CMPN206: Multimedia

Lecture 3: Golomb and Arithmetic Coding

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Spring 2014

Agenda

- Golomb Coding
- Arithmetic Coding

Acknowledgments: Most slides are adapted from Richard Ladner and from Li and Drew.

Unary Code

- Used to represent *non-negative integers*
- A non-negative integer $n >= 0$ is represented as n 1's followed by a 0
- Examples:
  - Code for 4 is 11110
  - Code for 7 is 1111 1110
  - Code for 0 is 0
- The unary code is actually optimal (similar to Huffman Coding) for the alphabet {1, 2, 3, … } with probability

$$P(k) = \frac{1}{2^k}$$

- Usually used together with other codes e.g. Golomb Codes

Golomb Codes

- Used to represent *positive integers*
- Parametrized by an integer $m > 0$
- An integer $n > 0$ is represented by two numbers q and r :

$$q = \lfloor \frac{n}{m} \rfloor \qquad r = n - qm$$

  q is the quotient and r is the remainder

- q takes on values $\{0, 1, 2, \ldots\}$ and is represented in *unary*
- r takes on values $\{0, 1, \ldots, m-1\}$ and is represented in binary using $\lfloor \log_2 m \rfloor$ bits or $\lceil \log_2 m \rceil$ bits using a fixed prefix code:
  - The first $2^{\lceil \log_2 m \rceil} - m$ values are represented using $2^{\lfloor \log_2 m \rfloor}$ bits
  - The remaining values are represented by the $2^{\lceil \log_2 m \rceil}$-bit representation of $r + 2^{\lceil \log_2 m \rceil} - m$

Example

$m = 5 \qquad 2^{\lceil \log_2 m \rceil} = 8 \qquad 2^{\lfloor \log_2 m \rfloor} = 4$

- The first $8 - 5 = 3$ values of r will be represented by the 2-bit binary representation of r
- The next 5 values of r will be represented by the 3-bit binary representation of $r + 3$
- The quotient q is always represented in unary
- The codeword for 3 is:

  $3 = 0\,m + 3$ i.e. $q = 0$ & $r = 3$

  0110 (0 in unary and 3 + 3 = 6 in 3-bit binary)
- The codeword for 21 is:

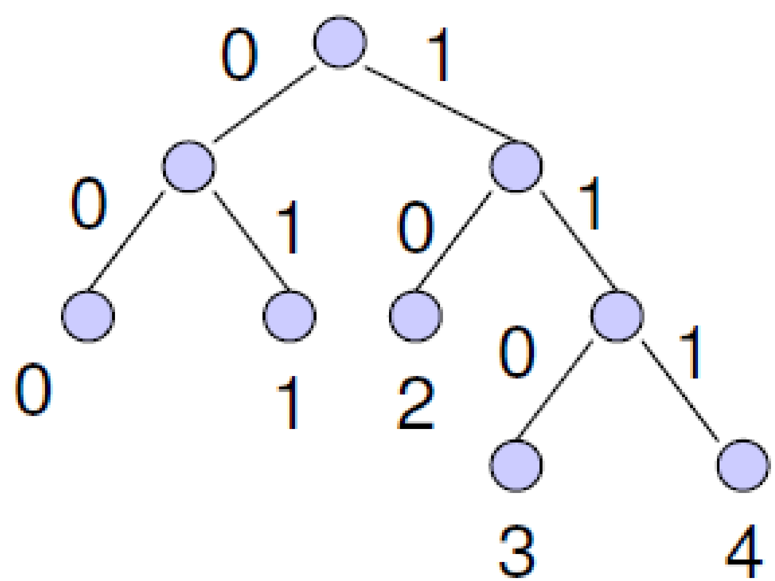
  $21 = 4\,m + 1$ i.e. $q = 4$ & $r = 1$

  1111001 (4 in unary and 1 in 2-bit binary)

Example

$$m = 5 \qquad 2^{\lceil \log_2 m \rceil} = 8 \qquad 2^{\lfloor \log_2 m \rfloor} = 4$$

- The first $8 - 5 = 3$ values of r will be represented by the 2-bit binary representation of r
- The next 5 values of r will be represented by the 3-bit binary representation of $r + 3$
- This is in fact a *prefix code*!



Code Tree for r

Example

$m = 5$ $\quad 2^{\lceil \log_2 m \rceil} = 8$ $\quad 2^{\lfloor \log_2 m \rfloor} = 4$

- The first $8 - 5 = 3$ values of r will be represented by the 2-bit binary representation of r
- The next 5 values of r will be represented by the 3-bit binary representation of $r + 3$

TABLE 3.16 Golomb code for $m = 5$.

n	q	r	Codeword	n	q	r	Codeword
0	0	0	000	8	1	3	10110
1	0	1	001	9	1	4	10111
2	0	2	010	10	2	0	11000
3	0	3	0110	11	2	1	11001
4	0	4	0111	12	2	2	11010
5	1	0	1000	13	2	3	110110
6	1	1	1001	14	2	4	110111
7	1	2	1010	15	3	0	111000

Run Length Coding

- So where do we get these *positive integers* to code?
- When the data we want to encode has lots of runs of 0's (or 1's), for example
  - fax
  - graphics
- Just send the *length* of each *run*
- Example: Assume we have long runs of 0's separated by a 1
  - Data: 00000010000000001000000000010001001
  - Represent as: 6 9 10 3 2
  - Encode these integers using Golomb codes

Example

- Data: 00000010000000001000000000010001001
- Represent as: 6 9 10 3 2
- Code: 1001 10111 11000 0110 010
- Compression ratio:

  = 35 / 21

Optimality

- It can be shown that the Golomb Codes are optimal when n follows the model:

$$P(n) = p^n (1 - p)$$

- The optimal m in that case is:

$$m = \left\lceil \frac{-1}{\log_2 p} \right\rceil$$

- This models numbers that are generated from runs of 0's followed by a 1, where $P(0) = p$ and $P(1) = 1 - p$
- For example to get $n = 5$ i.e. the run 000001, we have five 0's and one 1 and hence the probability is $p^5 (1 - p)$

Golomb Codes

- Useful for binary compression when one symbol is much more likely than another
  - binary images
  - fax documents
- Need to set the parameter m
  - Model the data
  - From training

Huffman Limitations

- Does not work well with *small* alphabets or *skewed* distributions.
- $S = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.95$, $P(a_2) = 0.02$, $P(a_3) = 0.03$

TABLE 4.1 **Huffman code for three-letter alphabet.**

Letter	Codeword
a_1	0
a_2	11
a_3	10

- $H = 0.335$ bits/symbol
- $Huffman = 1.05$ bits/symbol
- $redundancy = 1.05 – 0.335 = 0.715$ bits/symbol = 213% !!

Huffman Limitations

- Does not work well with *small* alphabets or *skewed* distributions.
- $S = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.95$, $P(a_2) = 0.02$, $P(a_3) = 0.03$
- Extend alphabet by grouping two symbols together

TABLE 4.2 **Huffman code for extended alphabet.**

Letter	Probability	Code
a_1a_1	0.9025	0
a_1a_2	0.0190	111
a_1a_3	0.0285	100
a_2a_1	0.0190	1101
a_2a_2	0.0004	110011
a_2a_3	0.0006	110001
a_3a_1	0.0285	101
a_3a_2	0.0006	110010
a_3a_3	0.0009	110000

- $H = 0.335$ bits/symbol
- *Extended Huffman* = 1.222 bits/2 symbols = 0.611 bits/sym
- *redundancy* = 0.611 – 0.335 = 0.276 bits/symbol = 72% !!

Huffman Limitations

- Does not work well with *small* alphabets or *skewed* distributions.
- $S = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.95$, $P(a_2) = 0.02$, $P(a_3) = 0.03$
- We can keep *extending* the alphabet, but the size grows *exponentially* with the block size:
  - 2 symbols per block $\rightarrow 3^2$ extended alphabet
  - 3 symbols per block $\rightarrow 3^3$ extended alphabet
  - …
- And we need to have codewords for every possible combination of symbols: large storage for the code tree and the code table!
- One solution: Arithmetic Coding!

Arithmetic Coding

- Creates codewords for groups of symbols or *sequences*
- Assigns a unique identifier or *tag* for every sequence of symbols
- This tag is then converted to a unique binary code or *codeword*
- A unique codeword can be assigned to a sequence of length m without having to generate codewords for *all* sequences of length m, unlike Huffman Coding
- What is this tag?

Binary Tags

- Arithmetic coding assigns a unique interval $[L, R)$ in the unit interval $[0, 1)$ for each sequence of symbols
  - For example, a sequence *abaa* can be assigned the interval [0.23, 0.35)
- Since each sequence has its own interval, the *tag* can be chosen as any *fraction* in that interval
  - For example, the tag for this sequence can be 0.23 or the midpoint 0.29
- The binary *codeword* for that sequence will be generated from the binary representation of that fraction i.e. tag

Binary Fractions

- Decimal fractions $x = 0.123$ means that:

  $x = 1 \text{ x } 10^{-1} + 2 \text{ x } 10^{-2} + 3 \text{ x } 10^{-3} = 0.1 + 0.02 + 0.003$

- The same applies for binary fractions e.g. $x = 0.101_b$ means that:

  $x = 1 \text{ x } 2^{-1} + 0 \text{ x } 2^{-2} + 1 \text{ x } 2^{-3} = 0.5 + 0.125 = 0.625$

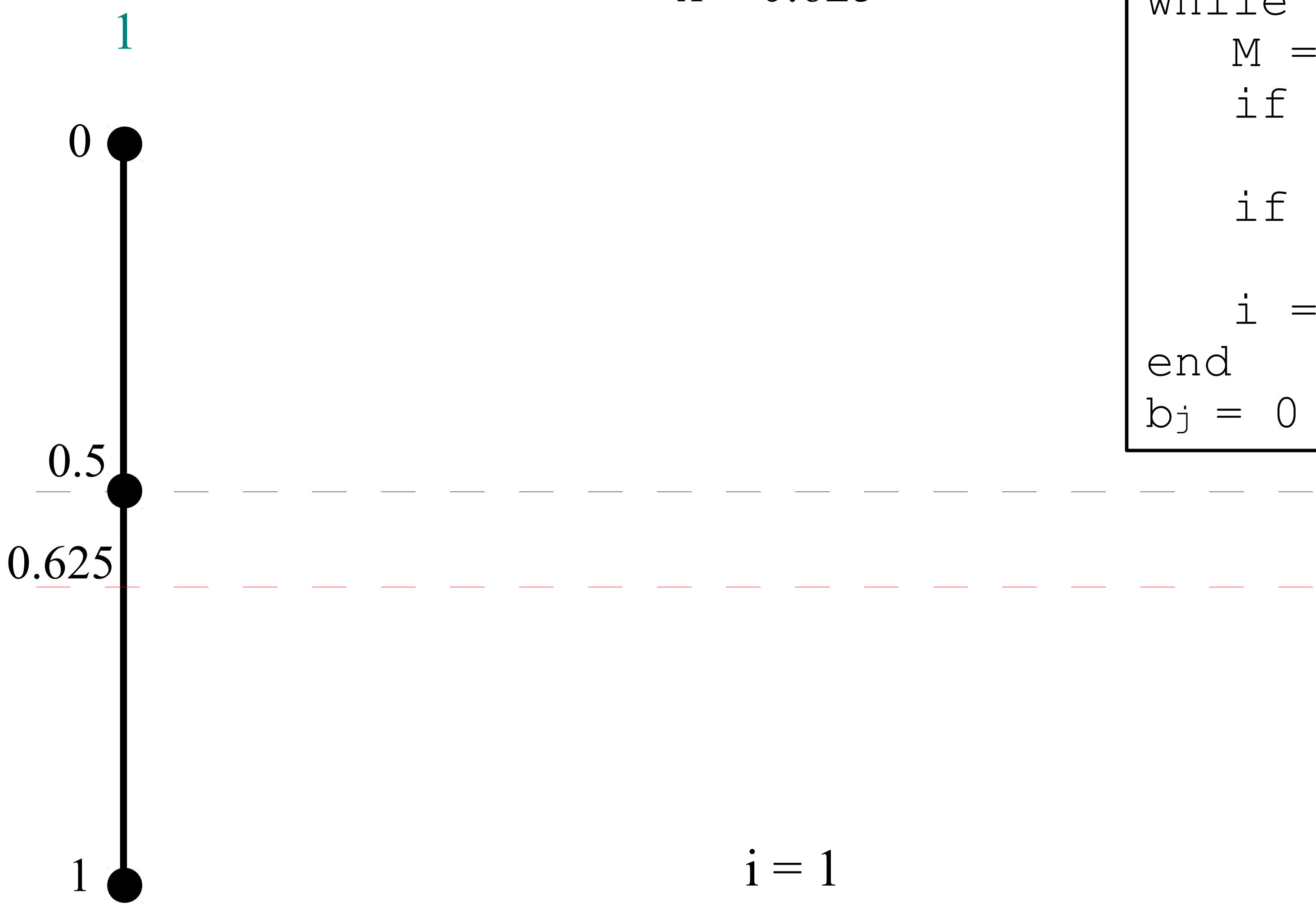
- Any real number in the interval [0, 1) can be represented by a *binary* fraction

Decimal to Binary Conversion

```
L = 0; R = 1; i = 1;
while x > L
   M = (L + R) / 2;
   if x < M then
      bi = 0; R = M;
   if x >= M then
      bi = 1; L = M;
   i = i + 1
end
bj = 0 for all j > i
```

Example

x = 0.625



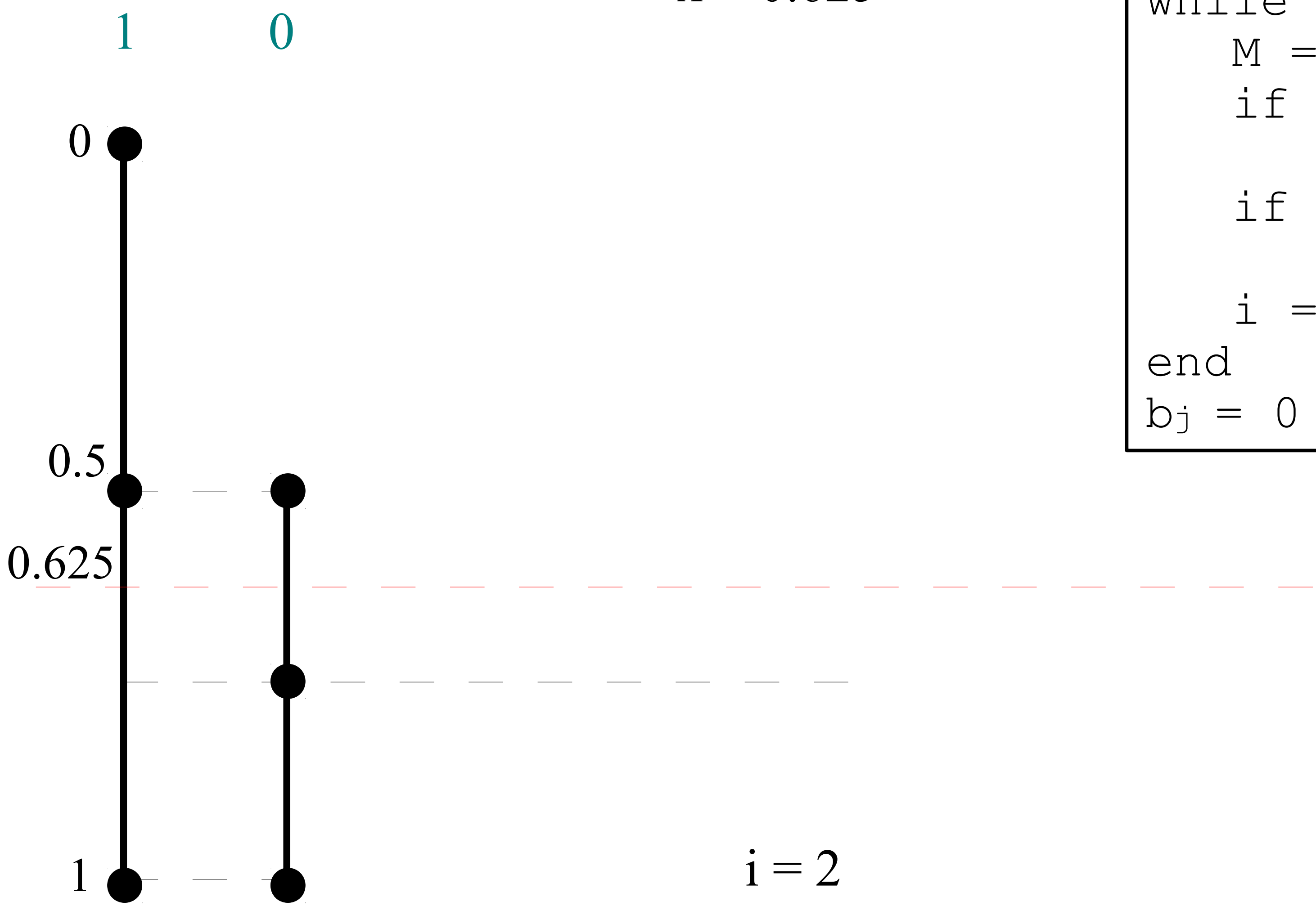

```
L = 0; R = 1; i = 1;
while x > L
   M = (L + R) / 2;
   if x < M then
      bi = 0; R = M;
   if x >= M then
      bi = 1; L = M;
   i = i + 1
end
bj = 0 for all j > i
```

