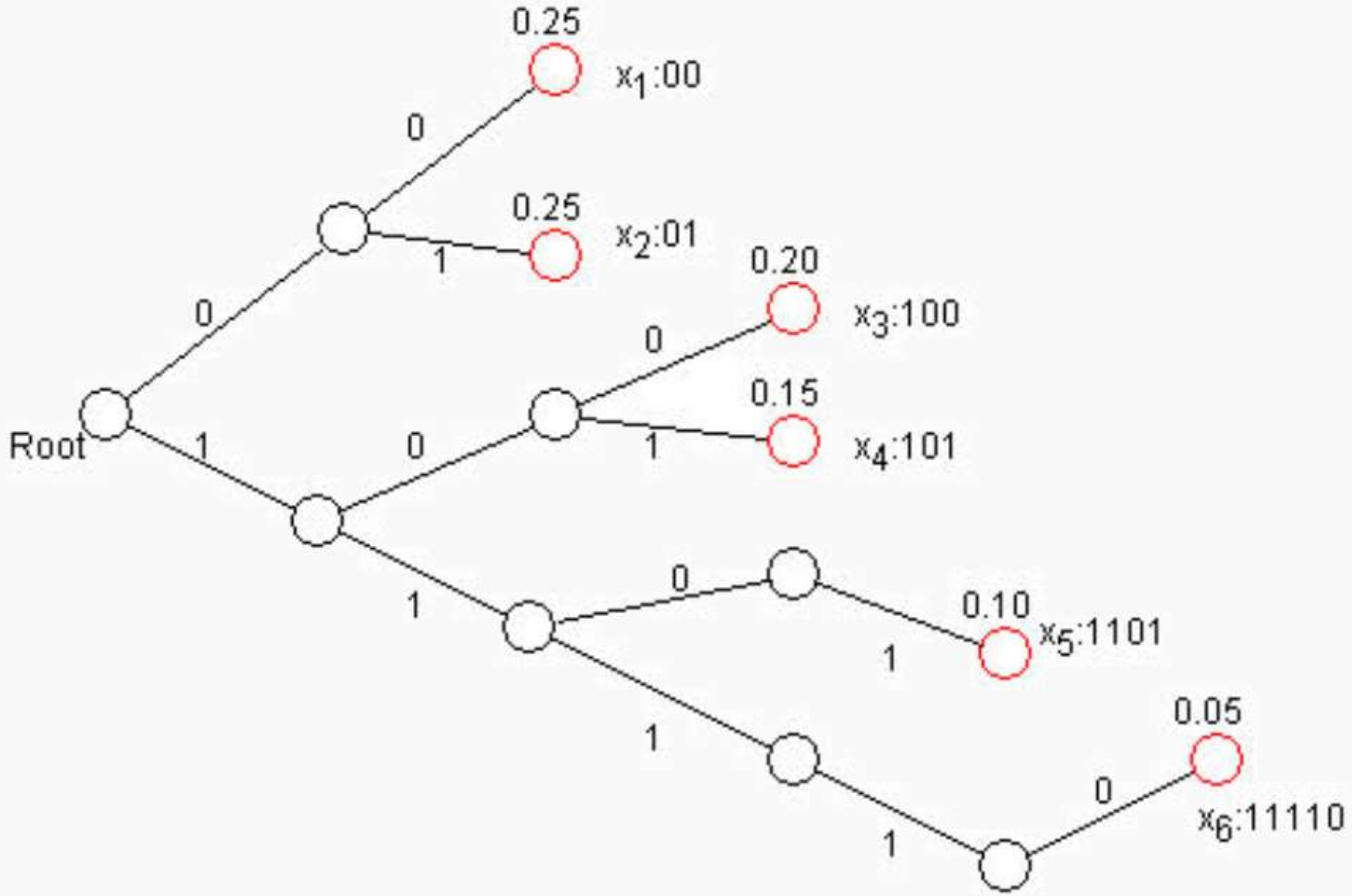
Shannon codes: Example

Assume

$$\begin{bmatrix} X \\ p(x) \end{bmatrix} = \left\{ \begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ 0.25 & 0.25 & 0.2 & 0.15 & 0.1 & 0.05 \end{array} \right\}, \sum_i p(x_i) = 1.$$

x_i	$p(x_i)$	i	$p_a(x_i)$	$p_a(x_i)$ binary	l_i	codeword
x_1	0.25	1	0.00	0.00	2	00
x_2	0.25	2	0.25	0.01	2	01
x_3	0.20	3	0.50	0.100	3	100
x_4	0.15	4	0.70	0.101***	3	101
x_5	0.10	5	0.85	0.1101**	4	1101
x_6	0.05	6	0.95	0.11110*	5	11110

Shannon codes: coding tree



- Any problem?

Shannon codes: analysis

- To evaluate the degree of one source coding algorithm close to the Shannon's data compression limit.

- Definition

$$\boxed{\eta = \frac{H(S)}{\bar{L}},}$$

source entropy

average code length

- Comments
  - For zero-error codes, $\eta \leq 1$. A larger η indicates higher coding efficiency.

