Huffman Algorithm for Data Compression

Multimedia Technology

Tutorial 1, section 4b

Huffman Theory Reminder

- Huffman coding is a lossless data compression algorithm for multimedia.
  - No data is lost during this compression.
- It assigns codes to characters based on their frequencies, ensuring no code is a prefix of another to avoid ambiguity during decoding.
  - Code length is closely related to the information content (or entropy) of each symbol.
- The algorithm involves building a Huffman Tree and traversing it to assign efficient binary codes to each character.

Huffman Algorithm

Step 1: **Create a leaf node for each character** and build a **min-heap** based on their frequencies.

Step 2: **Extract** the **two nodes** with the **lowest frequencies** from the heap.

Step 3: **Create an internal node** with a **frequency equal to the sum** of the **two extracted nodes**. Set the first as the left child and the second as the right child. **Insert** this new node back **into the heap**.

Step 4: **Repeat steps 2 and 3** until only one node remains in the heap, which becomes the root of the Huffman Tree.

Huffman Coding Tree Exercise

Let the alphabet {‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’}, with the following character probabilities: P(a) = 0.05, P(b) = 0.25, P(c) = 0.25, P(d) = 0.15, P(e) = 0.10, and P(f) = 0.20.

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

Note: In Huffman coding, the nodes are sorted again at each step!

Huffman Algorithm

a	0.05	b	0.25	c	0.25	d	0.15	e	0.10	f	0.20

Step 1: **Create a leaf node for each character** and build a **min-heap** based on their frequencies.

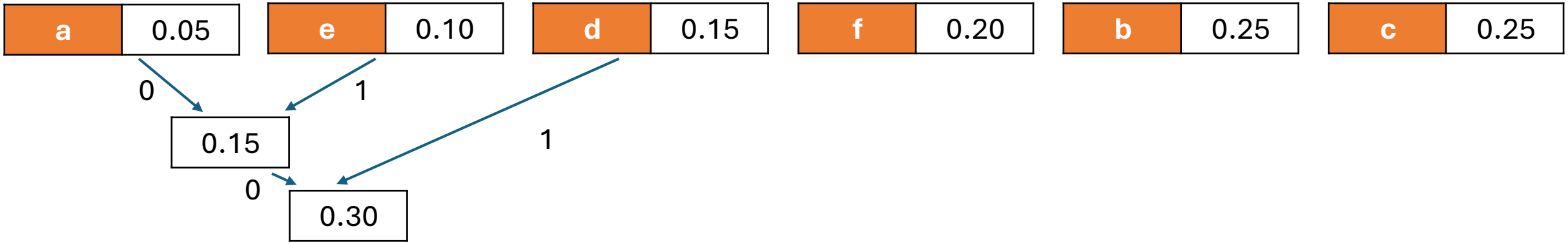
Step 2: **Extract** the **two nodes** with the **lowest frequencies** from the heap.

Step 3: **Create an internal node** with a **frequency equal to the sum** of the **two extracted nodes**. Set the first as the left child and the second as the right child. **Insert** this new node back **into the heap**.

Huffman Algorithm

a	0.05	b	0.25	c	0.25	d	0.15	e	0.10	f	0.20

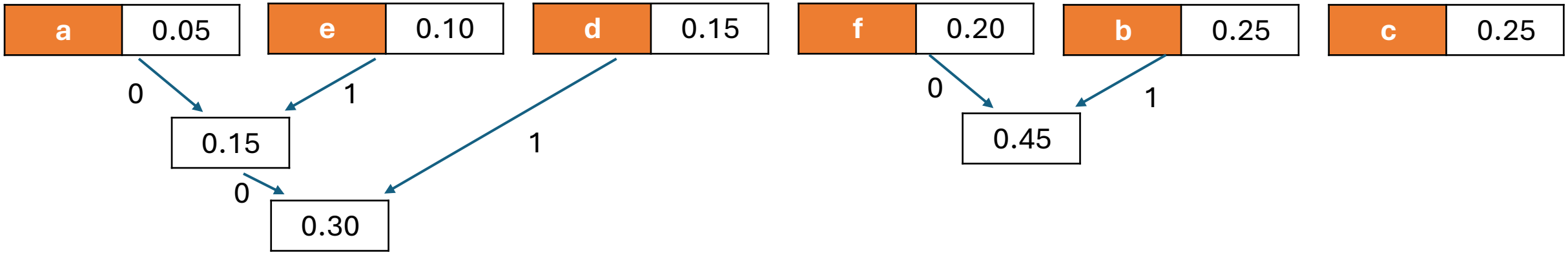
Step 1: **Create a leaf node for each character** and build a **min-heap** based on their frequencies.