i = 1

L = 0   R = 1   M = 0.5 $b_i = 1$

Example

$x = 0.625$



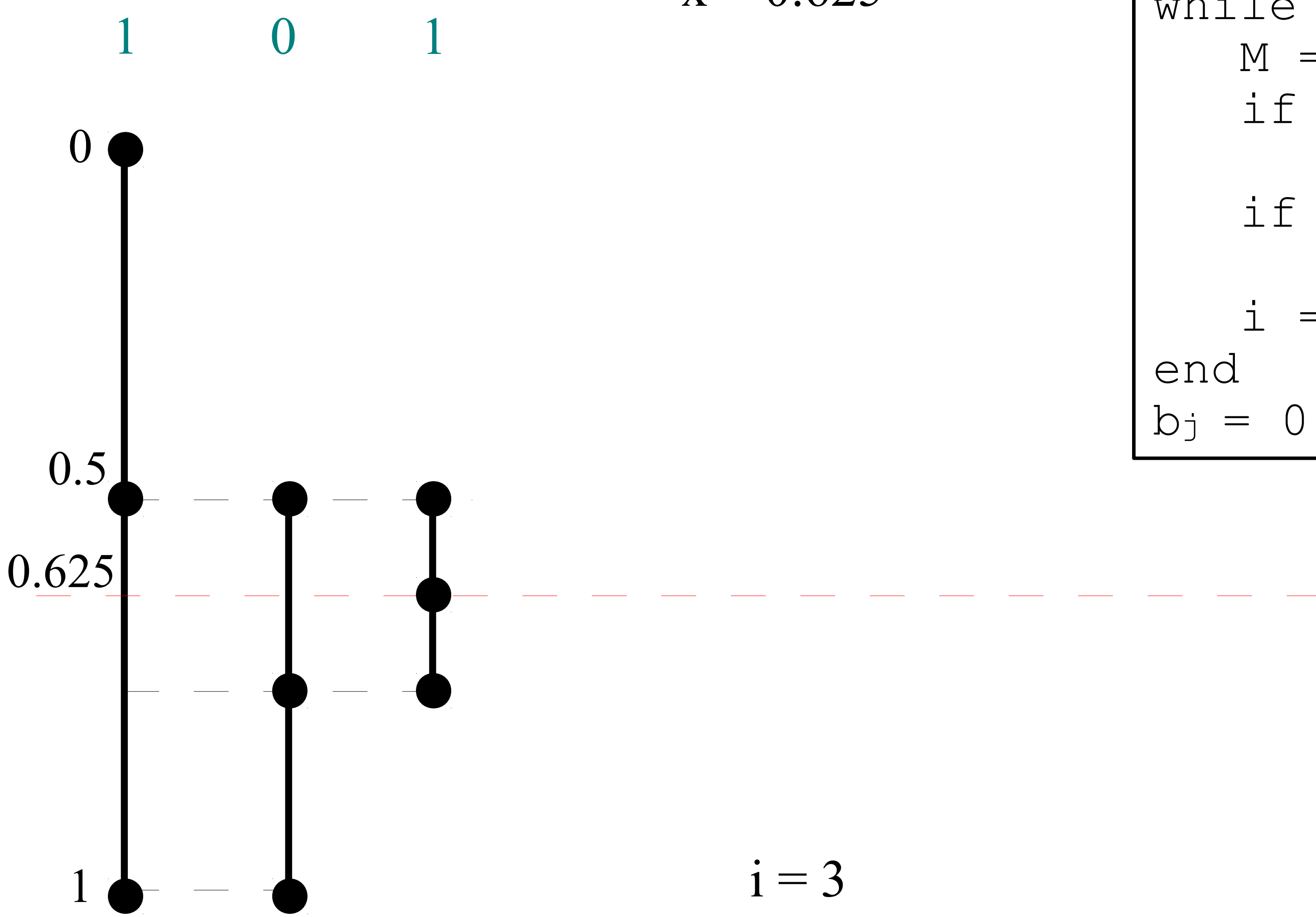

```
L = 0; R = 1; i = 1;
while x > L
    M = (L + R) / 2;
    if x < M then
        bi = 0; R = M;
    if x >= M then
        bi = 1; L = M;
    i = i + 1
end
bj = 0 for all j > i
```

$i = 2$

$L = 0.5$ $R = 1$ $M = 0.75$ $b_i = 0$

Example

$x = 0.625$



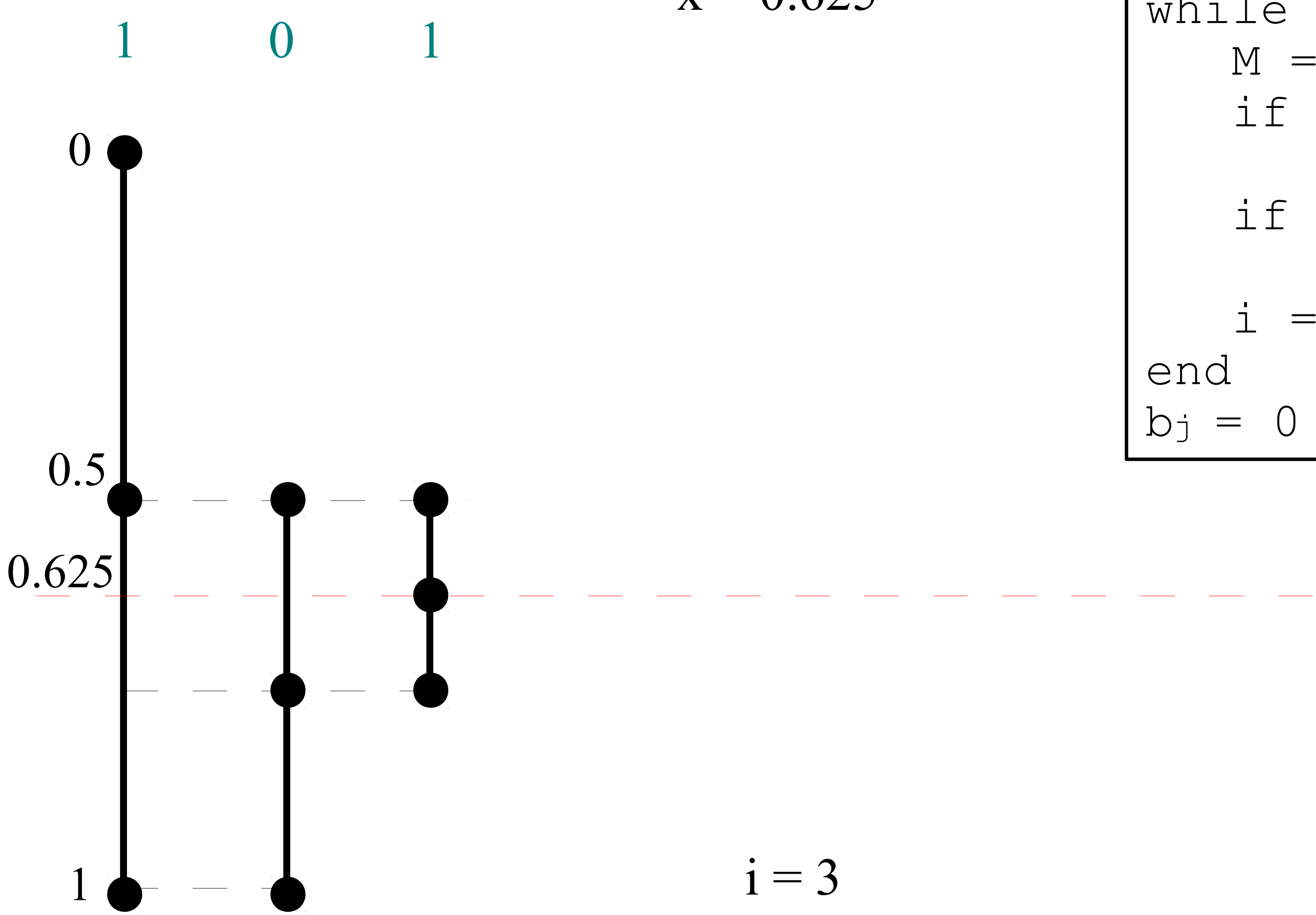

```
L = 0; R = 1; i = 1;
while x > L
    M = (L + R) / 2;
    if x < M then
        bi = 0; R = M;
    if x >= M then
        bi = 1; L = M;
    i = i + 1
end
bj = 0 for all j > i
```

$i = 3$

$L = 0.5$ $R = 0.75$ $M = 0.625$ $b_i = 1$

Example

$x = 0.625$




```
L = 0; R = 1; i = 1;
while x > L
    M = (L + R) / 2;
    if x < M then
        bi = 0; R = M;
    if x >= M then
        bi = 1; L = M;
    i = i + 1
end
bj = 0 for all j > i
```

$i = 3$

$L = 0.625$ $R = 0.75$ Loop done...

Conversion with Scaling

- What's the problem with this algorithm?
- Fractions get smaller and smaller, and eventually will approach the precision of the machine
- Solution?
- Scaling: scale the interval to the unit interval after each iteration...

```
L = 0; R = 1; i = 1;
while x > L
   M = (L + R) / 2;
   if x < M then
      bi = 0; R = M;
   if x >= M then
      bi = 1; L = M;
   i = i + 1
end
bj = 0 for all j > i
```

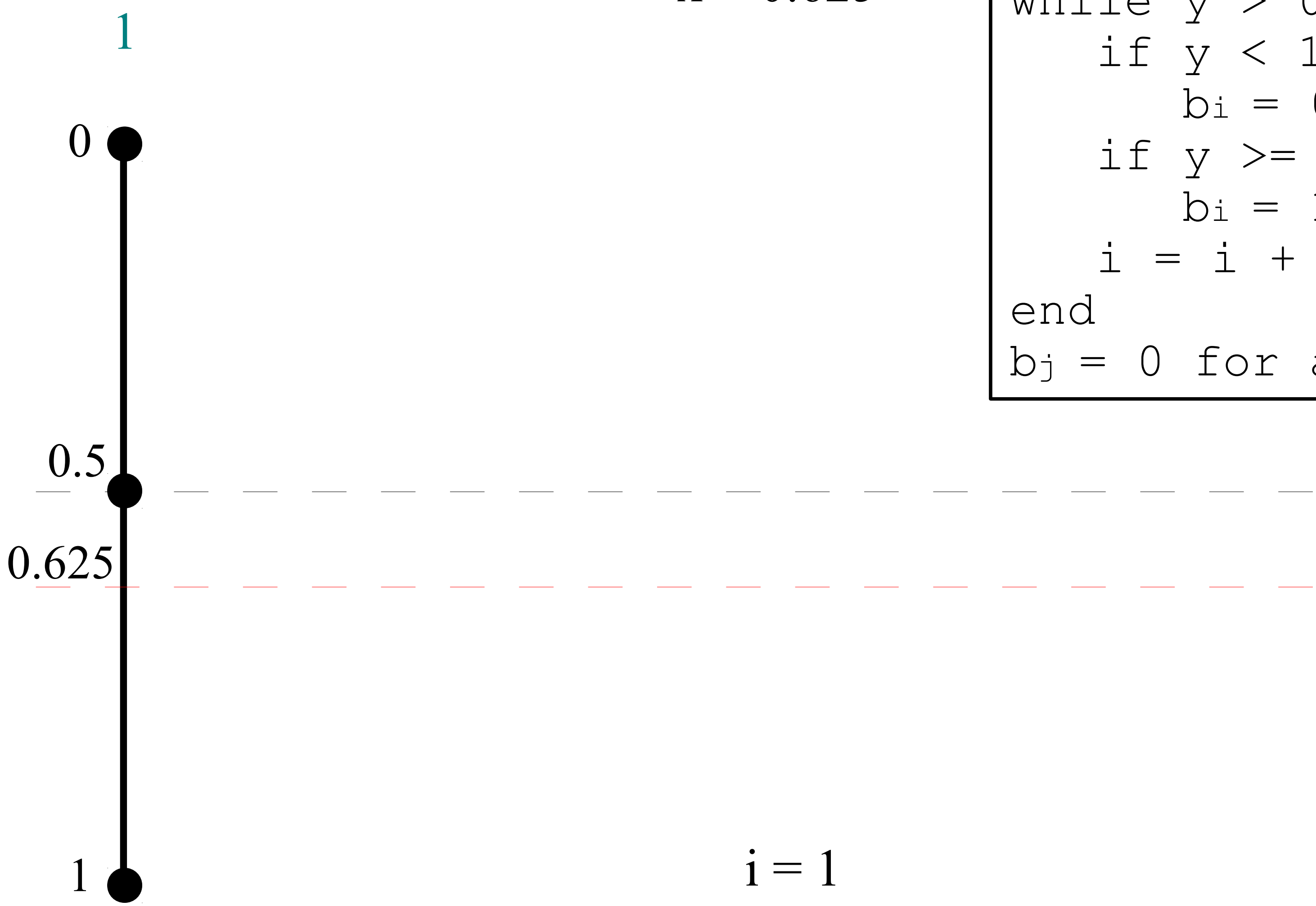
Conversion with Scaling

```
y = x; i = 1;
while y > 0
   if y < 1/2 then
      bi = 0; y = 2y;
   if y >= 1/2 then
      bi = 1; y = 2y - 1;
   i = i + 1
end
bj = 0 for all j > i
```

Example

$x = 0.625$

```
y = x; i = 1;
while y > 0
   if y < 1/2 then
      bi = 0; y = 2y;
   if y >= 1/2 then
      bi = 1; y = 2y - 1;
   i = i + 1
end
bj = 0 for all j > i
```