Shannon codes: analysis

- Average length

$$\bar{L} = 0.25 \times 2 \times 2 + (0.2 + 0.15) \times 3 + 0.1 \times 4 + 0.05 \times 5$$
$$= 2.7 \text{ bits/symbol}$$

- Source entropy:

$$H(X) = -\sum_{i=1}^{6} p(x_i) \log_2 p(x_i) = 2.42 \text{ bits/symbol}$$

- Code efficiency

$$\eta = \frac{H(X)}{\bar{L}} = 89.63\%$$

- Comments: the efficiency of Shannon codes is not very high, we need to search for more efficient coding methods.

$$\bar{L} = \sum_x p(x) l(x) = \sum_x p(x) \left(\left\lceil \log \frac{1}{p(x)} \right\rceil \right) < H(X) + 1$$

Shannon codes: summary

- The upper bound of optimal code lengths doesn't necessarily result in a good code.

$$\text{Shannon codes: } l_i = \left\lceil \log \frac{1}{p(x_i)} \right\rceil, \quad H(X) \le \bar{L} < H(X) + 1.$$

- Consider two symbols with probability 0.9999 and 0.0001. What are their codeword lengths for the Shannon code?

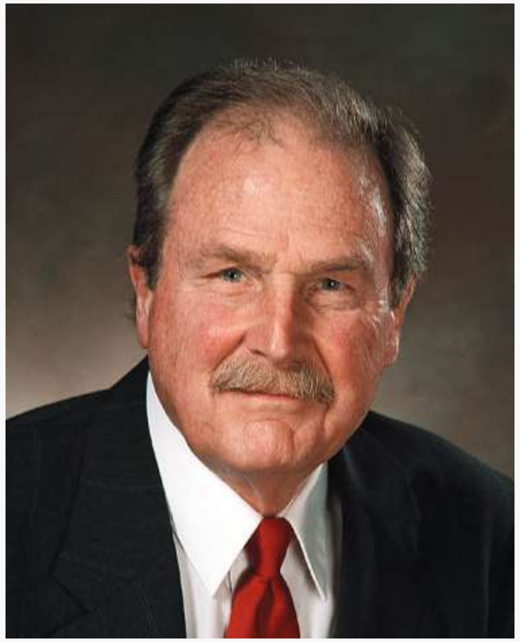
- Limitations: The codeword for **infrequent** symbol is usually **longer** in the Shannon code.

- In general, **Shannon codes are not compact codes.**
  - It can achieve the minimum average code length only when the source symbols are uniformly distributed.

Huffman Codes

Huffman codes: Overview

- A compact code construction algorithm invented by David A. Huffman in 1952.
- Basic idea: Constructed using a code tree, but starting at the leaves.
  - Do not know what is the code length at the beginning.
- Optimal in **average code length**, Widely used in data compression.


David Huffman

Simple **Effective** **Optimal**

D. A. Huffman, "A method for the construction of minimum redundancy codes," in IRE, vol. 40, pp. 1098-1101, 1952.

Huffman codes: Algorithm

1. Make a leaf node for each code symbol.
   - Add the generation probability of each symbol to the leaf node in a descending order.

2. Take the two leaf nodes with the smallest probability and connect them into a new node.
   - The probability of the new node is the sum of the probabilities of the two connecting nodes.
   - Add 1 or 0 to each of the two branches.

3. If there is only one node left, the code construction is completed. If not, go back to Step 2.

4. The codeword of each symbol is the binary sequence from the root to the leaf node.

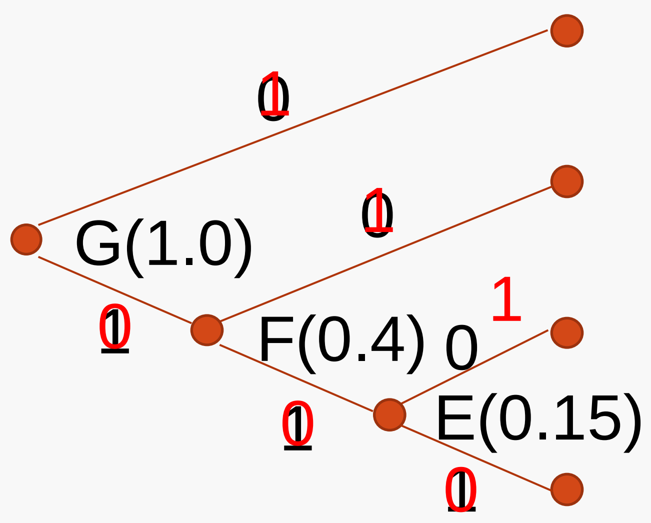
Huffman codes: Example #1

- Construct a binary Huffman code for the following source.

$$\begin{bmatrix} X \\ P \end{bmatrix} = \begin{bmatrix} A & B & C & D \\ 0.05 & 0.25 & 0.1 & 0.6 \end{bmatrix}$$

