СӘТБАЕВ УНИВЕРСИТЕТІ SATBAYEV UNIVERSITY

Efficient Coding. Huffman and Shannon-Fano Methods

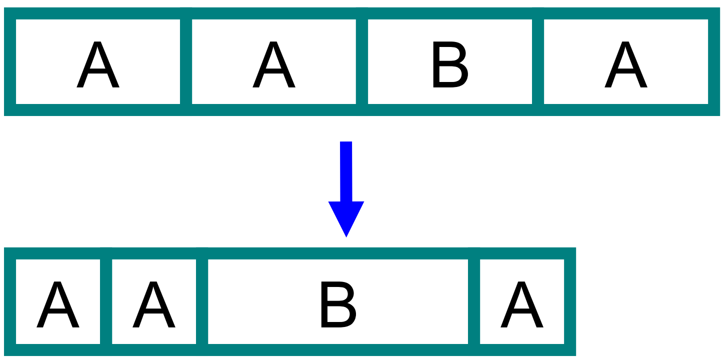
Dosbayev Zhandos Makhsutuly, senior lecturer
E-mail: zh.dosbayev@satbayev.university

Outline

Efficient Coding
Huffman coding
Shennon Fano coding

Huffman Code

- Approach
  - Variable length encoding of symbols
  - Exploit statistical frequency of symbols
  - Efficient when symbol probabilities vary widely
- Principle
  - Use fewer bits to represent frequent symbols
  - Use more bits to represent infrequent symbols



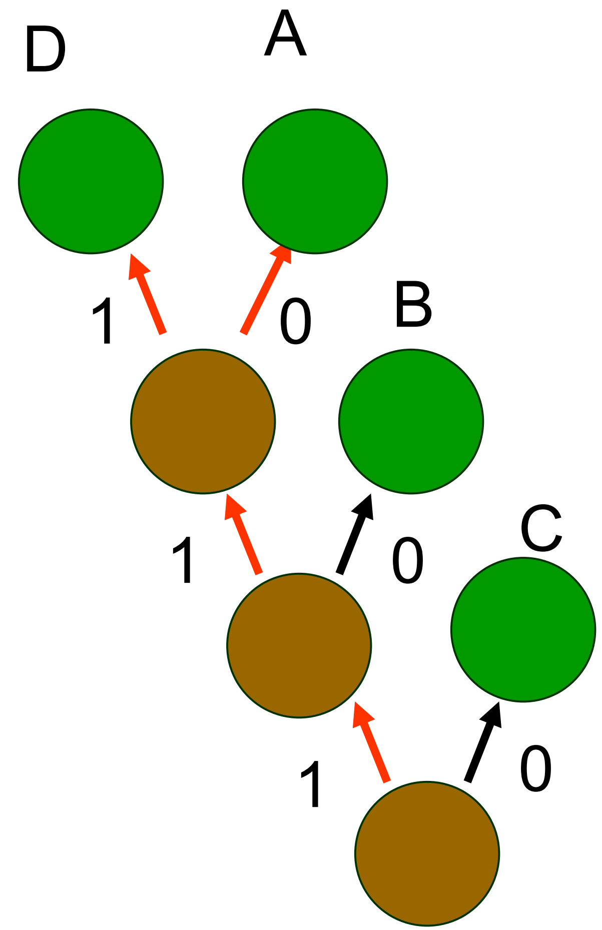
Huffman Code Example

Symbol	A	B	C	D
Frequency	13%	25%	50%	12%
Original Encoding	00	01	10	11
	2 bits	2 bits	2 bits	2 bits
Huffman Encoding	110	10	0	111
	3 bits	2 bits	1 bit	3 bits

- Expected size
  - Original $\Rightarrow$ $1/8 \times 2 + 1/4 \times 2 + 1/2 \times 2 + 1/8 \times 2 = 2$ bits / symbol
  - Huffman $\Rightarrow$ $1/8 \times 3 + 1/4 \times 2 + 1/2 \times 1 + 1/8 \times 3 = 1.75$ bits / symbol