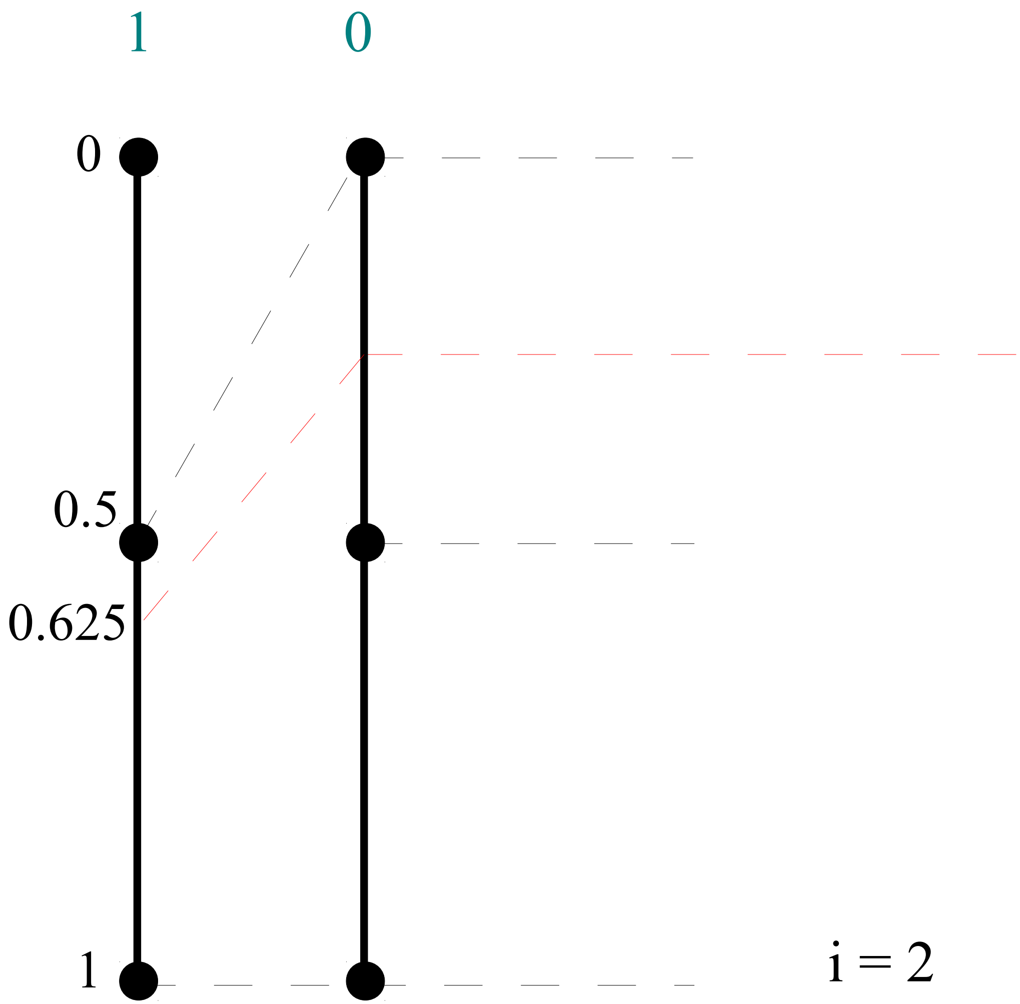
$i = 1$

$y = 0.625$

$b_i = 1$

Example

$x = 0.625$



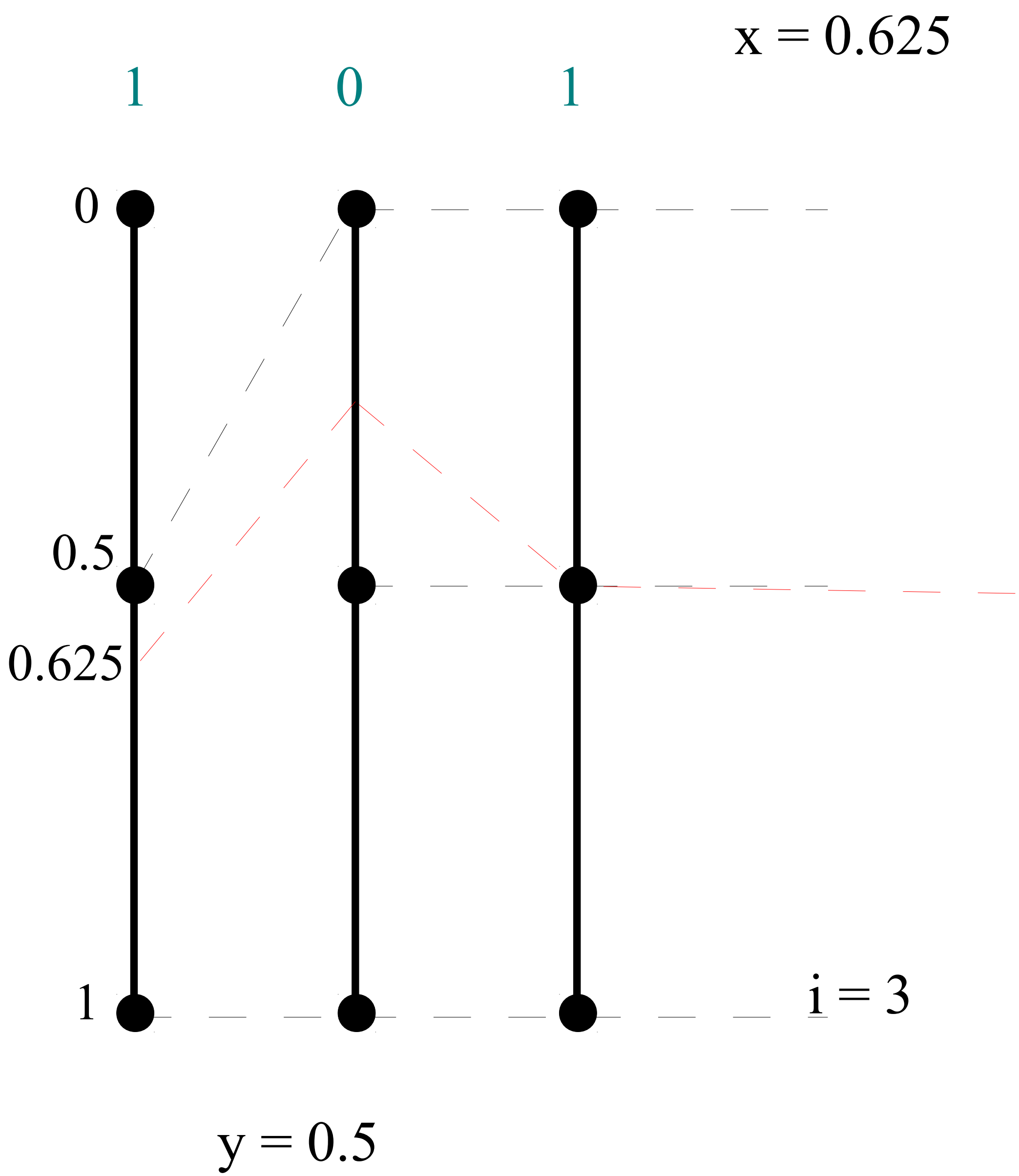

```
y = x; i = 1;
while y > 0
    if y < 1/2 then
        bi = 0; y = 2y;
    if y >= 1/2 then
        bi = 1; y = 2y - 1;
    i = i + 1
end
bj = 0 for all j > i
```

$i = 2$

$y = 0.25$

$b_i = 0$

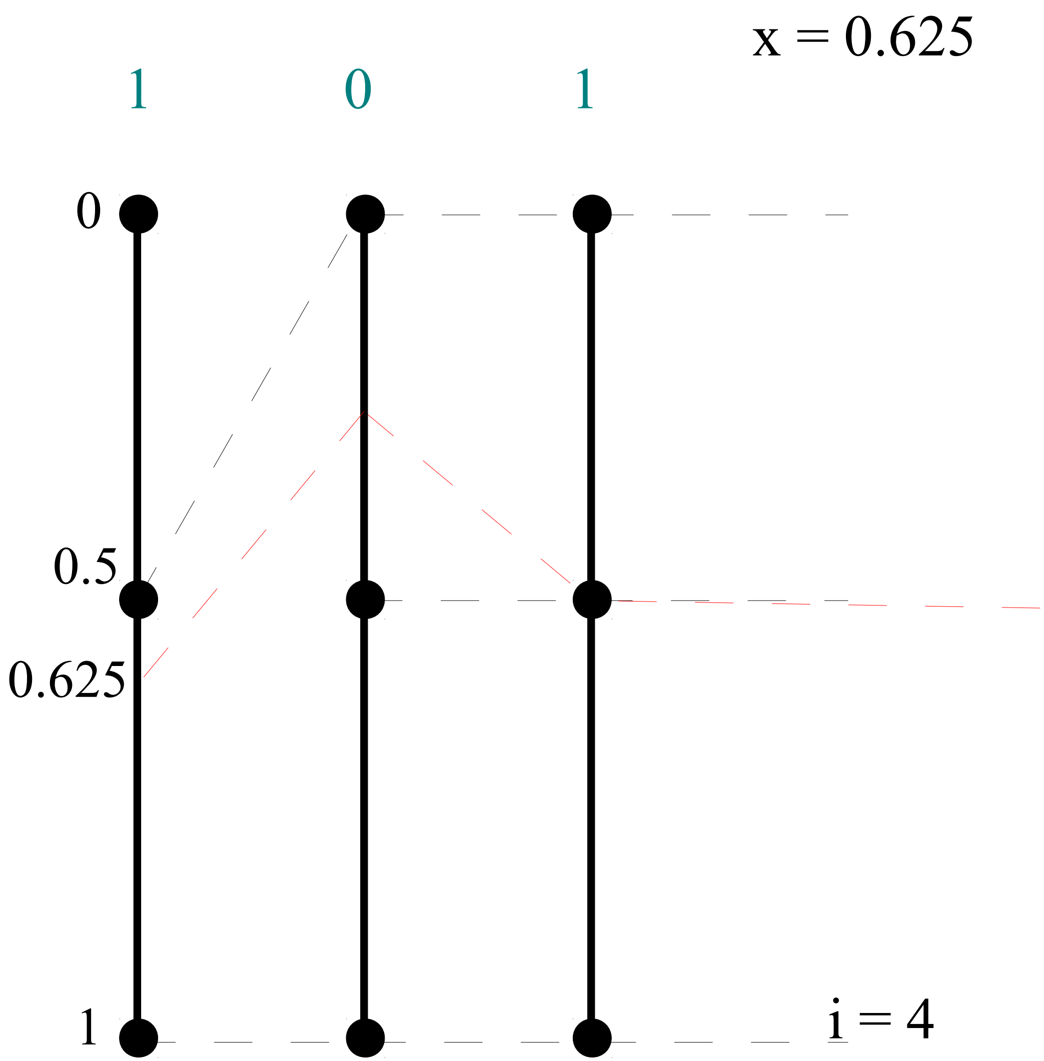
Example




```
y = x; i = 1;
while y > 0
    if y < 1/2 then
        bi = 0; y = 2y;
    if y >= 1/2 then
        bi = 1; y = 2y - 1;
    i = i + 1
end
bj = 0 for all j > i
```

$b_i = 1$

Example



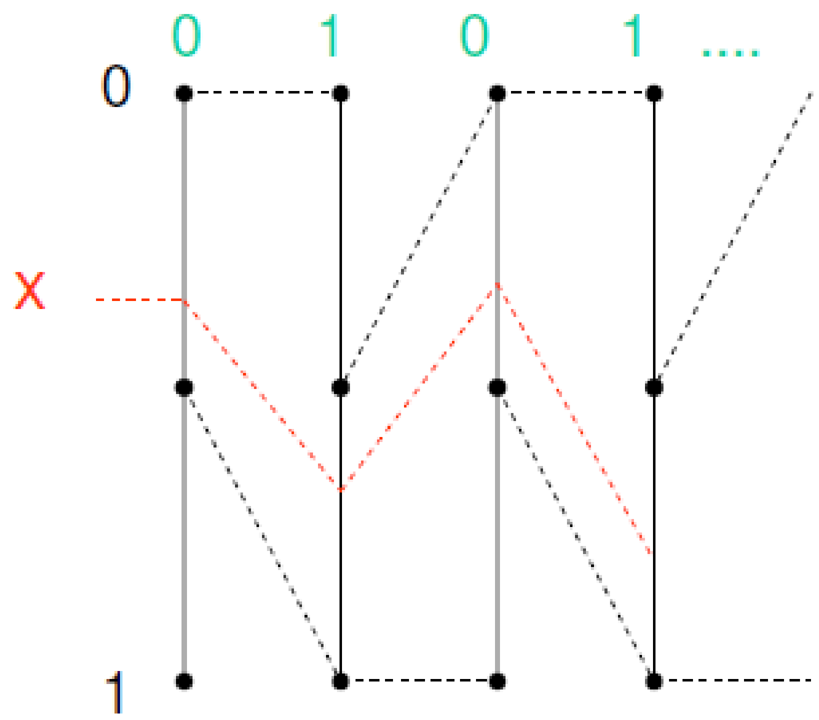

```
y = x; i = 1;
while y > 0
    if y < 1/2 then
        bi = 0; y = 2y;
    if y >= 1/2 then
        bi = 1; y = 2y - 1;
    i = i + 1
end
bj = 0 for all j > i
```

y = 0

Loop ends...

Another Example

x = 0.352...



Binary Tags

- Arithmetic coding assigns a unique interval $[L, R)$ in the unit interval $[0, 1)$ for each sequence of symbols
- Since each sequence has its own interval, the *tag* can be chosen as any *fraction* in that interval
- The binary *codeword* for that sequence will be generated from the binary representation of that fraction (tag) by keeping the first k significant bits
  - If the tag is $0.b_1b_2b_3\ldots b_kb_{k+1}\ldots$, the binary codeword will be $b_1b_2b_3\ldots b_k$ which belongs to the interval
- It turns out

$$k = \left\lceil \log_2 \frac{1}{R-L} \right\rceil + 1$$

Arithmetic Coding

- How do we find the tags/intervals for the sequences?
- We will use the *probabilities* of the symbols from the alphabet to restrict the interval
- Recall that the probabilities sum to 1, so all the probabilities fit in the unit interval [0, 1)
- As more and more symbols come in, the interval becomes smaller and smaller
- Once done with the sequence, we choose the *tag* as any fraction from the interval

Example

$S = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, $P(a_3) = 0.2$

When we get a_1, we restrict ourselves to the interval corresponding to a_1 i.e. [0.0, 0.7)

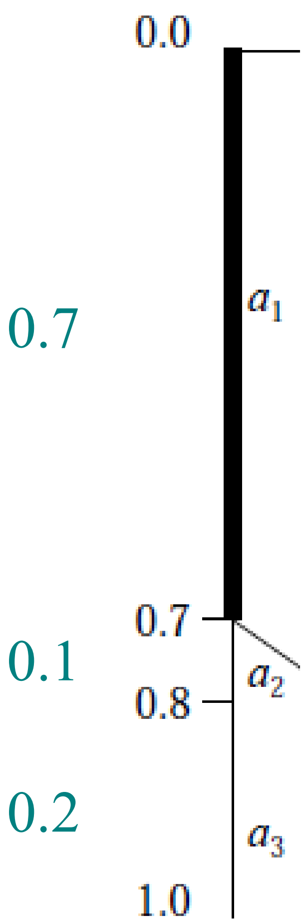


FIGURE 4.1 Restricting the interval containing the tag for the input sequence $\{a_1, a_2, a_3, \ldots\}$.

