# Shannon – Fano Algorithm for Data Compression

Multimedia Technology

Tutorial 1, section 3

# Shannon – Fano Theory Reminder

- Shannon-Fano coding is a lossless data compression algorithm for multimedia.
  - No data is lost during compression.
  - With lossy compression, we have size reduction by removing unnecessary information.
- It assigns codes to symbols based on their probabilities of occurrence, with more frequent symbols receiving shorter codes.
  - Code length is closely related to the information content (or entropy) of each symbol.
- Why do we need data compression?
  - To optimize the representation of data for efficient transmission and storage.

\- For more information refer to lecture slides.

# Shannon – Fano Algorithm

| | |
|---|---|
| **Step 1:** | **List probabilities** or frequency counts for the symbols to determine their relative frequency of occurrence. |
| **Step 2:** | **Sort symbols** in decreasing order of probability (most probable on the left, least probable on the right). |
| **Step 3:** | **Split the list** into two parts such that the total probability of both parts is as close as possible. |
| **Step 4:** | **Assign 0 to the left part** and **1 to the right part**. |
| **Step 5:** | **Repeat the split** for each part until all symbols are split into individual subgroups. (repeat steps 3 and 4) |

# Shannon – Fano Coding Tree Exercise

Let the alphabet {‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’}, with the following character probabilities: P(a) = 0.25, P(b) = 0.15, P(c) = 0.30, P(d) = 0.10, P(e) = 0.15, and P(f) = 0.05.

Construct a Shannon-Fano coding tree corresponding to this alphabet and calculate the average length of the resulting code.

> Note: In Shannon – Fano, we only <u>sort</u> the nodes at the <u>beginning</u> of the algorithm

# Shannon – Fano Algorithm

**Step 1:** **List probabilities** or frequency counts for the symbols to determine their relative frequency of occurrence.

| Symbol | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| Probability or Frequency | 0.25 | 0.15 | 0.30 | 0.10 | 0.15 | 0.05 |

**Step 2:** **Sort symbols** in decreasing order of probability (most probable on the left, least probable on the right).

| Symbol | c | a | b | e | d | f |
|---|---|---|---|---|---|---|
| Probability or Frequency | 0.30 | 0.25 | 0.15 | 0.15 | 0.10 | 0.05 |

**Step 3:** **Split the list** into two parts such that the total probability of both parts is as close as possible.

| Symbol | c | a | b | e | d | f |
|---|---|---|---|---|---|---|
| Probability or Frequency | 0.30 | 0.25 | 0.15 | 0.15 | 0.10 | 0.05 |

0 → left part; 1 → right part

Total Probability = 0.55

| Symbol | c | a |
|---|---|---|
| Probability or Frequency | 0.30 | 0.25 |

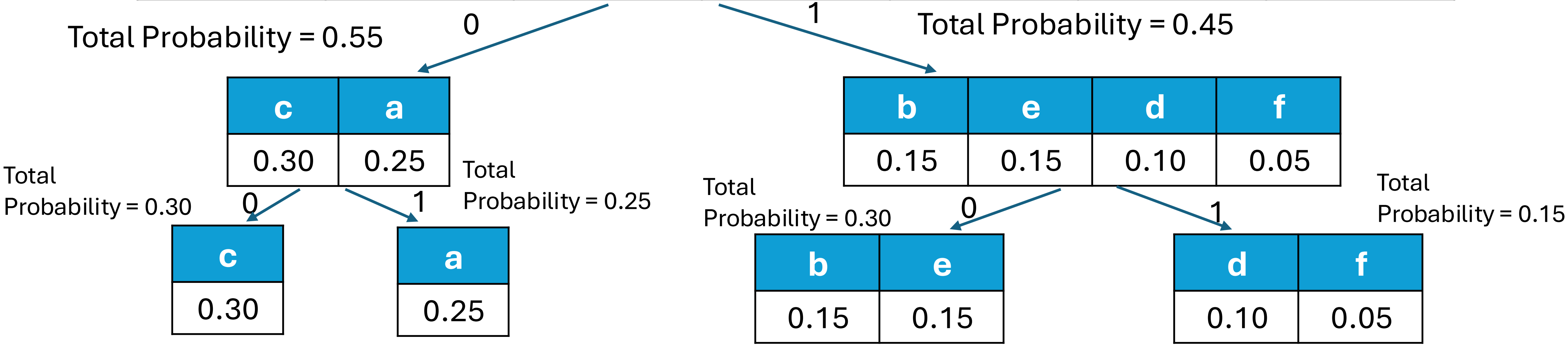
Total Probability = 0.45

| Symbol | b | e | d | f |
|---|---|---|---|---|
| Probability or Frequency | 0.15 | 0.15 | 0.10 | 0.05 |