Huffman Code Data Structures

- Binary (Huffman) tree
  - Represents Huffman code
  - Edge ⇒ code (0 or 1)
  - Leaf ⇒ symbol
  - Path to leaf ⇒ encoding
  - Example
    - A = “110”, B = “10”, C = “0”
- Priority queue
  - To efficiently build binary tree



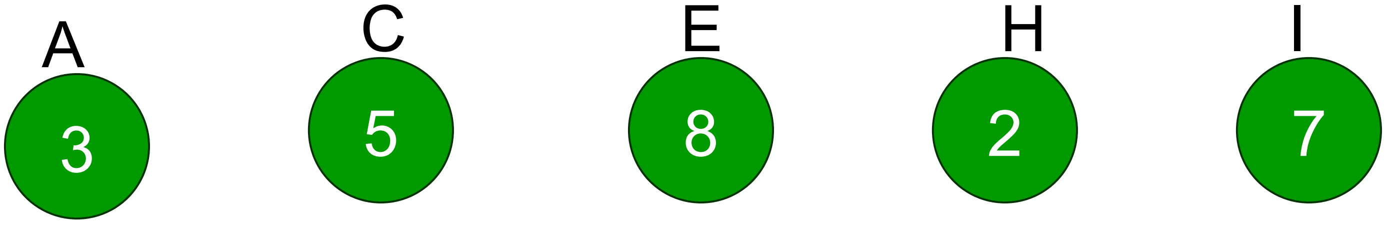
Huffman Code Algorithm Overview

- Encoding
  - Calculate frequency of symbols in file
  - Create binary tree representing “best” encoding
  - Use binary tree to encode compressed file
    - For each symbol, output path from root to leaf
    - Size of encoding = length of path
  - Save binary tree

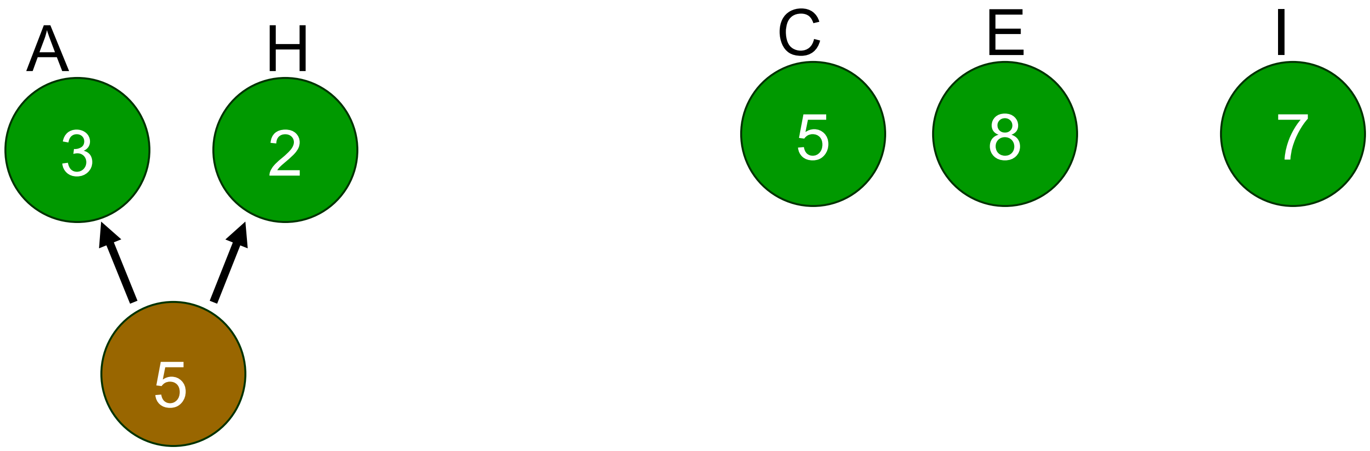
Huffman Code – Creating Tree

- Algorithm
  - Place each symbol in leaf
    - Weight of leaf = symbol frequency
  - Select two trees L and R (initially leafs)
    - Such that L, R have lowest frequencies in tree
  - Create new (internal) node
    - Left child $\Rightarrow$ L
    - Right child $\Rightarrow$ R
    - New frequency $\Rightarrow$ frequency(L) + frequency(R)
  - Repeat until all nodes merged into one tree