Example

$S = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, $P(a_3) = 0.2$

We then divide this interval into sections proportional to the original probabilities e.g. a_1 will now correspond to the interval [0, 0 + 0.7x0.7) = [0.0, 0.49) and a_2 corresponds to the interval [0.7 x 0.7, 0.8 x 0.7) = [0.49, 0.56)

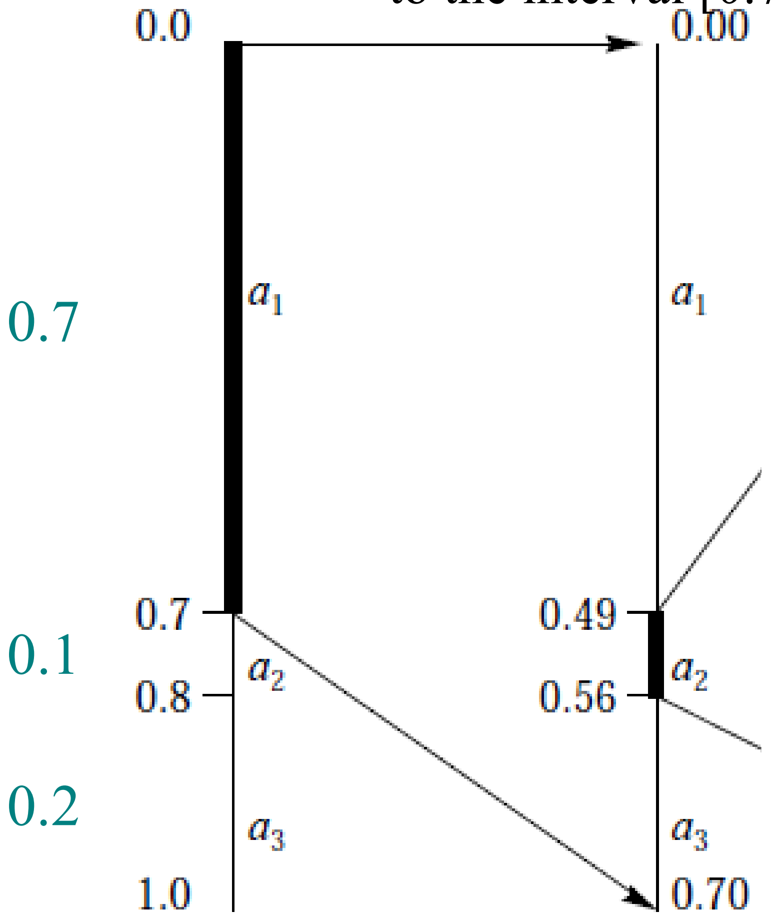


FIGURE 4.1 **Restricting the interval containing the tag for the input sequence $\{a_1, a_2, a_3, \ldots\}$.**

Example

$S = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, $P(a_3) = 0.2$

When we get a_2, we restrict ourselves to the interval corresponding to a_2 i.e. $[0.49, 0.56)$

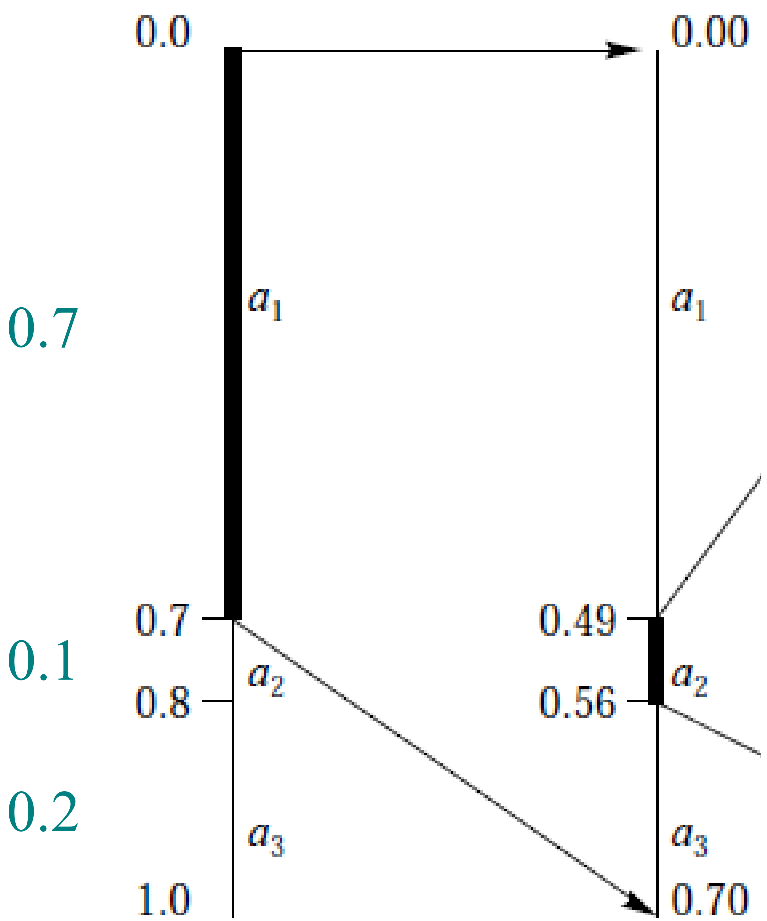


FIGURE 4.1 **Restricting the interval containing the tag for the input sequence $\{a_1, a_2, a_3, \ldots\}$.**

Example

$S = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, $P(a_3) = 0.2$

We then divide this interval into sections proportional to the original probabilities e.g. a_1 will now correspond to the interval [0.49, 0.49 + 0.7x0.07) = [0.49, 0.539)

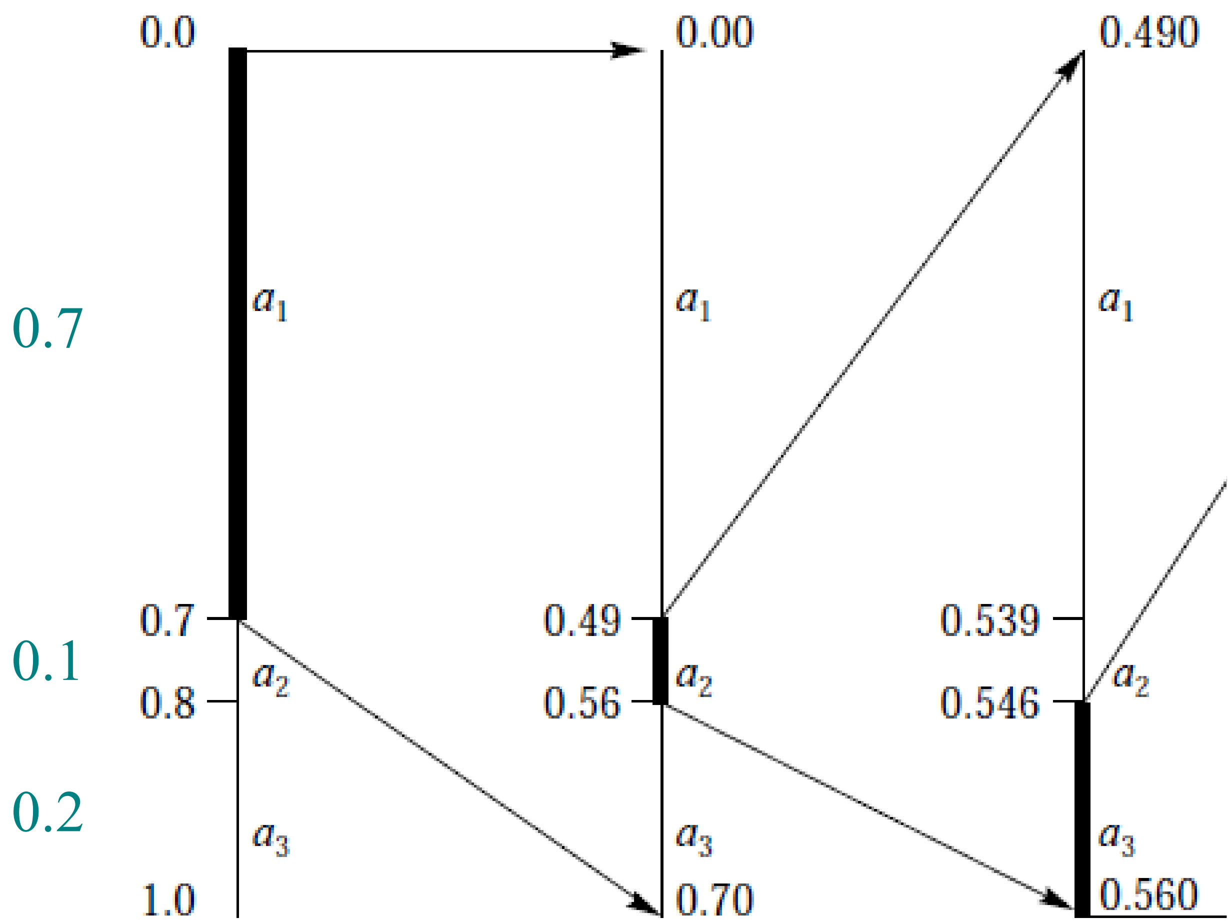


FIGURE 4.1 Restricting the interval containing the tag for the input sequence $\{a_1, a_2, a_3, \ldots\}$.

Example

$S = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, $P(a_3) = 0.2$

When we get a_3, we restrict ourselves to the interval corresponding to a_3 i.e. [0.546, 0.560)

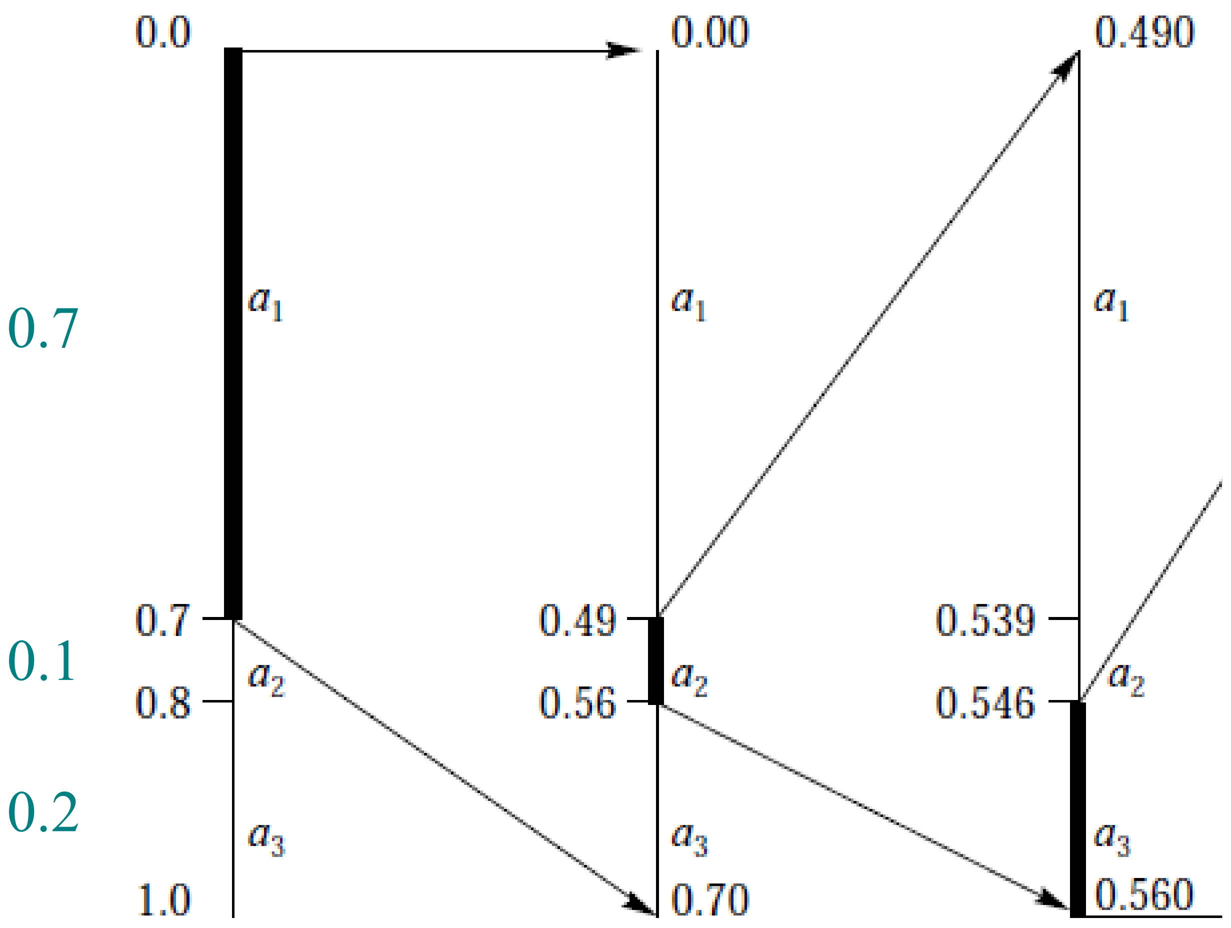


FIGURE 4. 1 **Restricting the interval containing the tag for the input sequence $\{a_1, a_2, a_3, \ldots\}$.**

Example

$S = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, $P(a_3) = 0.2$

Once we done with the sequence, we choose any fraction in that interval as the *tag*

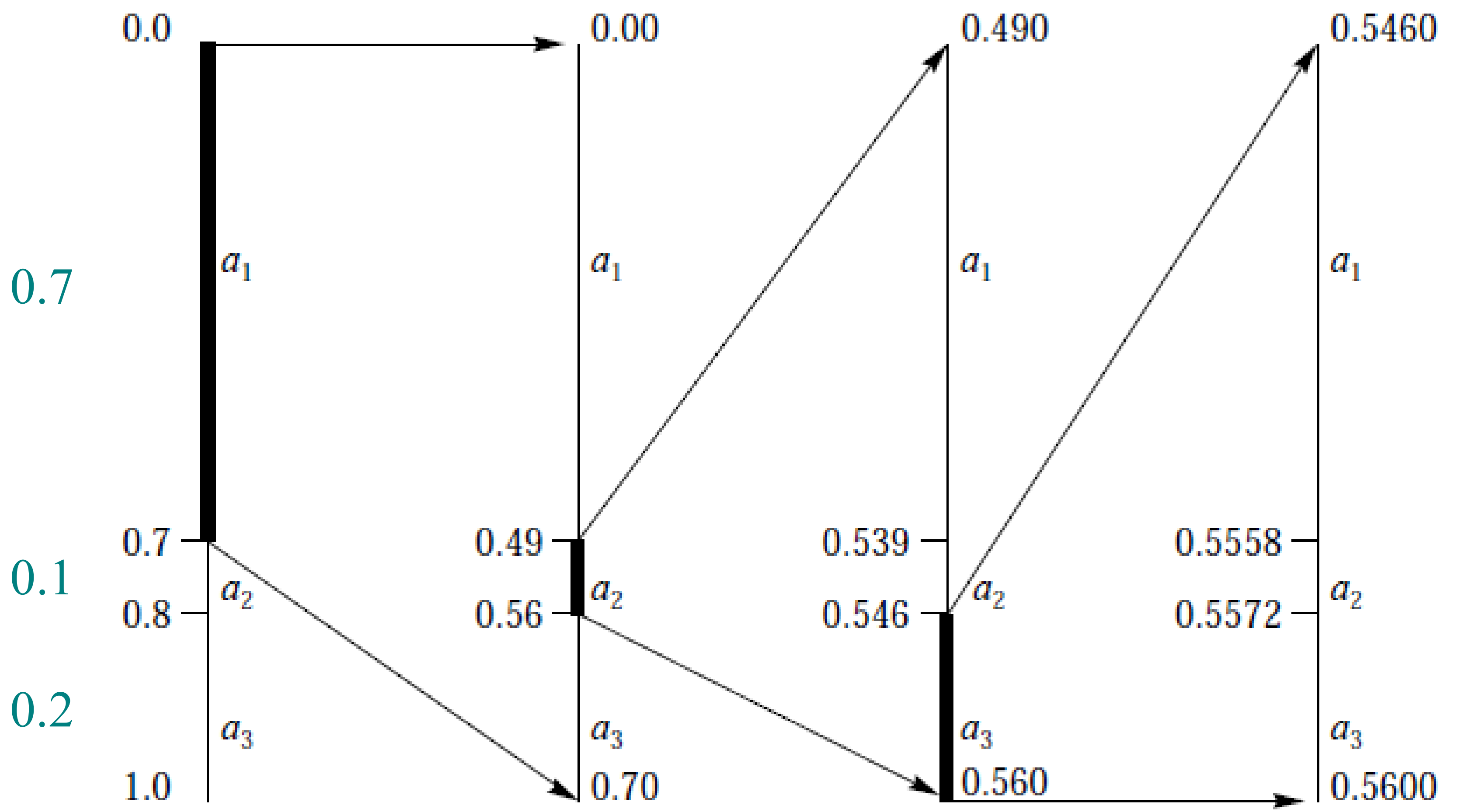


FIGURE 4. 1 **Restricting the interval containing the tag for the input sequence $\{a_1, a_2, a_3, \ldots\}$.**

Encoding Algorithm

Input

- Probabilities for symbols $P(a_i)$ for every i
- Cumulative probability for symbols $C(a_i) = \sum_{j=1}^{i} P(a_j)$
- Message to be encoded: $x_1 x_2 \cdots x_n$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
   W = R - L;
   L = L + W * C(x_{i-1});
   R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```

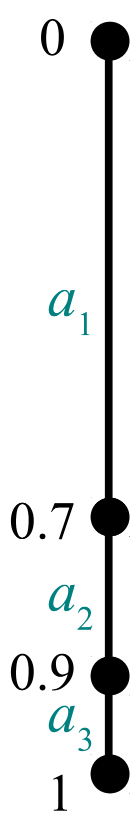
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