Step 2: **Extract** the **two nodes** with the **lowest frequencies** from the heap.

Step 3: **Create an internal node** with a **frequency equal to the sum** of the **two extracted nodes**. Set the first as the left child and the second as the right child. **Insert** this new node back **into the heap**.

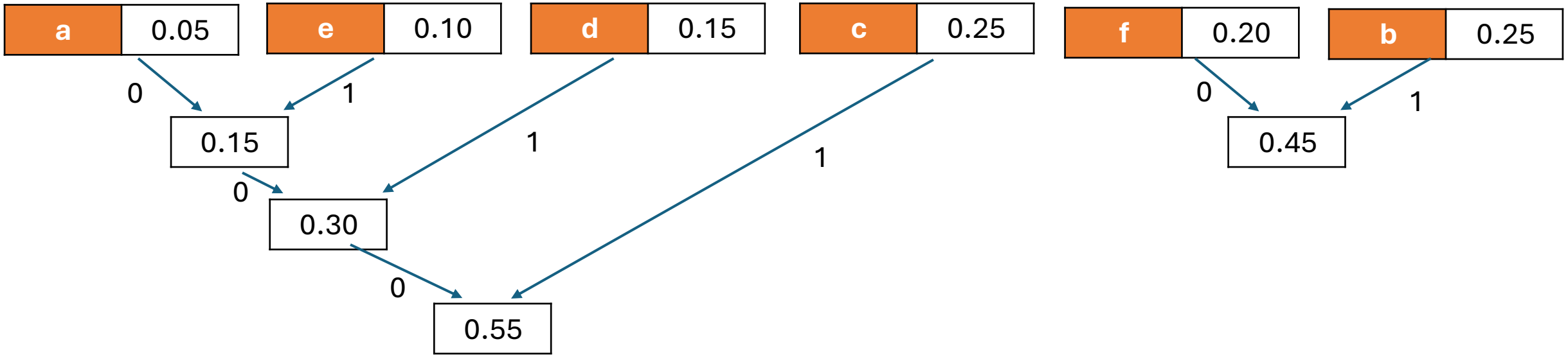
Huffman Algorithm



Step 2: **Extract** the **two nodes** with the **lowest frequencies** from the heap.

Step 3: **Create an internal node** with a **frequency equal to the sum** of the **two extracted nodes**. Set the first as the left child and the second as the right child. **Insert** this new node back **into the heap**.