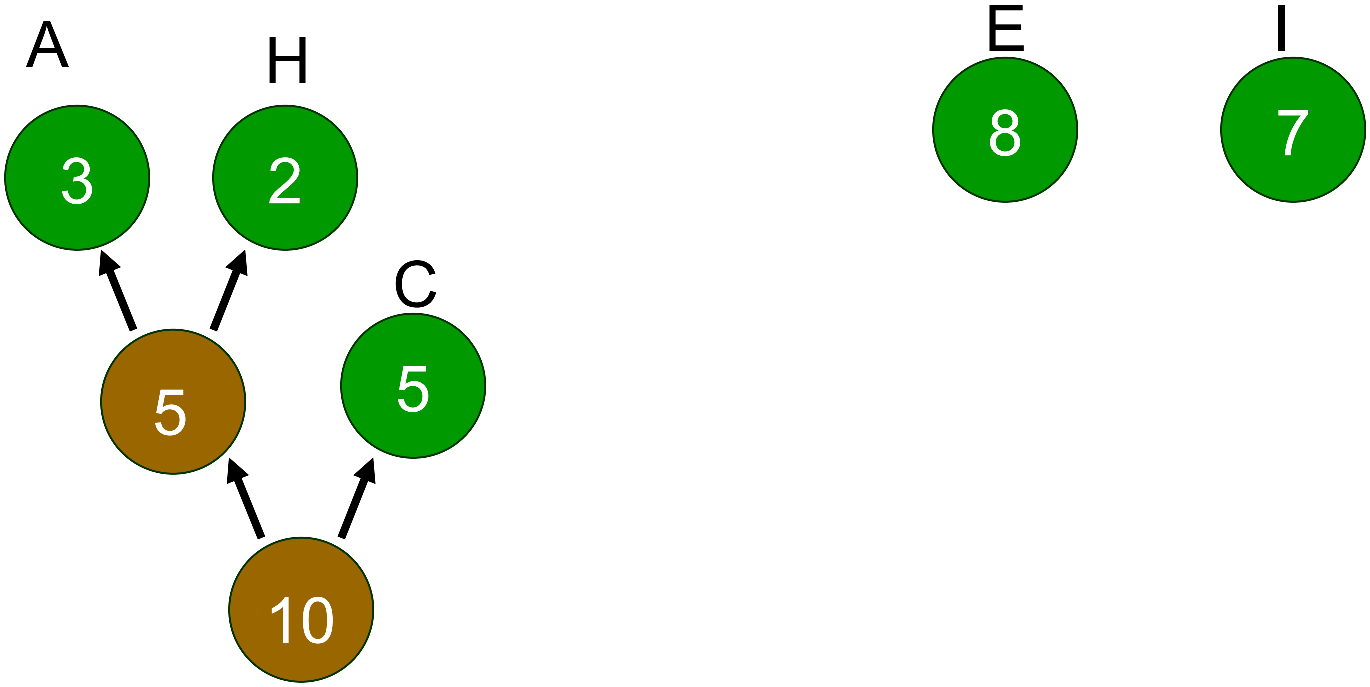
Huffman Tree Construction 1



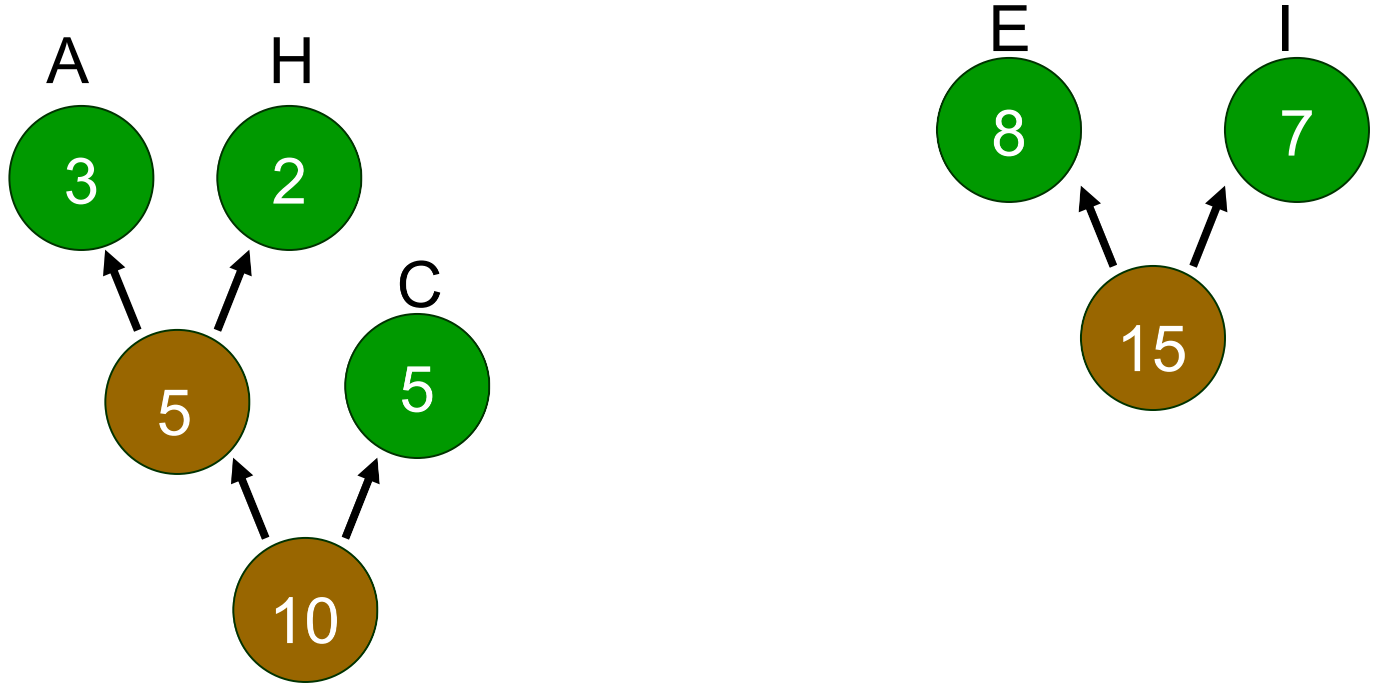
Huffman Tree Construction 2



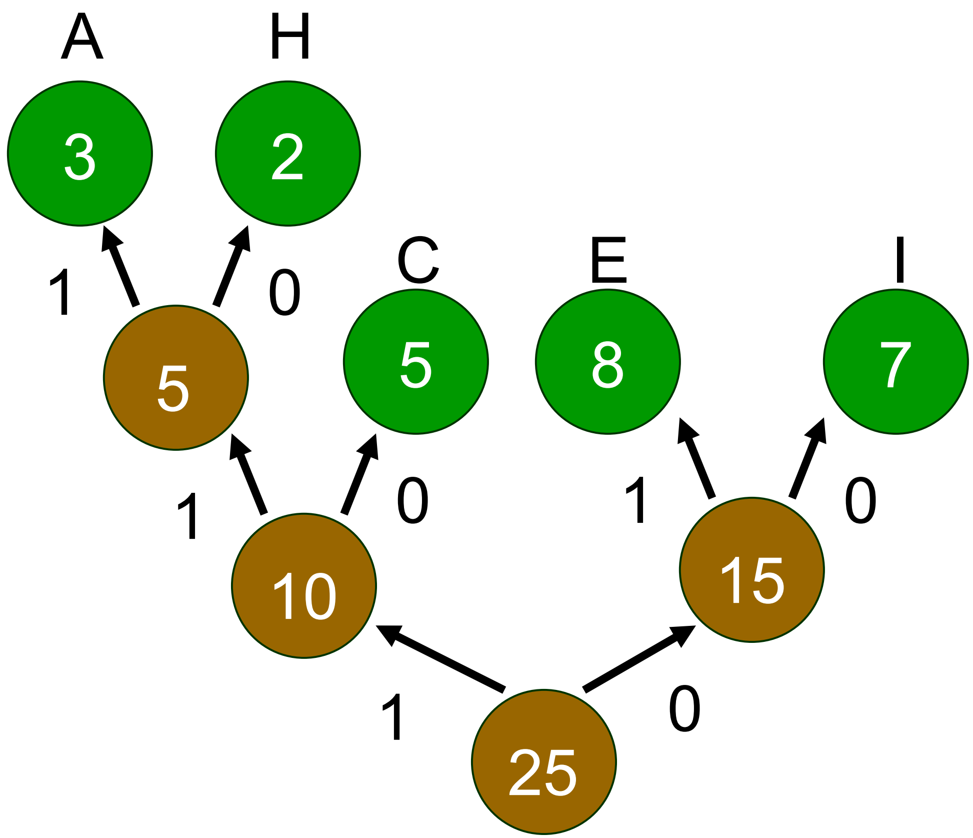
Huffman Tree Construction 3



Huffman Tree Construction 4



Huffman Tree Construction 5



E = 01
I = 00