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log [p(x)]$$

$$\boxed{L(C) = \sum_{x \in \mathcal{X}} p(x) l(x)}$$

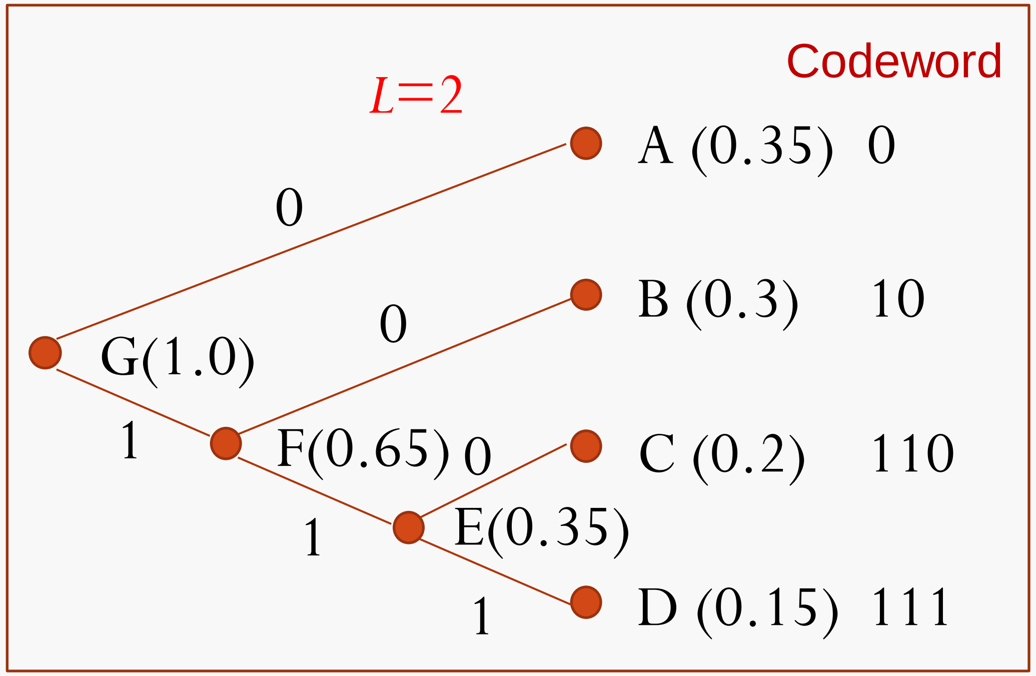


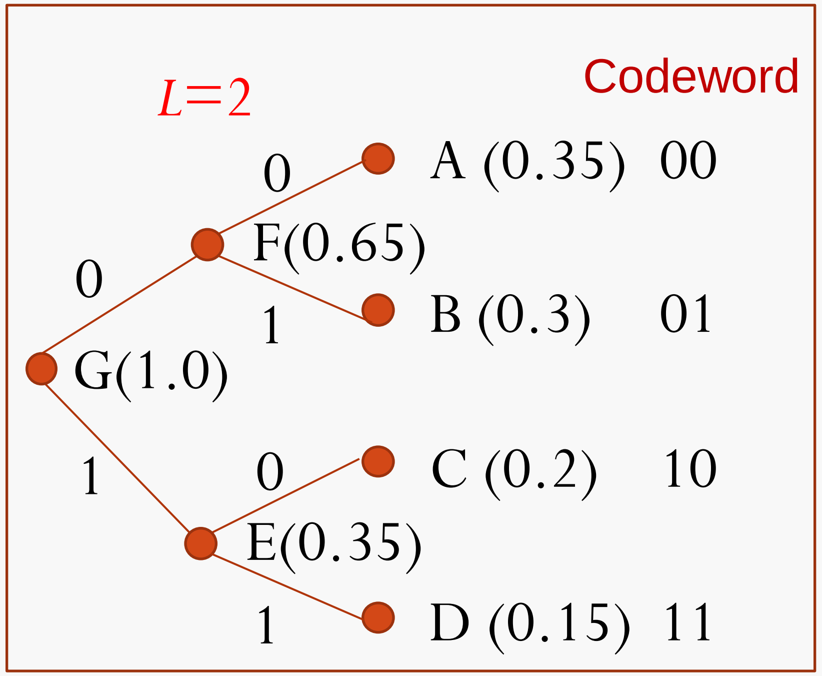
		Codeword
D (0.6)	0	1
B (0.25)	10	01
C (0.1)	110	001
A (0.05)	111	000

- Can you find another Huffman code for this source?

Huffman codes: Example #2

- Construct a binary Huffman code for the following source.





- H=1.93 bits.
- Which code is better in average code length?

Huffman codes: Example #3

- Symbols A, B, C, D, E, F are being produced by the information source with probabilities 0.3, 0.4, 0.06, 0.1, 0.1 and 0.04, respectively.
- What is the binary Huffman code?
  - A = 00, B = 1, C = 0110, D = 0100, E = 0101, F = 0111
  - A = 00, B = 1, C = 01000, D = 011, E = 0101, F = 01001
  - A = 11, B = 0, C = 10111, D = 100, E = 1010, F = 10110

Revisiting: Shannon codes

Assume

$$\begin{bmatrix} X \\ p(x) \end{bmatrix} = \left\{ \begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ 0.25 & 0.25 & 0.2 & 0.15 & 0.1 & 0.05 \end{array} \right\}, \sum_i p(x_i) = 1.$$

Then

x_i	$p(x_i)$	i	$p_a(x_i)$	$p_a(x_i)$ binary	l_i	codeword
x_1	0.25	1	0.00	0.00	2	00
x_2	0.25	2	0.25	0.01	2	01
x_3	0.20	3	0.50	0.100	3	100
x_4	0.15	4	0.70	0.101***	3	101
x_5	0.10	5	0.85	0.1101**	4	1101
x_6	0.05	6	0.95	0.11110*	5	11110

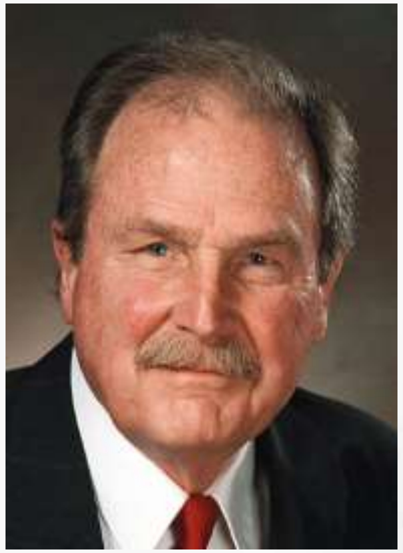
- Source entropy: $H(X) = 2.42$ bits/symbol.
- Shannon codes: Average code length $L = 2.7$ bits/symbol. Code efficiency $\eta = H(X)/L = 89.63\%$.
- What about Huffman codes?