W = 1
L = 0
R = 1

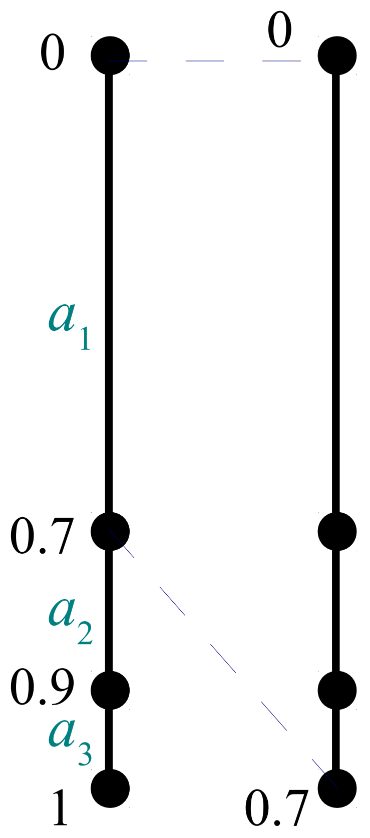
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



i = 1

W = 1
L = 0
R = 0.7

$C(x_{i-1}) = 0$
$P(x_i) = 0.7$

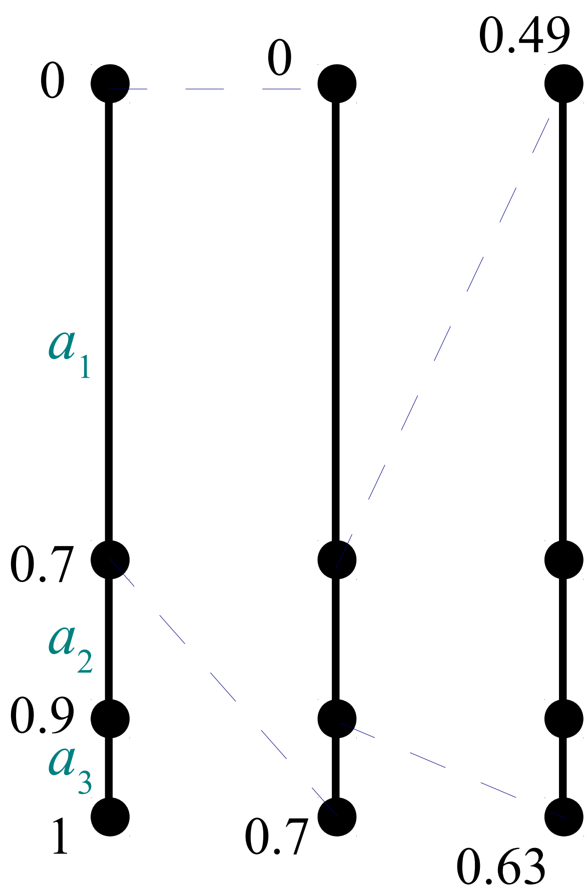
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



$i = 2$

$W = 0.7$
$L = 0.49$
$R = 0.63$

$C(x_{i-1}) = 0.7$
$P(x_i) = 0.2$

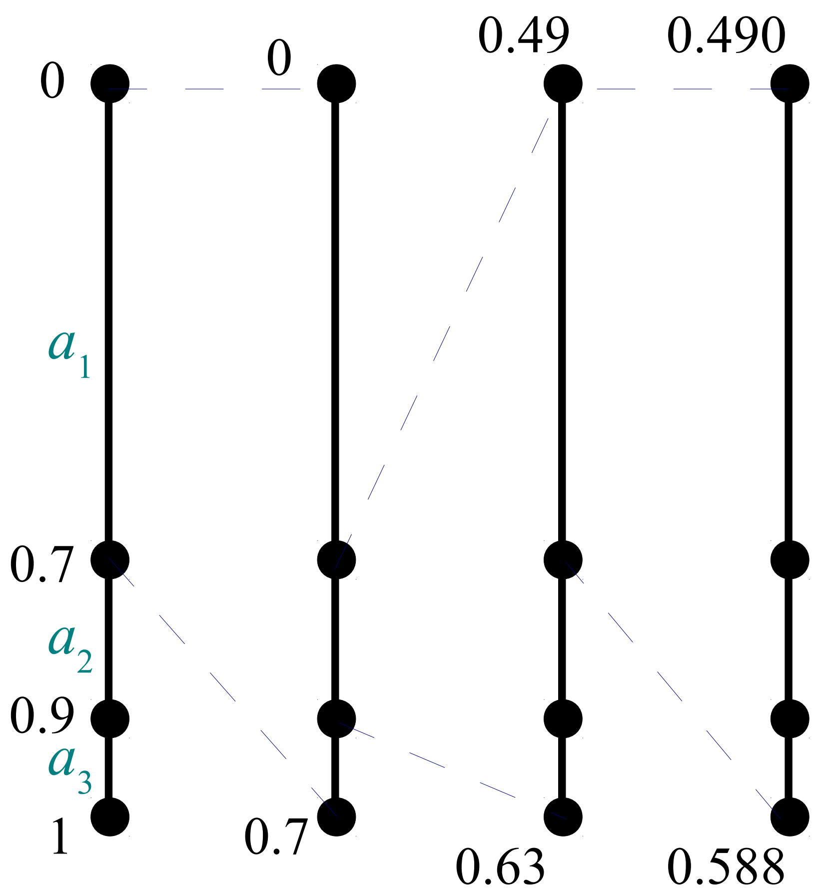
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



i = 3

W = 0.14
L = 0.49
R = 0.588

$C(x_{i-1}) = 0$
$P(x_i) = 0.7$

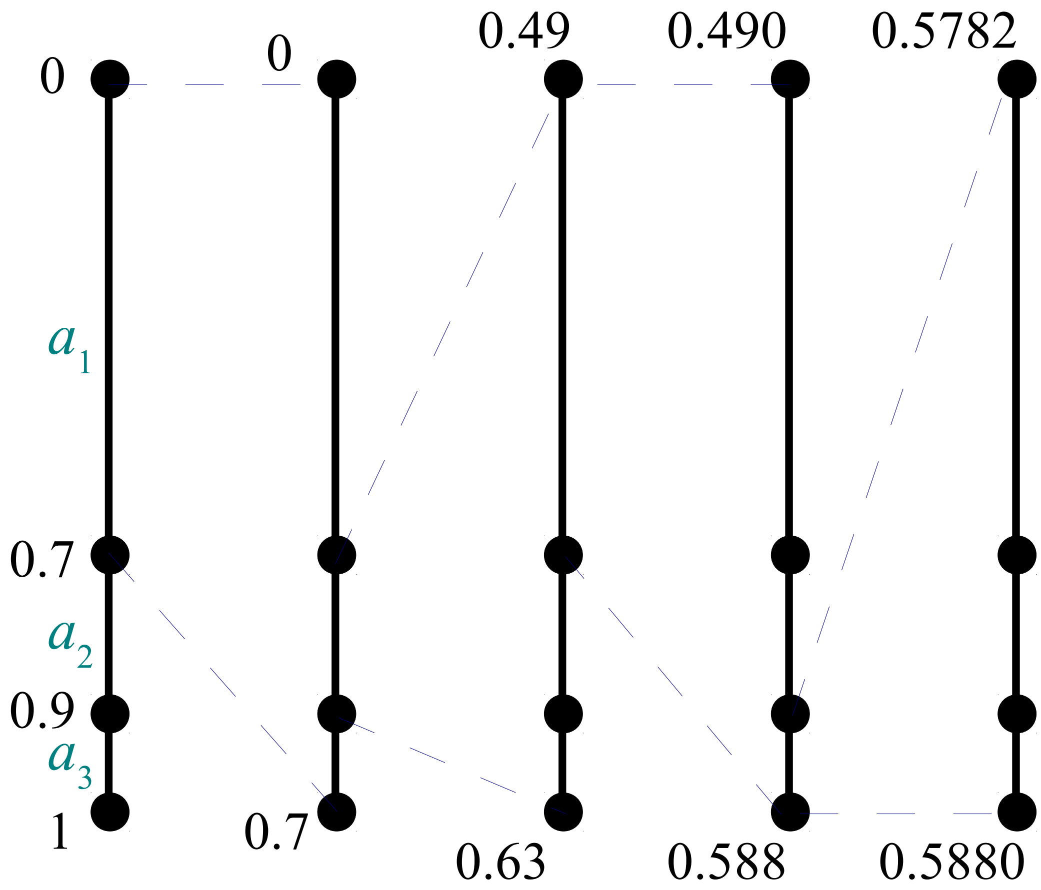
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



i = 4

W = 0.098
L = 0.5782
R = 0.5880

$C(x_{i-1}) = 0.9$
$P(x_i) = 0.1$

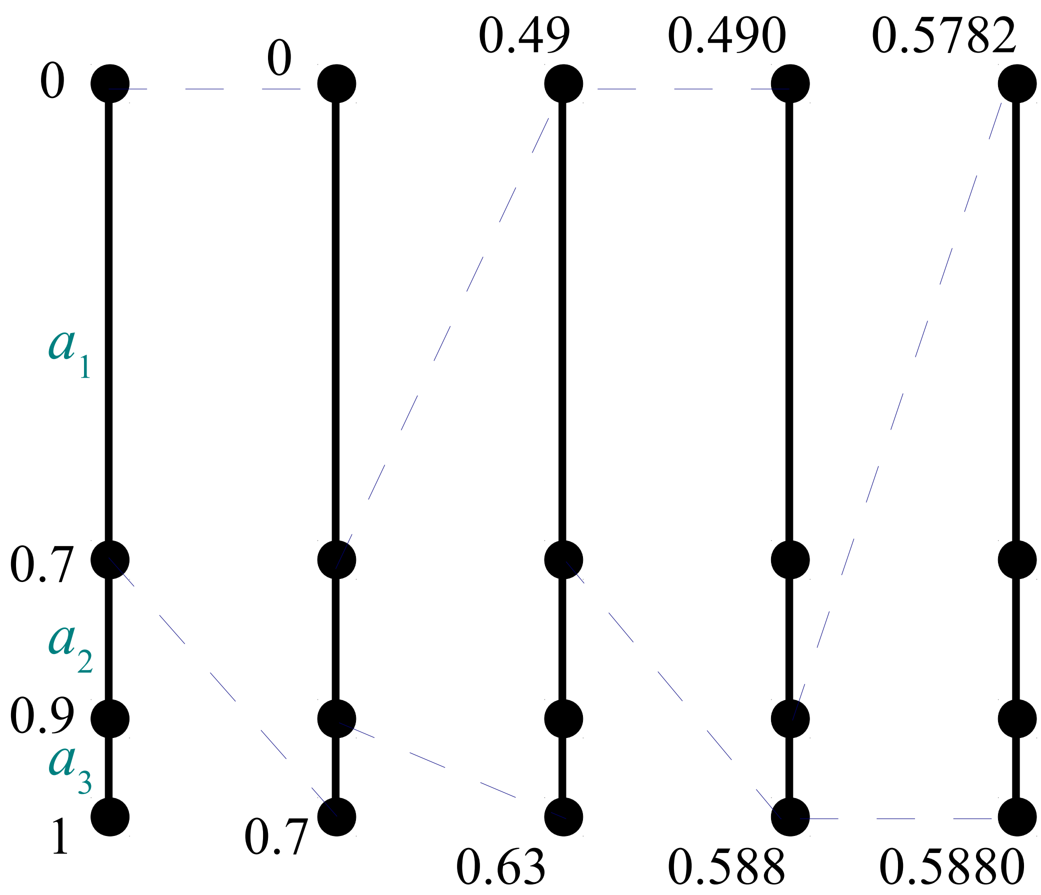
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



$L = 0.5782$
$R = 0.5880$

Choose the tag: $t = \frac{L+R}{2}$

$t = 0.5831$

Binary Codeword

- The codeword is the first k most-significant bits (MSB) of the *tag*
- How many bits k to use?
- To guarantee that binary code is unique i.e. the binary fraction lies within the interval $[L, R)$:

$$k = \left\lceil \log_2 \frac{1}{R-L} \right\rceil + 1$$

- This resulting code is also a *prefix code*, and it can be shown to have a *rate* for a message of m symbols such that:

$$H \leqslant r_A \leqslant H + \frac{2}{m}$$

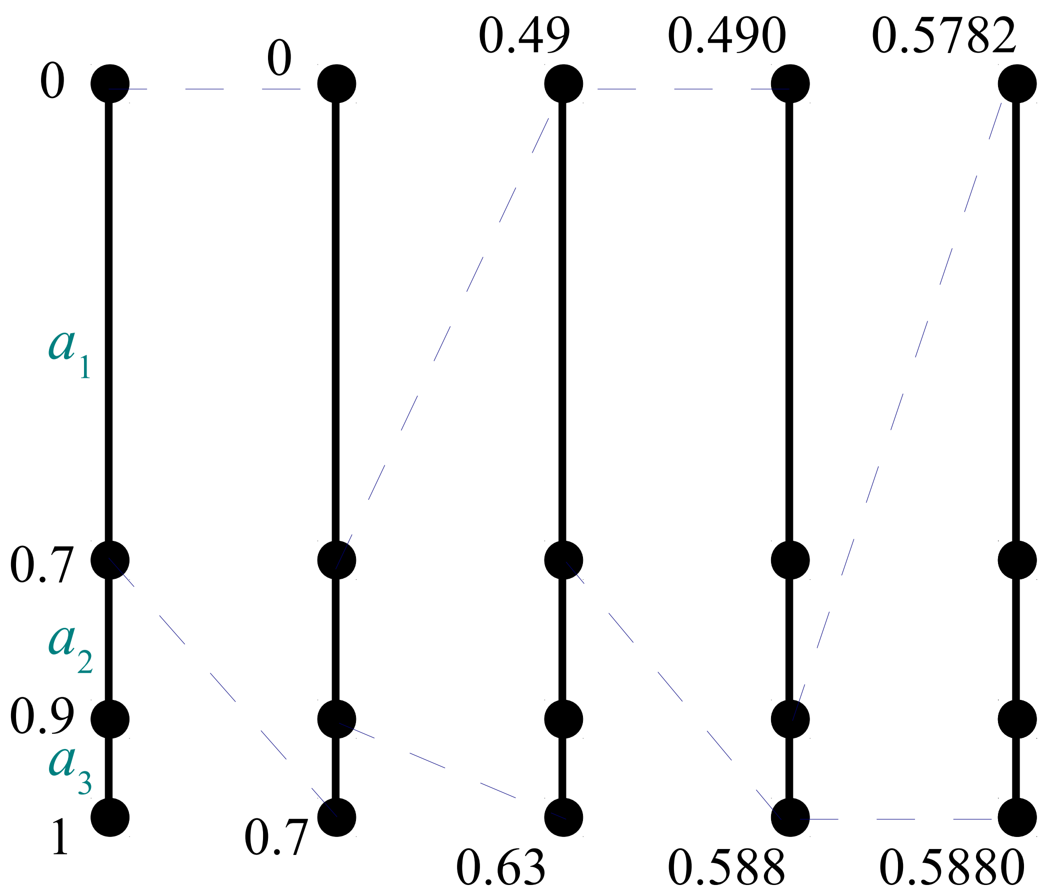
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



L = 0.5782
R = 0.5880

Choose the tag: $t = \dfrac{L+R}{2}$

$t = 0.5831 = 0.10010101010001$

$$k = \left\lceil \log_2 \frac{1}{R-L} \right\rceil + 1 = 8$$

Codeword = 10010101

Decoding

- Once we have the *tag*, we can *decode* the message to obtain the sequence corresponding to that tag
- The *decoder* performs similar operations to the *encoder* that generated the tag

Decoding Algorithm

Input

- Probabilities for symbols $P(a_i)$ for every i
- Cumulative probability for symbols $C(a_i)=\sum_{j=1}^{i} P(a_j)$
- Codeword for message of m symbols: $b_1 b_2 \cdots b_k$