**Step 4:** **Assign 0 to the left part** and **1 to the right part**.

**Step 3:** **Split the list** into two parts such that the total probability of both parts is as close as possible.

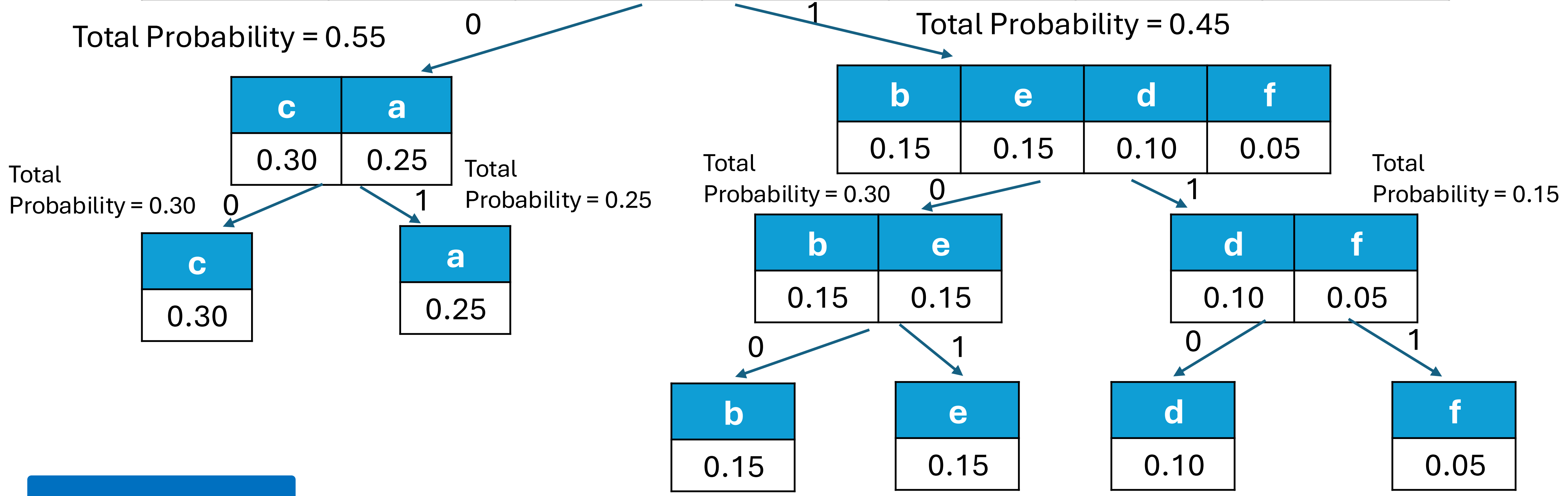
| Symbol | c | a | b | e | d | f |
|---|---|---|---|---|---|---|
| Probability or Frequency | 0.30 | 0.25 | 0.15 | 0.15 | 0.10 | 0.05 |