C = 10
A = 111
H = 110

Huffman Coding Example

- Huffman code

E	=	01
I	=	00
C	=	10
A	=	111
H	=	110

- Input
  - ACE
- Output
  - (111)(10)(01) = 1111001

Shannon-Fano Coding

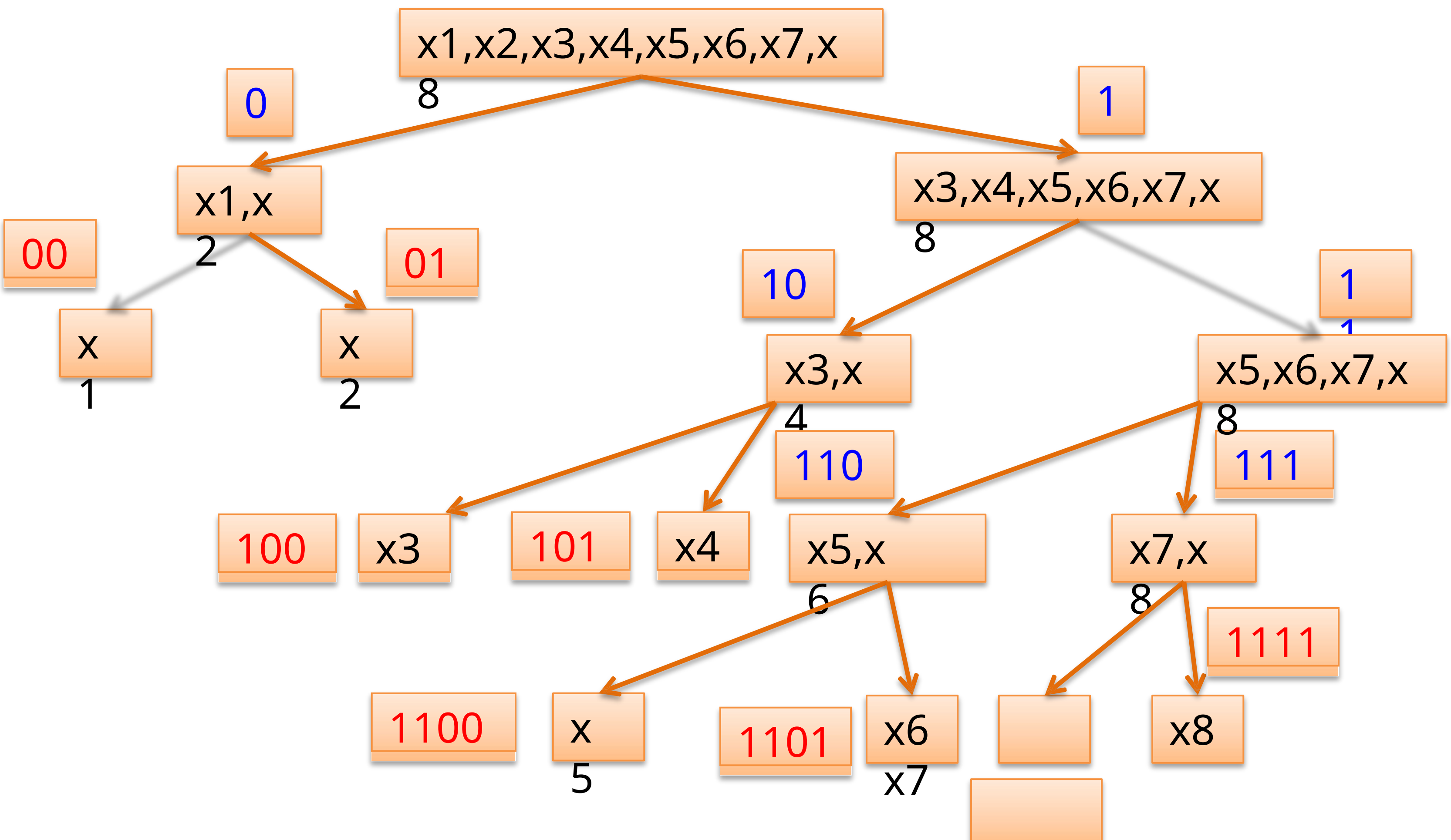
- An efficient code can be obtained by the following simple algorithm as steps given below:
- Shannon-Fano Algorithm
  - The letters (messages) of (over) the input alphabet must be arranged in order from most probable to least probable.
  - Then the initial set of messages must be divided into two subsets whose total probabilities are as close as possible to being equal.