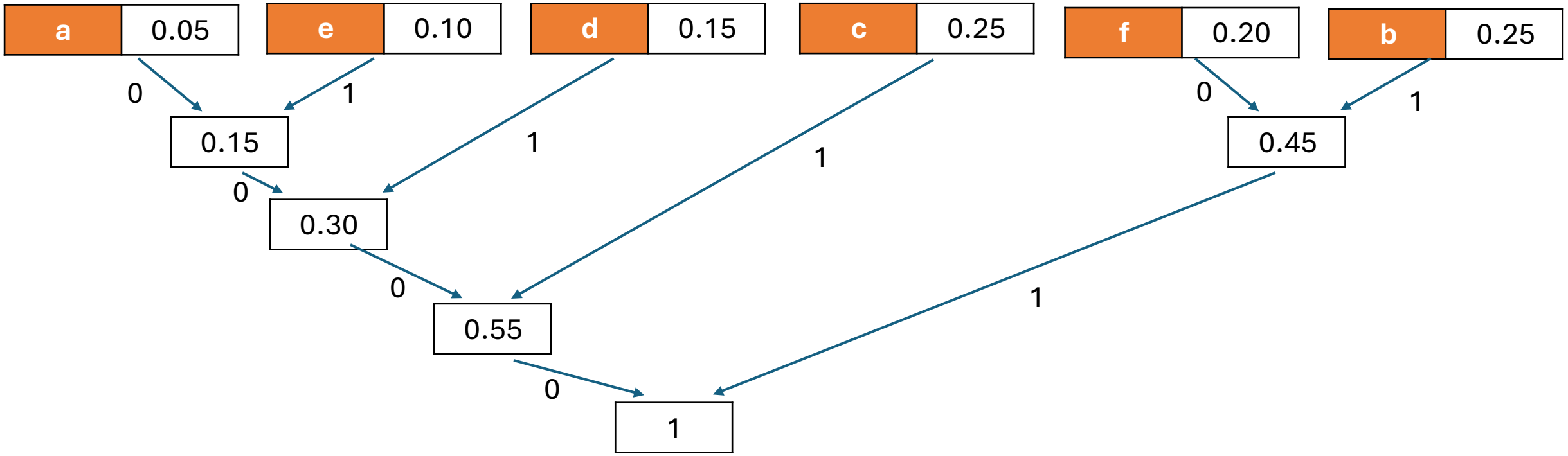
Huffman Algorithm



Step 2: **Extract** the **two nodes** with the **lowest frequencies** from the heap.

Step 3: **Create an internal node** with a **frequency equal to the sum** of the **two extracted nodes**. Set the first as the left child and the second as the right child. **Insert** this new node back **into the heap**.

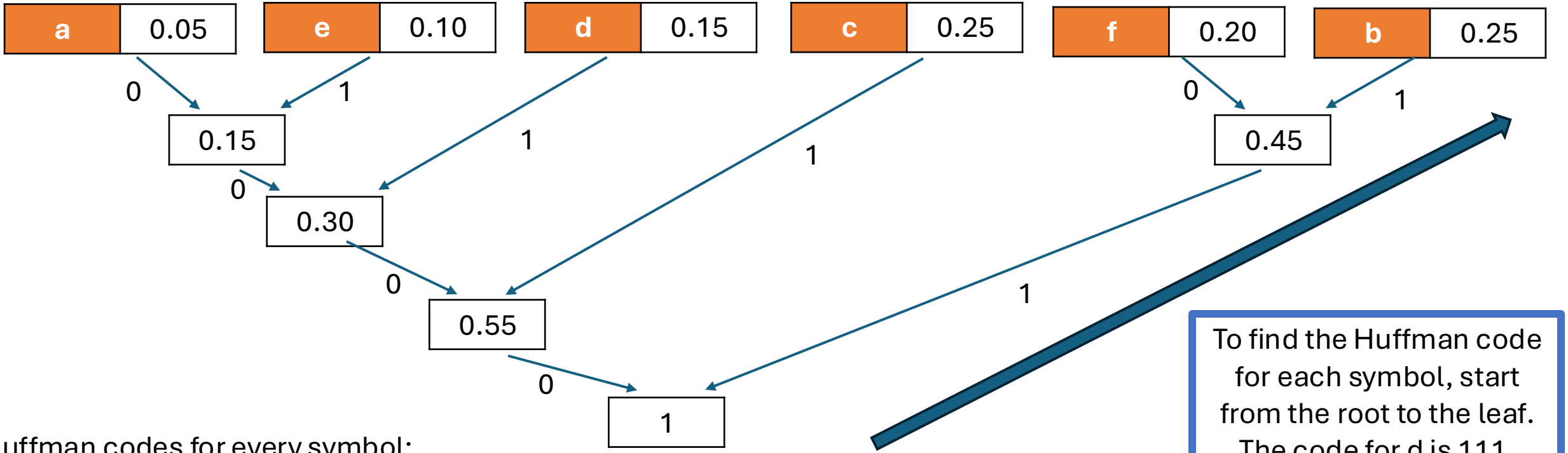
Huffman Algorithm



Step 2: **Extract** the **two nodes** with the **lowest frequencies** from the heap.

Step 3: **Create an internal node** with a **frequency equal to the sum** of the **two extracted nodes**. Set the first as the left child and the second as the right child. **Insert** this new node back **into the heap**.

Huffman Algorithm



To find the Huffman code for each symbol, start from the root to the leaf. The code for d is 111.

Huffman codes for every symbol:

a : 0000
b : 11
c : 01
d : 001
e : 0001
f : 10

Average length of the resulting code

P(a)*4bit + P(b)*2bit + P(c)*2bit + P(d)*3bit + P(e)*4bit + P(f)*2bit =

0.05 * 4 + 0.25 * 2 + 0.25 * 2 + 0.15 * 3 + 0.10 * 4 + 0.20 * 2 =

4 * 0.15 + 3 * 0.15 + 2 * 0.70 = 0.60 + 0.45 + 1.4 = 2.45

Huffman codes for every symbol:

w(a) = 0000
w(b) = 11
w(c) = 01
w(d) = 001
w(e) = 0001
w(f) = 10

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