Huffman codes: discussions

- There are no unique Huffman codes.
  - If there are nodes with the same probability, it doesn't matter how they are connected.
  - Assigning 0 and 1 to the branches is arbitrary.

- Every Huffman code has the same average code length!

Why are Huffman codes optimal in average code length?



David Huffman

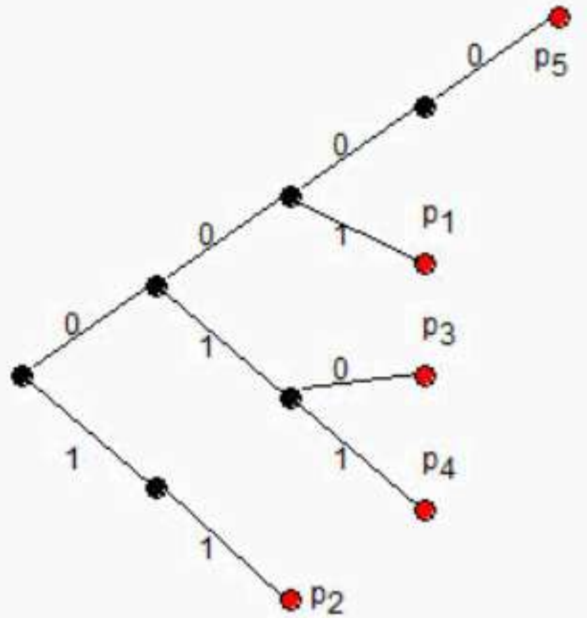
Optimality of Huffman codes

- Lemma: for any distribution, there exists an optimal instantaneous code (with the minimum expected length) that satisfies the following properties:
  1. If $p_j > p_k \Rightarrow l_j \leq l_k$.
  2. The two longest codewords have the same length.
  3. The two longest codewords differ only in the last bit, and correspond to the two least likely symbols.

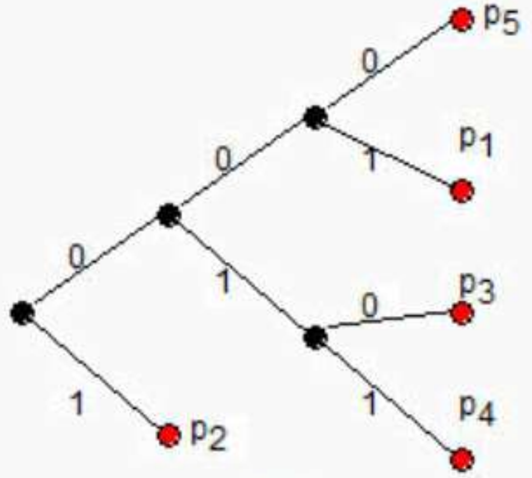
- By this lemma, swap, trim, and rearrange the code tree.

Optimality of Huffman codes

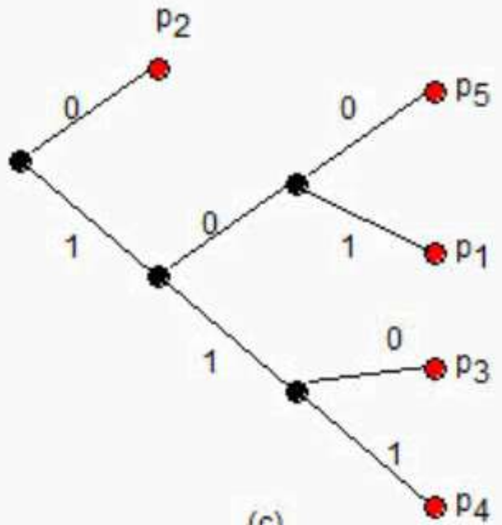
Assume $p_1 \geq p_2 \geq p_3 \geq p_4 \geq p_5$.



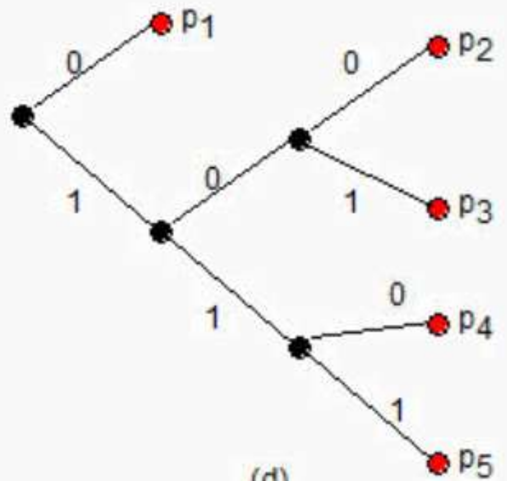
(a)



(b)



(c)



(d)

1. If $p_j > p_k \Rightarrow l_j \leq l_k$.
2. The two longest codewords have the same length.
3. The two longest codewords differ only in the last bit, and correspond to the two least likely symbols.