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
   W = R - L;
   find j such that:
      L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
   output aj;
   L = L + W * C(aj-1);
   R = L + W * P(aj);
```

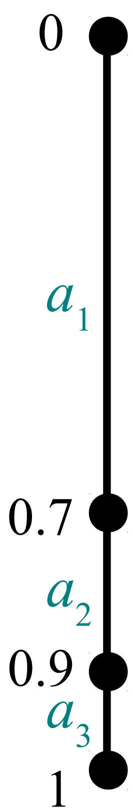
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Code: 10010101

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



$t = 0.5831$

$L = 0$
$R = 1$

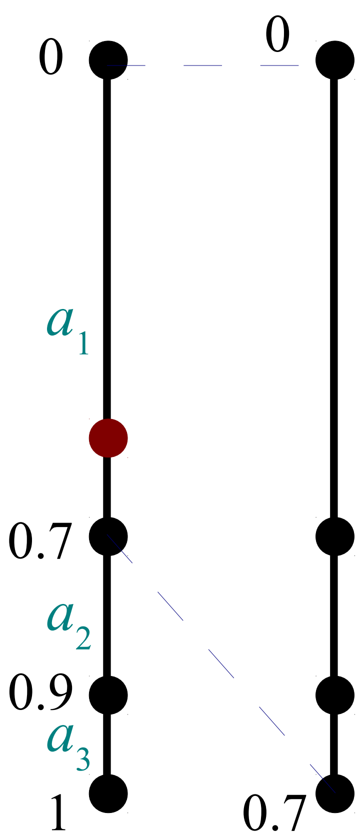
Example

$P(a_1) = 0.7,\ P(a_2) = 0.1,\ P(a_3) = 0.2$

$C(a_1) = 0.7,\ C(a_2) = 0.8,\ C(a_3) = 1$

Code: 10010101

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



$i = 1$

$t = 0.5831$

$W = 1$
$L = 0$
$R = 0.7$

$j = 1$

Emit: a_1

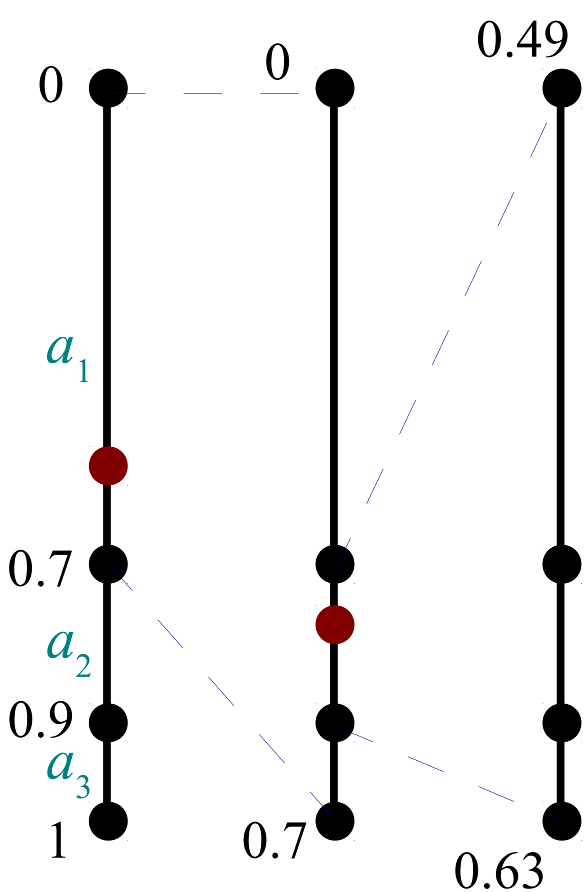
Example

$P(a_1) = 0.7,\ P(a_2) = 0.1,\ P(a_3) = 0.2$

$C(a_1) = 0.7,\ C(a_2) = 0.8,\ C(a_3) = 1$

Code: 10010101

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



$i = 2$

$t = 0.5831$

$W = 0.7$
$L = 0.49$
$R = 0.63$

$j = 2$

Emit: $a_1 a_2$

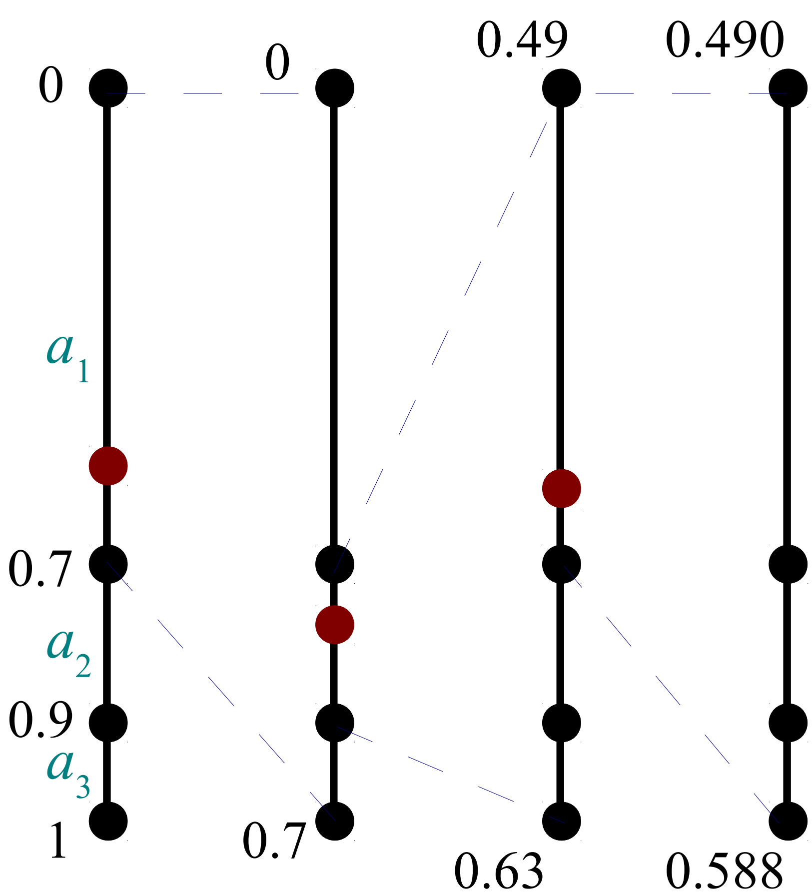
Example

$P(a_1) = 0.7$, $P(a_2) = 0.1$, $P(a_3) = 0.2$

$C(a_1) = 0.7$, $C(a_2) = 0.8$, $C(a_3) = 1$

Code: 10010101

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



i = 3

t = 0.5831

W = 0.14
L = 0.49
R = 0.588

j = 1

Emit: $a_1 a_2 a_1$

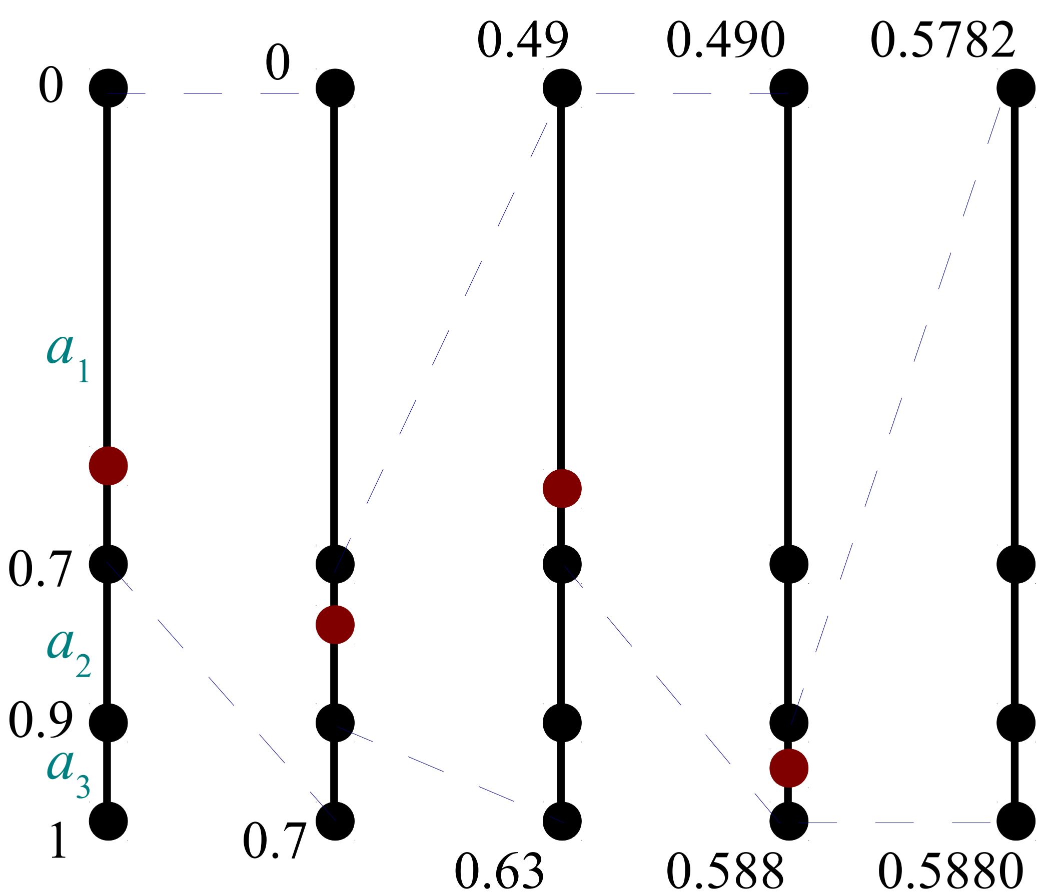
Example

$P(a_1) = 0.7$, $P(a_2) = 0.1$, $P(a_3) = 0.2$

$C(a_1) = 0.7$, $C(a_2) = 0.8$, $C(a_3) = 1$

Code: 10010101

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



i = 4

t = 0.5831

W = 0.098

L = 0.5782

R = 0.5880

j = 3

Emit: $a_1 a_2 a_1 a_3$

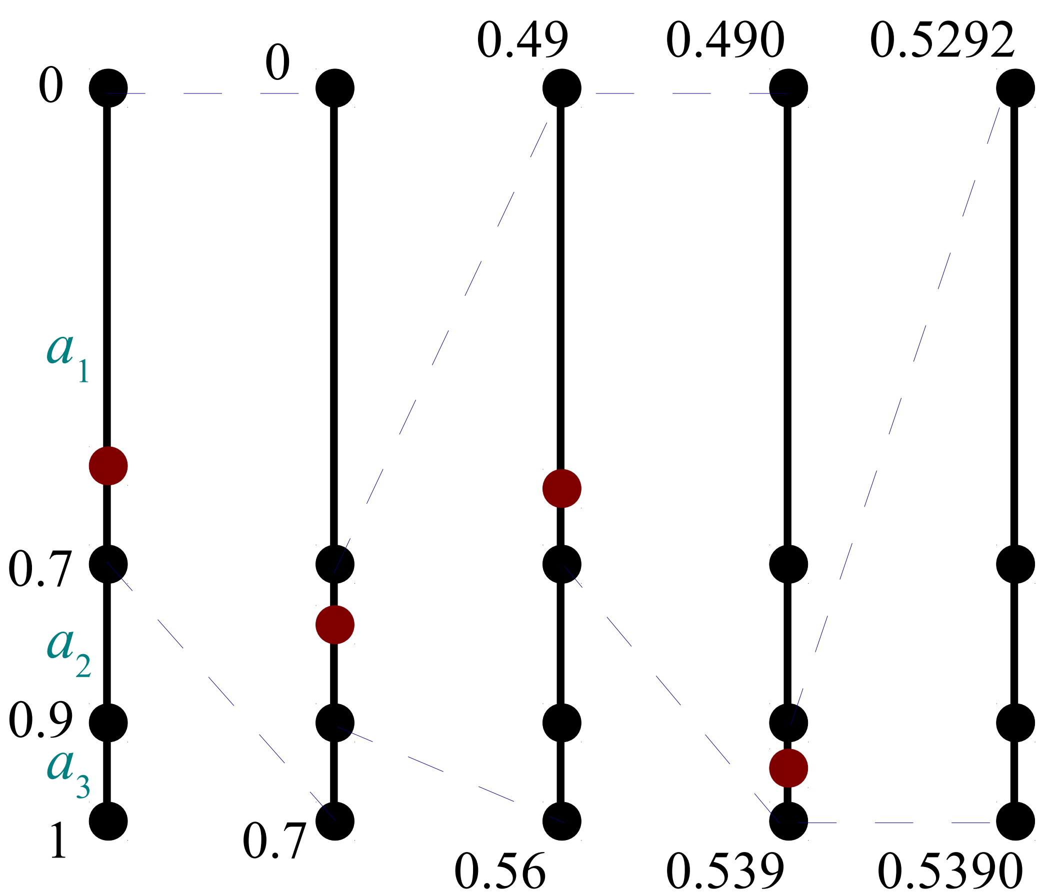
Example

$P(a_1) = 0.7,\ P(a_2) = 0.1,\ P(a_3) = 0.2$

$C(a_1) = 0.7,\ C(a_2) = 0.8,\ C(a_3) = 1$

Code: 10010101

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



Decoded message: $a_1 a_2 a_1 a_3$

Decoding Issues

- How do we know that the message ended?
- We have two options:
  - Transmit the length of the message
  - Transmit a unique symbol denoting the end of the message, like EOF is used for files

Practical Arithmetic Coding

- All that has been described will work, but is not efficient
- We will look at two improvements:
  - *Scaling*: to avoid floating point underflow, like we did for decimal-to-binary conversion
  - *Integer Arithmetic*: avoids using floating point numbers altogether

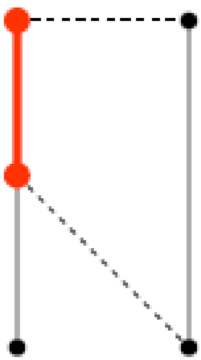
Scaling

- By scaling we can keep the *interval* we are working $[L, R)$ in a reasonable range of values so that its width W does not become very small a.k.a. *underflow*
- The *encoder* can produce the codeword bit by bit, and doesn't have to wait till the end of the message to convert the tag to binary and keep the top k MSB
- The *decoder* is more complicated

Encoding with Scaling

- During encoding, once the interval is confined in the *bottom* half [0, 0.5) it stays there forever
- Any number in that interval starts with a 0 in the MSB
- So we can just transmit a 0 and scale the interval

```
Lower half
If [L,R) is contained in [0,.5) then
   L = 2L; R = 2R
   output 0, followed by C 1's
   C = 0.
```

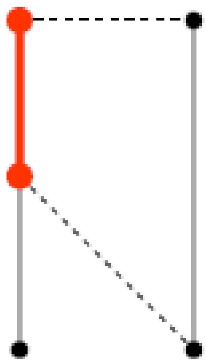


We will talk about the *C* 's later.

Encoding with Scaling

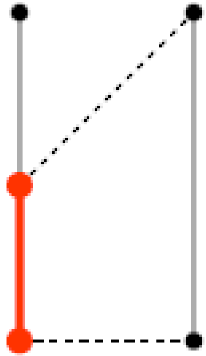
Lower half

```
If [L,R) is contained in [0,.5) then
   L = 2L; R = 2R
   output 0, followed by C 1's
   C = 0.
```



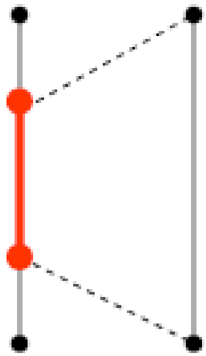
Upper half

```
If [L,R) is contained in [.5,1) then
   L = 2L - 1, R = 2R - 1
   output 1, followed by C 0's
   C = 0
```

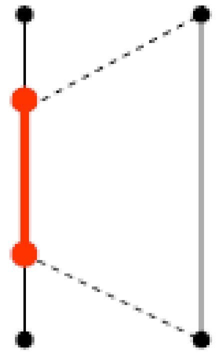


Middle Half

```
If [L,R) is contained in [.25,.75) then
   L = 2L - 0.5, R = 2R - 0.5
   C = C + 1.
```



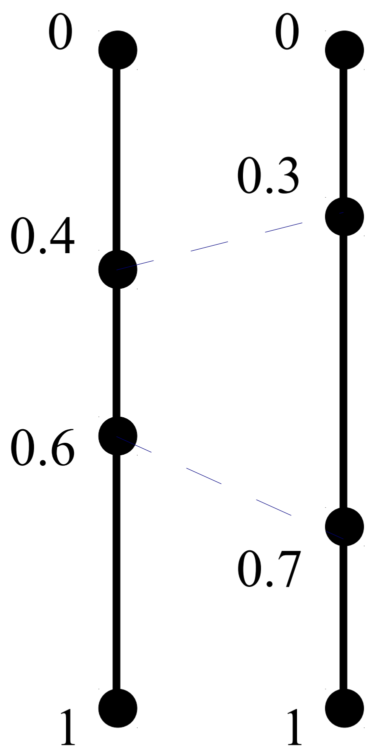
Encoding with Scaling



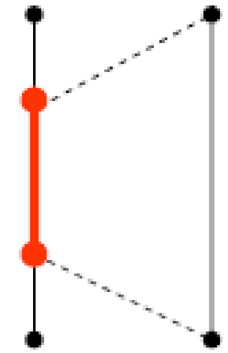
Why do we keep track of this scaling?