Shannon-Fano Coding

- All symbols then have the first digits of their codes assigned; symbols in the first set receive "0" and symbols in the second set receive "1".
- The same process is repeated on those subsets, to determine successive digits of their codes, as long as any sets with more than one member remain.
- When a subset has been reduced to one symbol, this means the symbol's code is complete.

Shannon-Fano Coding: Example

Message	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
Probability	**0.25**	**0.25**	**0.125**	**0.125**	**0.0625**	**0.0625**	**0.0625**	**0.0625**



Shannon-Fano Coding: Example

Message	x_1	x_2	x_3	x_4	x_5	x_6	x_7	X_8
Probability	**0.25**	**0.25**	**0.125**	**0.125**	**0.0625**	**0.0625**	**0.0625**	**0.0625**
Encoding vector	**00**	**01**	**100**	**101**	**1100**	**1101**	**1110**	**1111**

- **Entropy** $H = -\left(2 \cdot \left(\frac{1}{4} \log \frac{1}{4}\right) + 2 \cdot \left(\frac{1}{8} \log \frac{1}{8}\right) + 4 \cdot \left(\frac{1}{16} \log \frac{1}{16}\right)\right) = 2.75$
- **Average length of the encoding vector**
  $L = \sum P\{x^i\} n_i = \left(2 \cdot \left(\frac{1}{4} \cdot 2\right) + 2 \cdot \left(\frac{1}{8} \cdot 3\right) + 4 \cdot \left(\frac{1}{16} \cdot 4\right)\right) = 2.75$
- **The Shannon-Fano code gives 100% efficiency**

Shannon-Fano Encoding: Example

Message	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
Probability	**0.25**	**0.25**	**0.125**	**0.125**	**0.0625**	**0.0625**	**0.0625**	**0.0625**
Encoding vector	**00**	**01**	**100**	**101**	**1100**	**1101**	**1110**	**1111**

- The Shannon-Fano code gives 100% efficiency. Since the average length of the encoding vector for this code is 2.75 bits, it gives the 0.25 bits/symbol compression, while the direct uniform binary encoding (3 bits/symbol) is redundant.

Shannon-Fano Encoding: Properties

- It should be taken into account that the Shannon-Fano code is not unique because it depends on the partitioning of the input set of messages, which, in turn, is not unique.
- If the successive equiprobable partitioning is not possible at all, the Shannon-Fano code may not be an optimum code, that is, a code that leads to the lowest possible average length of the encoding vector.