Optimality of Huffman codes

- Lemma: There are at least two leaf nodes at the end of the longest path of a code tree of a compact instantaneous code, and the probabilities of the source symbols α and β connected to these two leaf nodes have the two minimal probabilities among all source symbols.
- Proof:
  - The probabilities of α and β are p_α and p_β with $p_\alpha \leq p_\beta$.
  - From each node N at the end of the longest path, there are at least two branches.
    - If not, this single branch can be removed without breaking the instantaneous requirement.
  - If there is a source symbol γ with $p_\gamma < p_\beta$, then β and γ can be exchanged, resulting in a smaller average code length. This contradicts the compactness requirement.

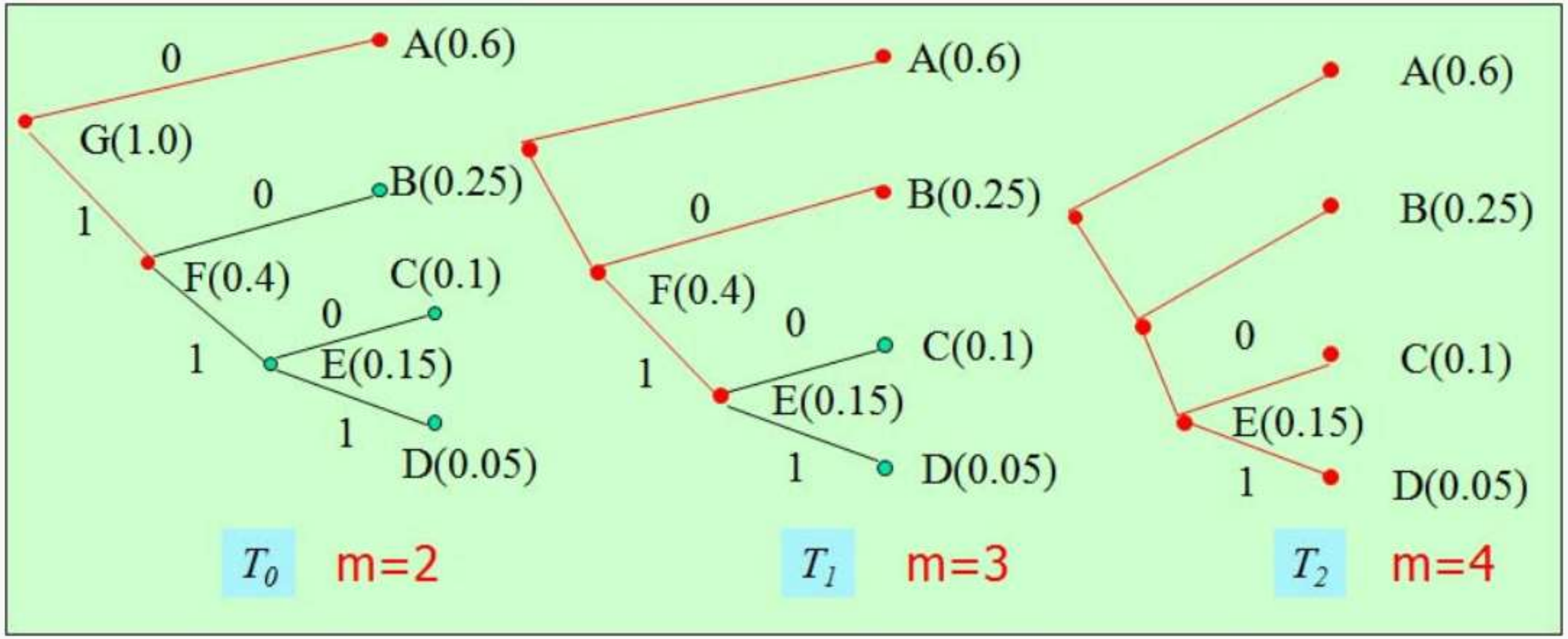
Optimality of Huffman codes

Theorem: All Huffman codes are compact codes.

Observation: The Huffman code construction reduces the number of leaf nodes by taking together the two leaf nodes with the smallest probability (e.g. from $\{A, B, C, D\}$ to $\{A, B, E\}$ to $\{A, F\}$)

- Let's call the final tree T_0 (the complete Huffman tree) and the tree T_i before the i-th iteration of T_0.
- The final tree T_0 is clearly compact, as there are only two branches.
- Proof by induction: Prove that if T_i is compact T_{i+1} is compact.

Optimality of Huffman codes: proof



- The average code lengths L_{i+1} and L_i of T_{i+1} and T_i.
  - Suppose that the leaf nodes in T_{i+1} with the smallest probability have probability p_α and p_β.
  - Taking these together gives $L_{i+1} = L_i + p_\alpha + p_\beta$.

Optimality of Huffman codes: proof

- Suppose T_i is a compact tree, but T_{i+1} is not a compact tree.
- There is a code tree T'_{i+1} with the same nodes as T_{i+1} but with an average code length $L'_{i+1} < L_{i+1}$.
- T'_{i+1} has the same nodes as T_{i+1} and according to the lemma the longest path in T'_{i+1} has two leaf nodes with the smallest source symbol probabilities, which were defined as p_α and p_β.
- Therefore $L'_{i+1} = L'_i + p_\alpha + p_\beta < L_i + p_\alpha + p_\beta = L_{i+1}$.
- Thus, T_i is not compact.
- Contradiction! So T_{i+1} must be a compact tree.