```
Middle Half
If [L,R) is contained in [.25,.75) then
   L = 2L – 0.5, R = 2R - 0.5
   C = C + 1.
```

Assume [L, R) = [0.4, 0.6) i.e. we need to apply this scaling and set C=1



Encoding with Scaling

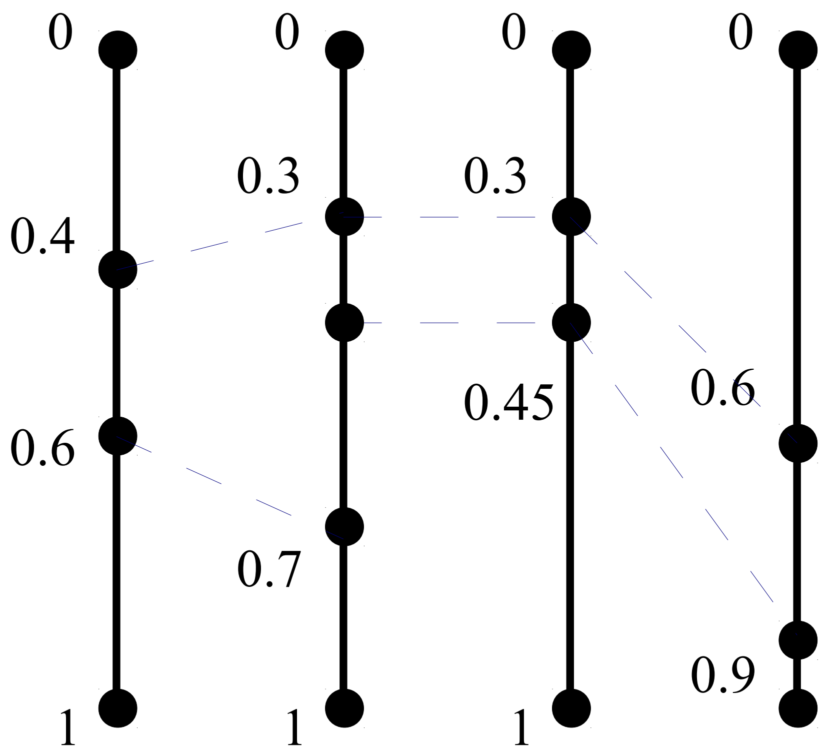


Why do we keep track of this scaling?

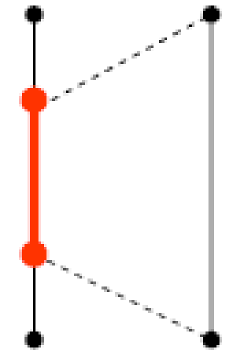
```
Middle Half
If [L,R) is contained in [.25,.75) then
    L = 2L – 0.5, R = 2R - 0.5
    C = C + 1.
```

Assume [L, R) = [0.4, 0.6) i.e. we need to apply this scaling and set C=1

After that [L, R) is confined in the *lower* half [0, 0.5) which is then scaled and emit 01



Encoding with Scaling

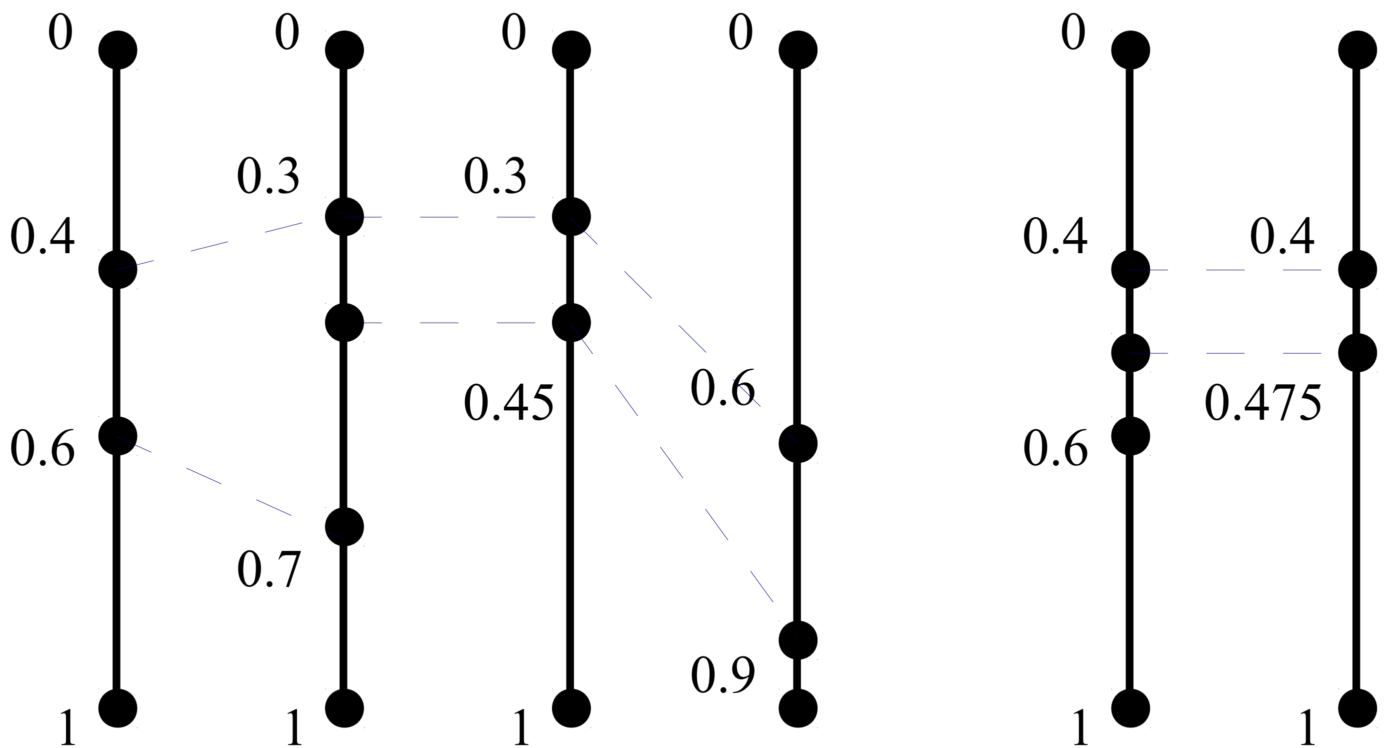


Why do we keep track of this scaling?

```
Middle Half
If [L,R) is contained in [.25,.75) then
   L = 2L - 0.5, R = 2R - 0.5
   C = C + 1.
```

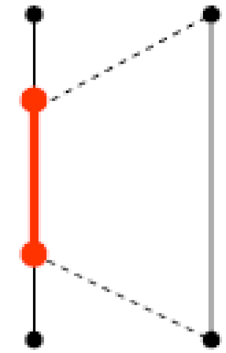
Now assume we haven't scaled [L, R) = [0.4, 0.6)

After that [L, R) would be [0.4, 0.475) (instead of [0.3, 0.45) with scaling)



Encoding with Scaling

Why do we keep track of this scaling?

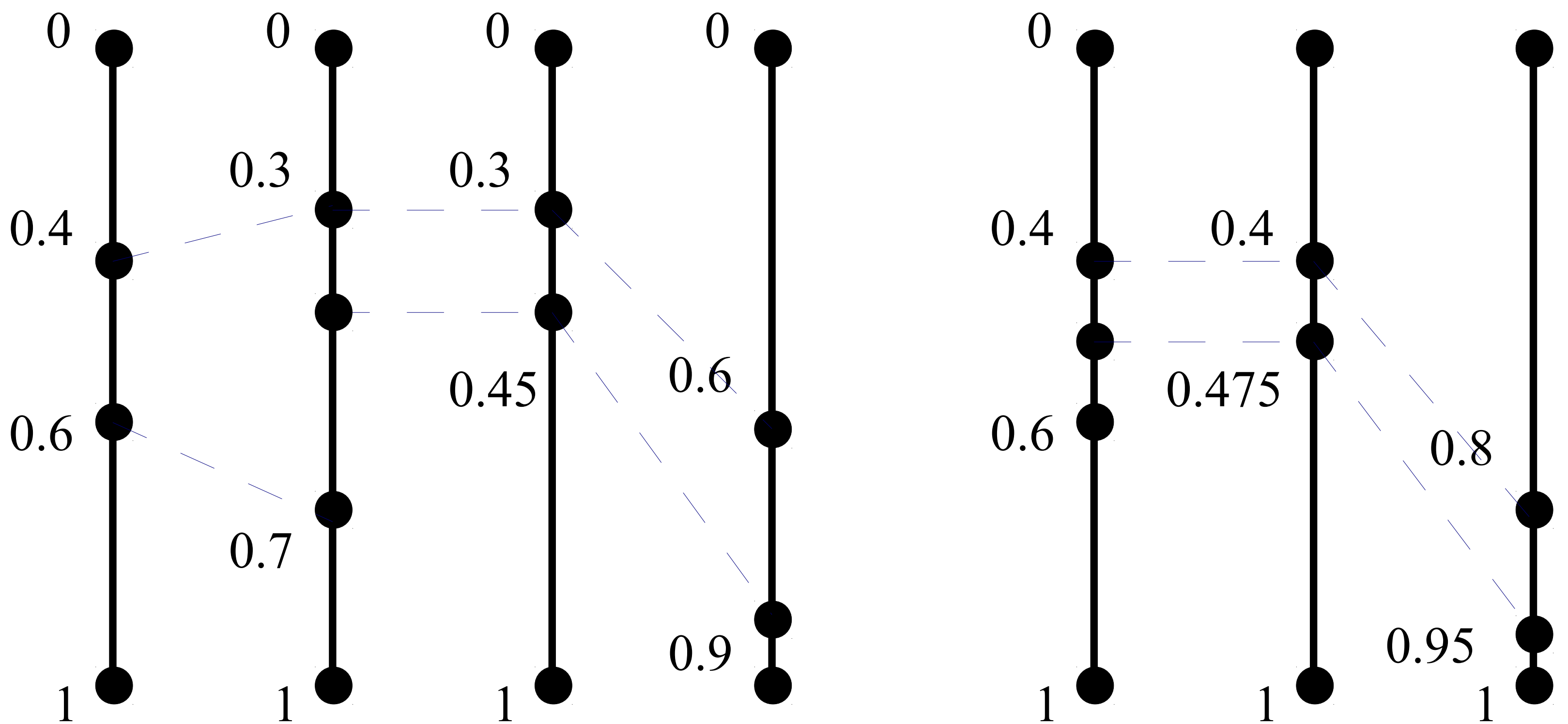