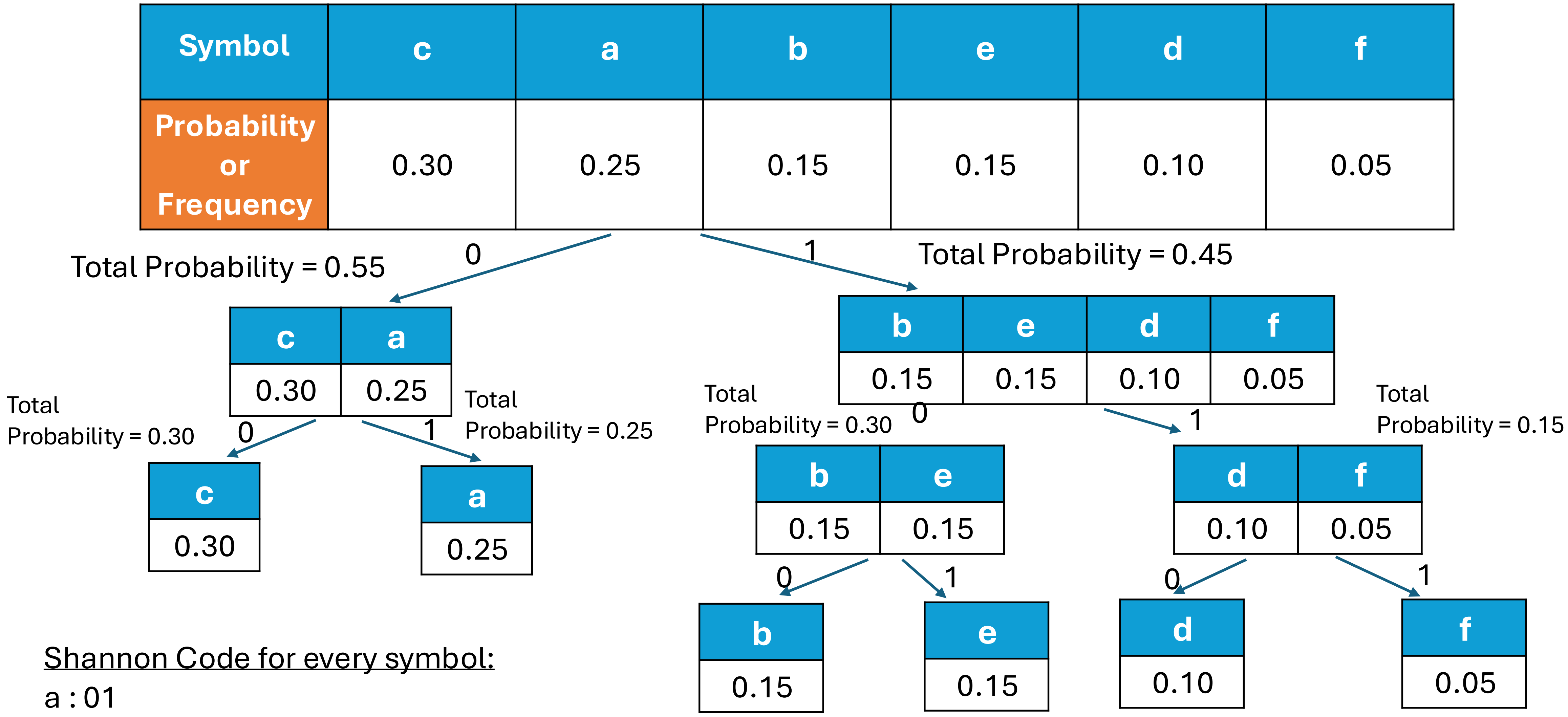

**Step 4:** **Assign 0 to the left part** and **1 to the right part**.

**Step 3:** **Split the list** into two parts such that the total probability of both parts is as close as possible.

| Symbol | c | a | b | e | d | f |
|---|---|---|---|---|---|---|
| Probability or Frequency | 0.30 | 0.25 | 0.15 | 0.15 | 0.10 | 0.05 |




**Step 4:** **Assign 0 to the left part** and **1 to the right part**.




<u>Shannon Code for every symbol:</u>

a : 01
b : 100
c : 00
d : 110
e : 101
f : 111

# Average length of the resulting code

P(a)*2bit + P(b)*3bit + P(c)*2bit + P(d)*3bit + P(e)*3bit + P(f)*3bit =

= 0.25 * 2 + 0.15 * 3 + 0.30 * 2 + 0.1 * 3 + 0.15 * 3 + 0.05 * 3 =

3 * 0.45 + 2 * 0.55 = 1.35 + 1.1 = 2.45

Shannon Code for every symbol:

w(a) = 01
w(b) = 100
w(c) = 00
w(d) = 110
w(e) = 101
w(f) = 111

Average Length = $\Sigma\, p(s_i)$ * number_of_bits, for i={0, 1,.., N}, where N the total number of symbols