Optimality of Huffman codes: discussions

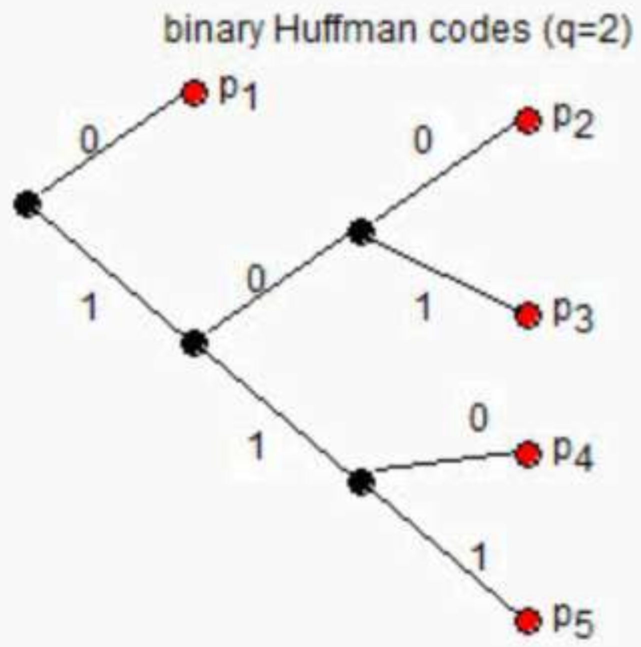
- Huffman codes are optimal in the expected codeword length: for Huffman code C^* and any other uniquely decodable code C':

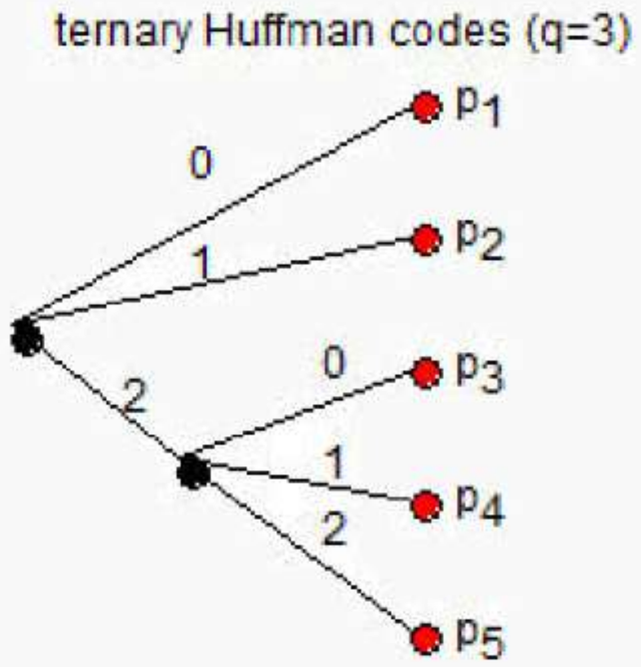
$$\boxed{L(C^*) \le L(C').}$$

- Does it mean that the codeword lengths for a Huffman code are always less than the Shannon code?
  - Input symbols A, B, C, D with probability 1/3, 1/3, 1/4 and 1/12.

- The above discussions are all based on binary Huffman codes
  - What about Q-ary Huffman codes?

Q-ary Huffman codes





Q	*Q*-ary code
2	Binary
3	Ternary
4	Quaternary
5	Quinary
8	Octal
10	Decimal
16	Hexadecimal

- Q-ary Huffman codes are constructed in the same way as binary Huffman codes.
- Instead of two leaf nodes, take the *q* leaf nodes with the smallest probability

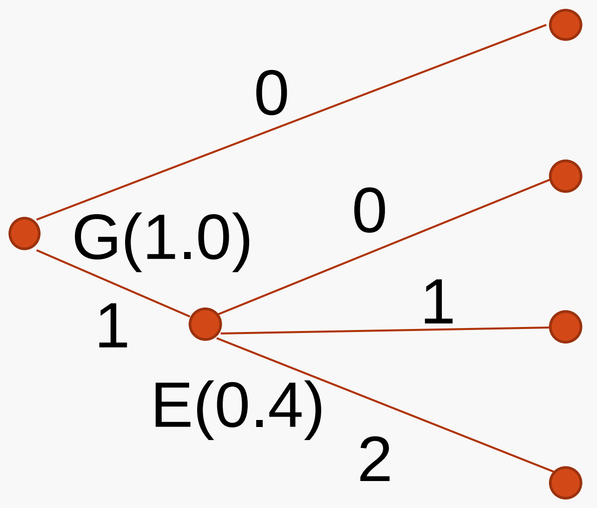
Q-ary Huffman codes: example

- Construct a Ternary Huffman code for the following source.

$$\begin{bmatrix} X \\ P \end{bmatrix} = \begin{bmatrix} A & B & C & D \\ 0.05 & 0.25 & 0.1 & 0.6 \end{bmatrix}$$

$$H(X) = -\sum_{x\in\mathcal{X}} p(x)\log[p(x)]$$

$$\boxed{L(C) = \sum_{x\in\mathcal{X}} p(x)l(x)}$$

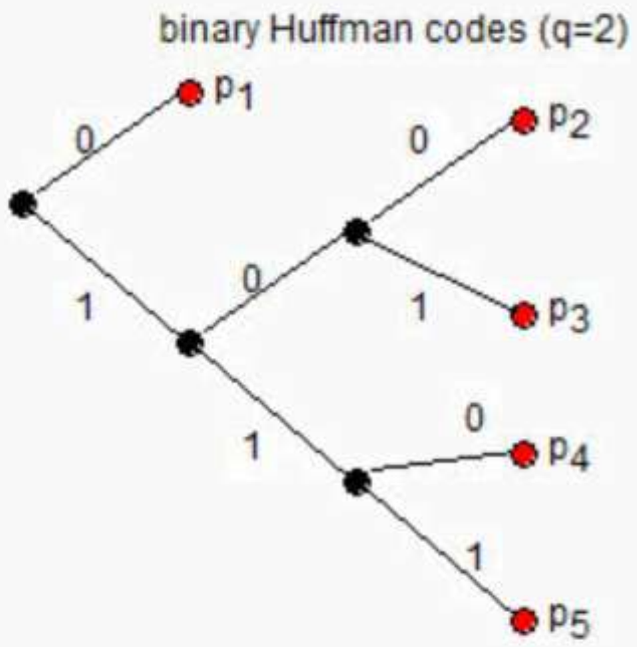


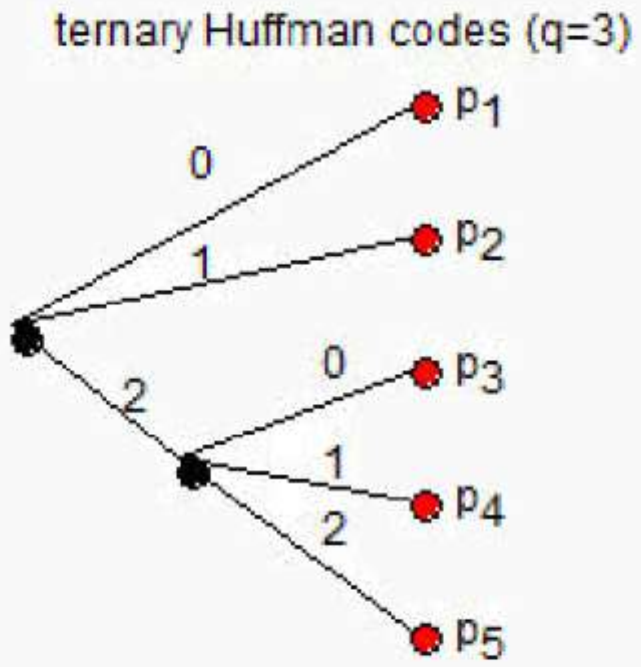
	Codeword
D (0.6)	0
B (0.25)	10
C (0.1)	11
A (0.05)	12

- Average code length *L*=1.4

Can you do better?

Q-ary Huffman codes: algorithm





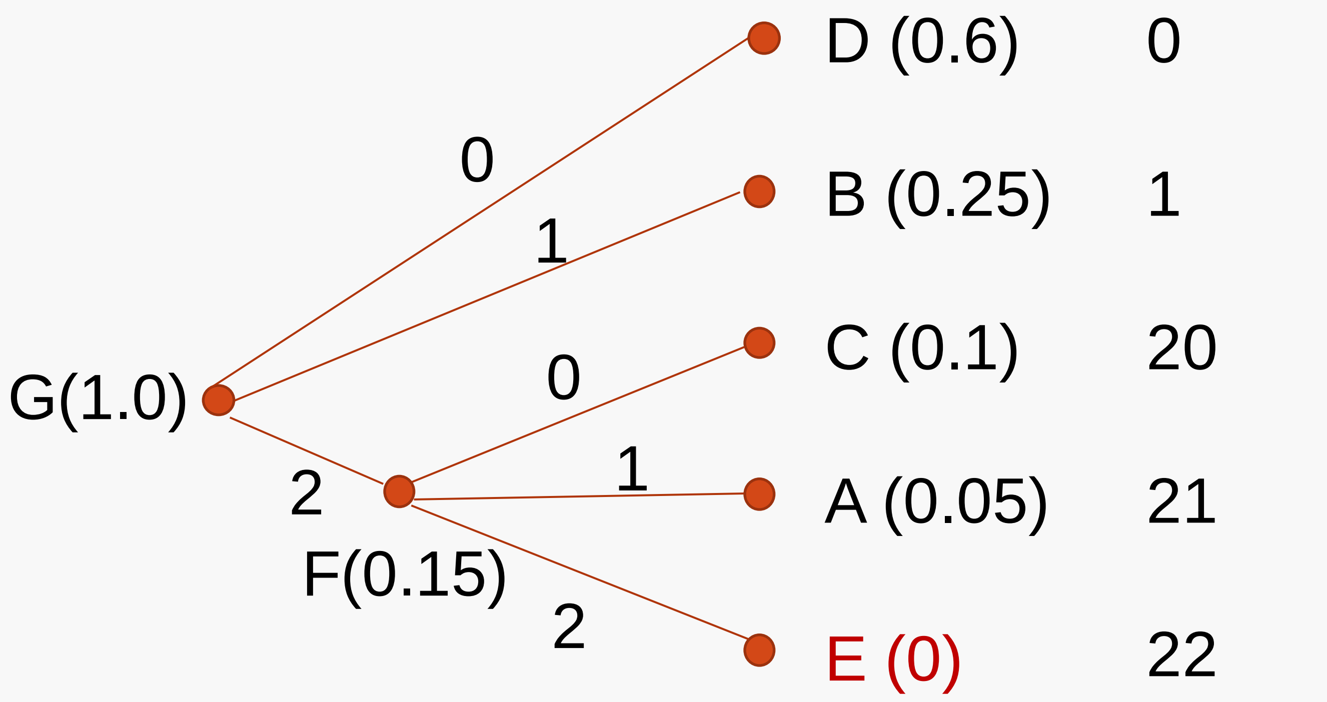
- We should take advantage of **all the shortest codes**.
- To take full advantage of the shortest codes, the final tree should have *q* leaf nodes.
- If there are less than ***(q − 1)m + q*** source symbols for some positive integer m, **“dummy" symbols with probability 0 must be added**.

Q-ary Huffman codes: example

- Construct a Ternary Huffman code for the following source.

$$\begin{bmatrix} X \\ P \end{bmatrix} = \begin{bmatrix} A & B & C & D \\ 0.05 & 0.25 & 0.1 & 0.6 \end{bmatrix}$$

We need to have ***(q − 1)m + q*** symbols.



- Average code length L=1.15

Why is dummy symbol necessary?

Q-ary Huffman codes: example

- Construct a Quaterary Huffman code for the following source.

$$\begin{bmatrix} X \\ p(x) \end{bmatrix} = \begin{Bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\ 0.3 & 0.2 & 0.15 & 0.12 & 0.1 & 0.05 & 0.05 & 0.03 \end{Bmatrix}.$$

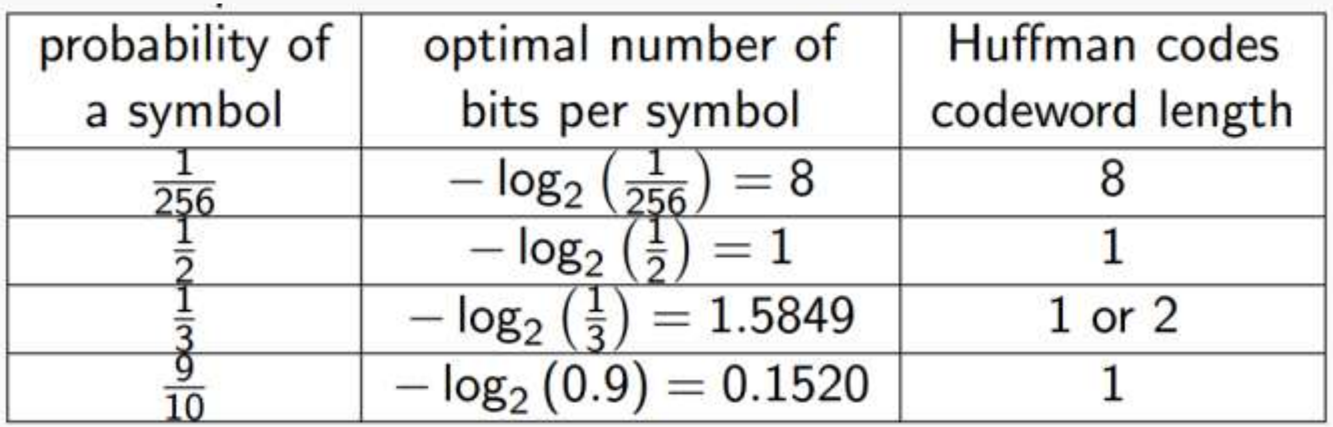
- Does it satisfy the Kraft inequality?

- Does it satisfy Shannon's first theorem?

- What is the coding efficiency?

Limitations of Huffman codes

- **Quantization effect**
  - Huffman codes have to be an integer of bits long. At most 1 bit overhead.
  - For those high probability symbol in common set, or for small set, Huffman coding would use much longer codeword length than that is necessary.



probability of a symbol	optimal number of bits per symbol	Huffman codes codeword length
$\frac{1}{256}$	$-\log_2\left(\frac{1}{256}\right) = 8$	8
$\frac{1}{2}$	$-\log_2\left(\frac{1}{2}\right) = 1$	1
$\frac{1}{3}$	$-\log_2\left(\frac{1}{3}\right) = 1.5849$	1 or 2
$\frac{9}{10}$	$-\log_2(0.9) = 0.1520$	1

- Improvements: (see chapter 13.)
  - Arithmetic coding: remove the quantization effect from which Huffman codes suffers with small source alphabets.

Limitations of Huffman codes

- Need to have the knowledge of **the statistics of information source.**
    - **Difficult to obtain in practice**

- Improvements: (see chapter 13.)
    - Universal coding: achieve related good length without the knowledge of the source, such as LZ codes..

本节学习目标

1. 写出Shannon code的算法流程
2. 能够编写Shannon code
3. 说出为什么Shannon code不是compact code
4. 写出Huffman code的算法流程
5. 能够编写Huffman code
6. 说出Huffman code的≥3个特点
7. 证明Huffman code的最优性
8. 能够编写Q-ary Huffman code
9. 说出Huffman code的2个局限性

重难点：

- Shannon code
- Huffman code

Summary of Chapter 3

- **Motivation**
  - Idea: eliminate redundancy to compress data
  - Source coding: encoder and decoder
  - Optimal codes: the instantaneous code with the minimum expected length

- **Theory**: Zero-error source coding theorem
  - The theoretical limit: entropy of the source
  - The existence of ideal source codes

- **Applications**: Practical source coding algorithms
  - Shannon codes
  - Huffman codes

Thank you!

My Homepage

Yayu Gao

School of Electronic Information and Communications

Huazhong University of Science and Technology

Email: yayugao@hust.edu.cn