```
Middle Half
If [L,R) is contained in [.25,.75) then
   L = 2L – 0.5, R = 2R - 0.5
   C = C + 1.
```

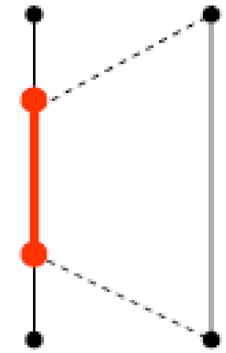
Now assume we haven't scaled [L, R) = [0.4, 0.6)

After that [L, R) would be [0.4, 0.475) (instead of [0.3, 0.45) with scaling)

This is confined in the *lower* half $\rightarrow$ scale and emit 0



Encoding with Scaling



Why do we keep track of this scaling?

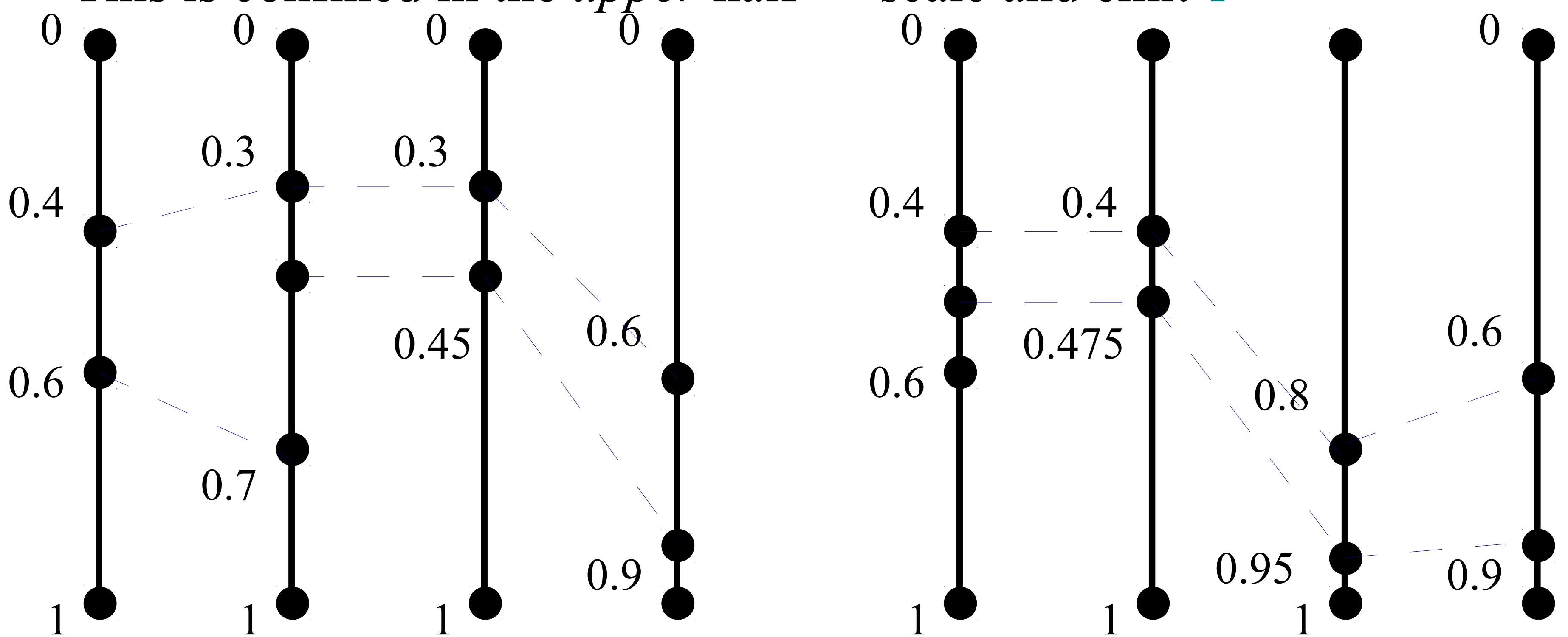
```
Middle Half
If [L,R) is contained in [.25,.75) then
    L = 2L - 0.5, R = 2R - 0.5
    C = C + 1.
```

Now assume we haven't scaled [L, R) = [0.4, 0.6)

After that [L, R) would be [0.4, 0.475) (instead of [0.3, 0.45) with scaling)

This is confined in the *lower* half → scale and emit 0

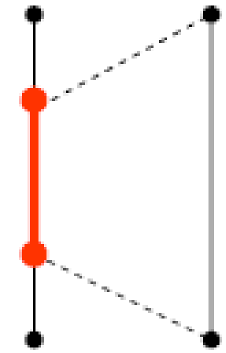
This is confined in the *upper* half → scale and emit 1



Encoding with Scaling

Why do we keep track of this scaling?

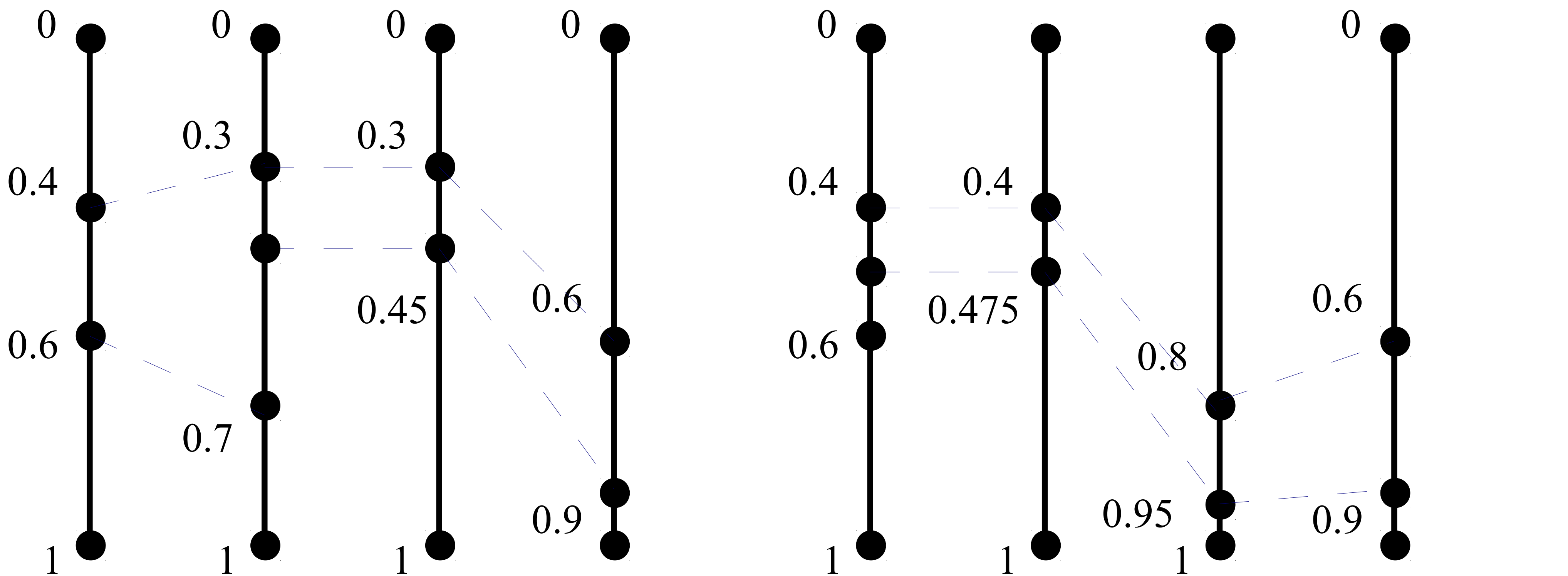
```
Middle Half
If [L,R) is contained in [.25,.75) then
   L = 2L – 0.5, R = 2R - 0.5
   C = C + 1.
```



Scaling the *middle* half followed by scaling the *lower* half (emit 01)

⇕

Scaling the *lower* half followed by scaling the *upper* half (emit 01)



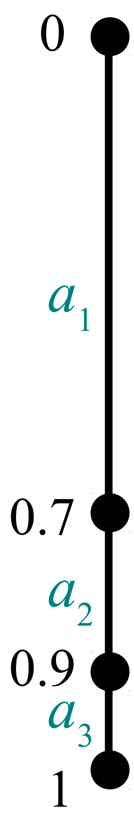
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



W = 1
L = 0
R = 1

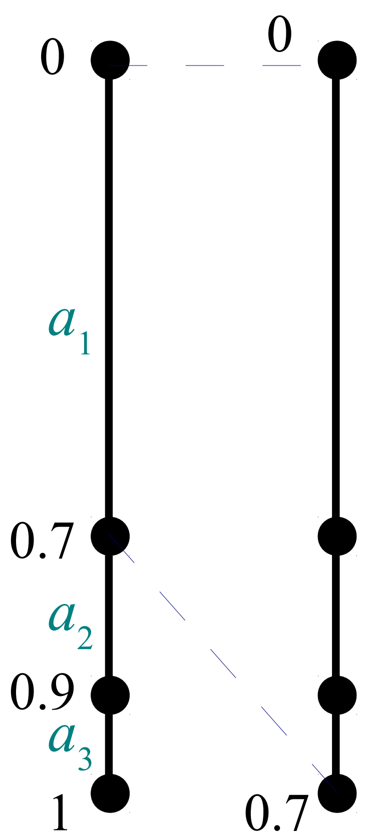
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



$i = 1$

$W = 1$
$L = 0$
$R = 0.7$

$C(x_{i-1}) = 0$
$P(x_i) = 0.7$

$C = 0$

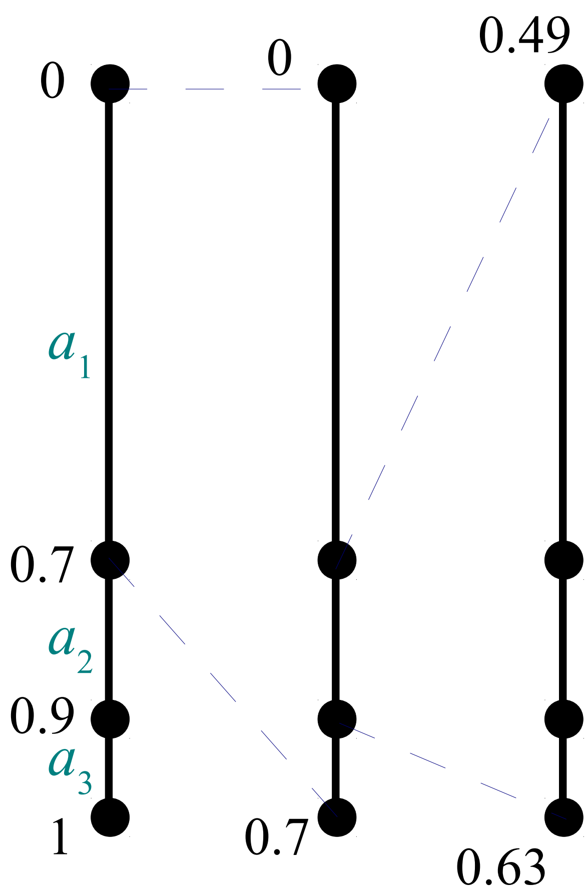
Example

$P(a_1) = 0.7, P(a_2) = 0.2, P(a_3) = 0.1$

$C(a_1) = 0.7, C(a_2) = 0.9, C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



[L,R) in [0.25, 0.75)

i = 2

W = 0.7
L = 0.49
R = 0.63

$C(x_{i-1}) = 0.7$
$P(x_i) = 0.2$

C = 0

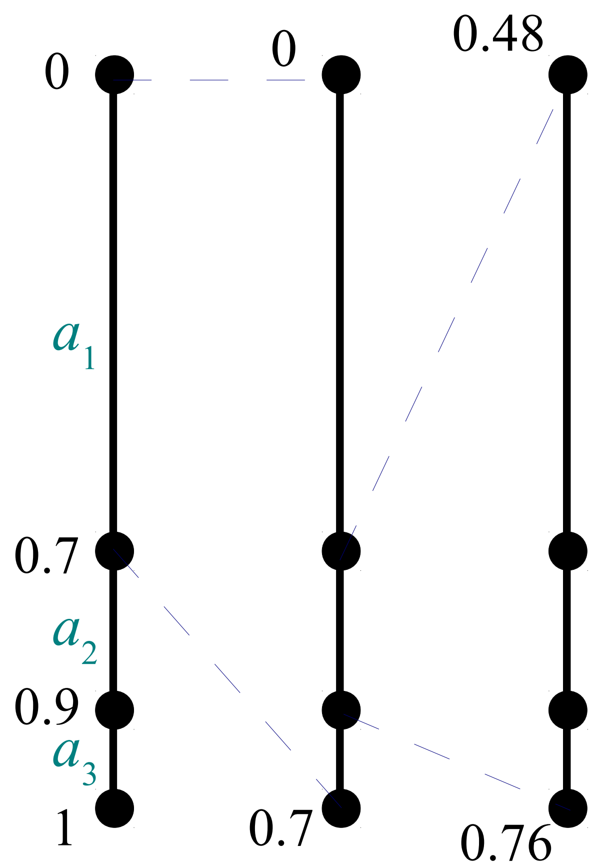
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



Scale

i = 2

W = 0.7
L = 0.48
R = 0.76

$C(x_{i-1}) = 0.7$
$P(x_i) = 0.2$

C = 1

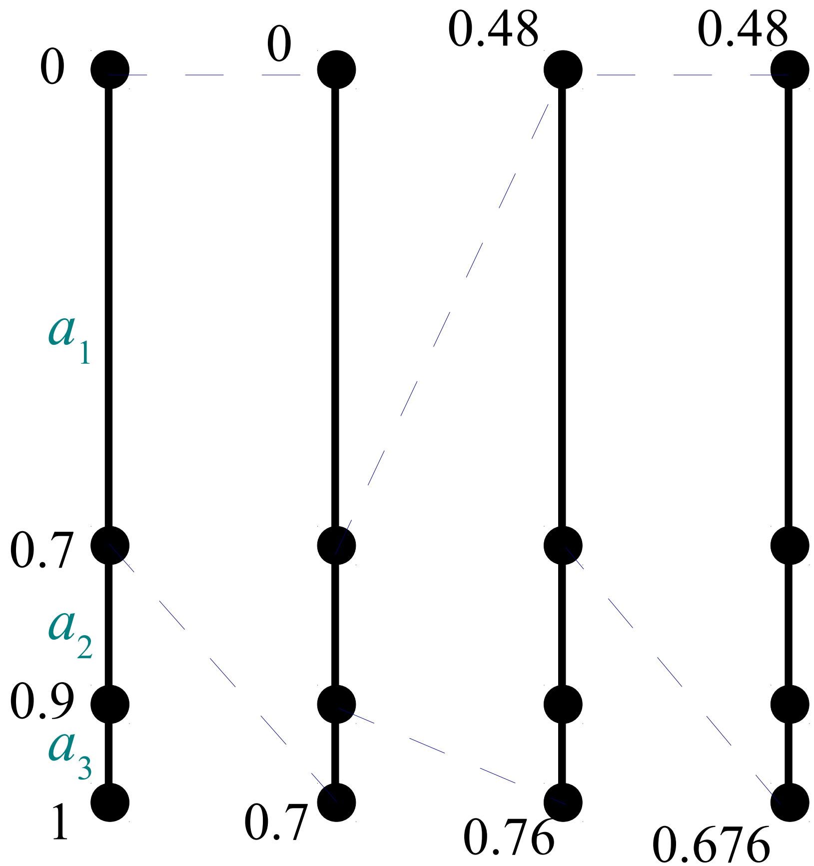
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



i = 3

[L, R) within [0.25, 0.75)

W = 0.28
L = 0.48
R = 0.676

$C(x_{i-1}) = 0$
$P(x_i) = 0.7$

C = 1

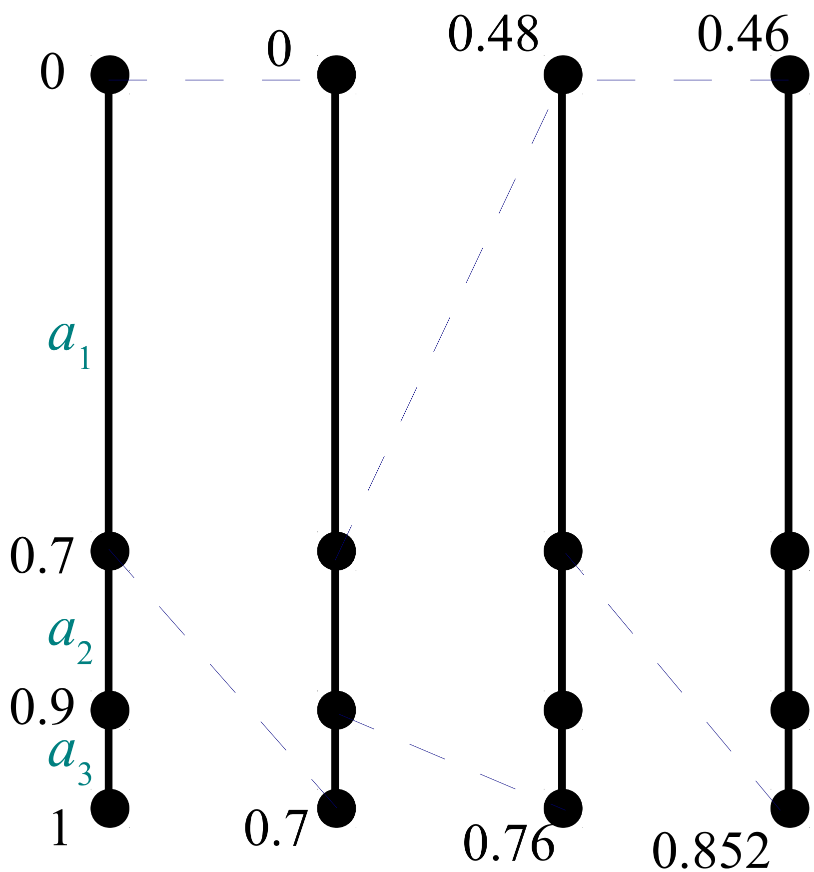
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



Scale

i = 3

W = 0.28
L = 0.46
R = 0.852

$C(x_{i-1}) = 0$
$P(x_i) = 0.7$

C = 2

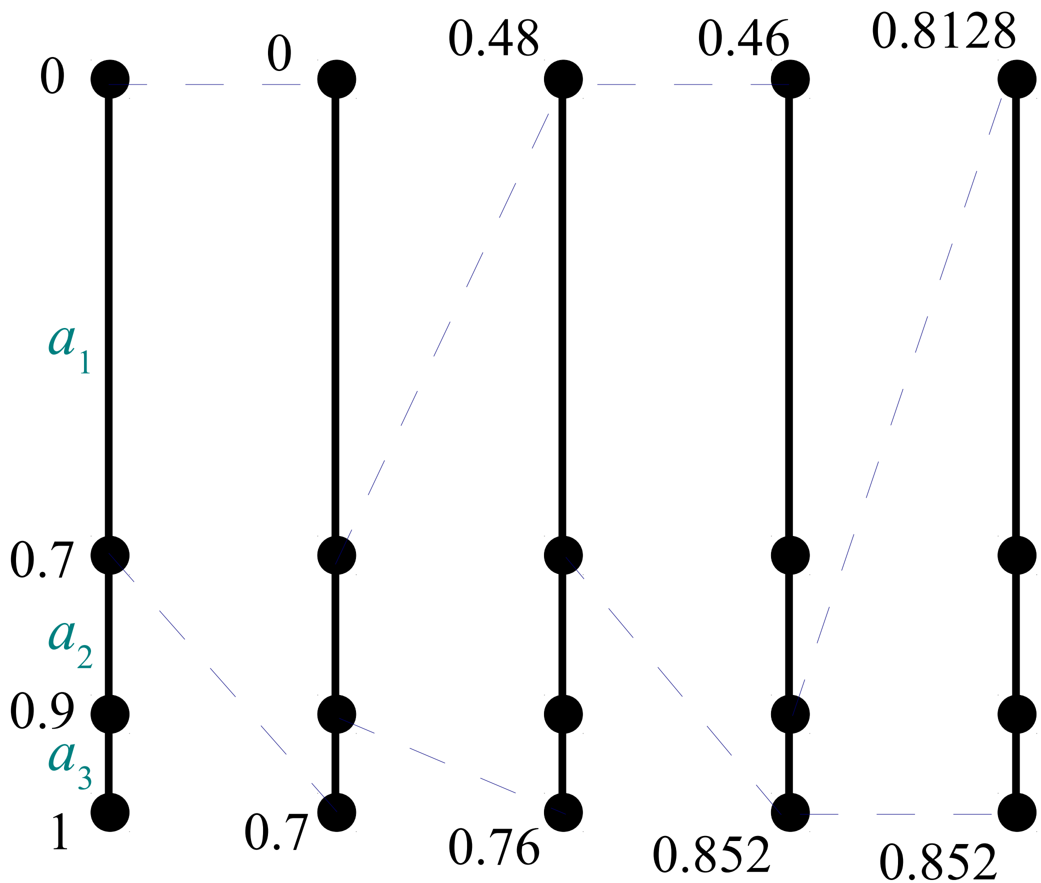
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



[L, R) within [0.5, 1)

i = 4

W = 0.392
L = 0.8128
R = 0.852

$C(x_{i-1}) = 0.9$
$P(x_i) = 0.1$

C = 2

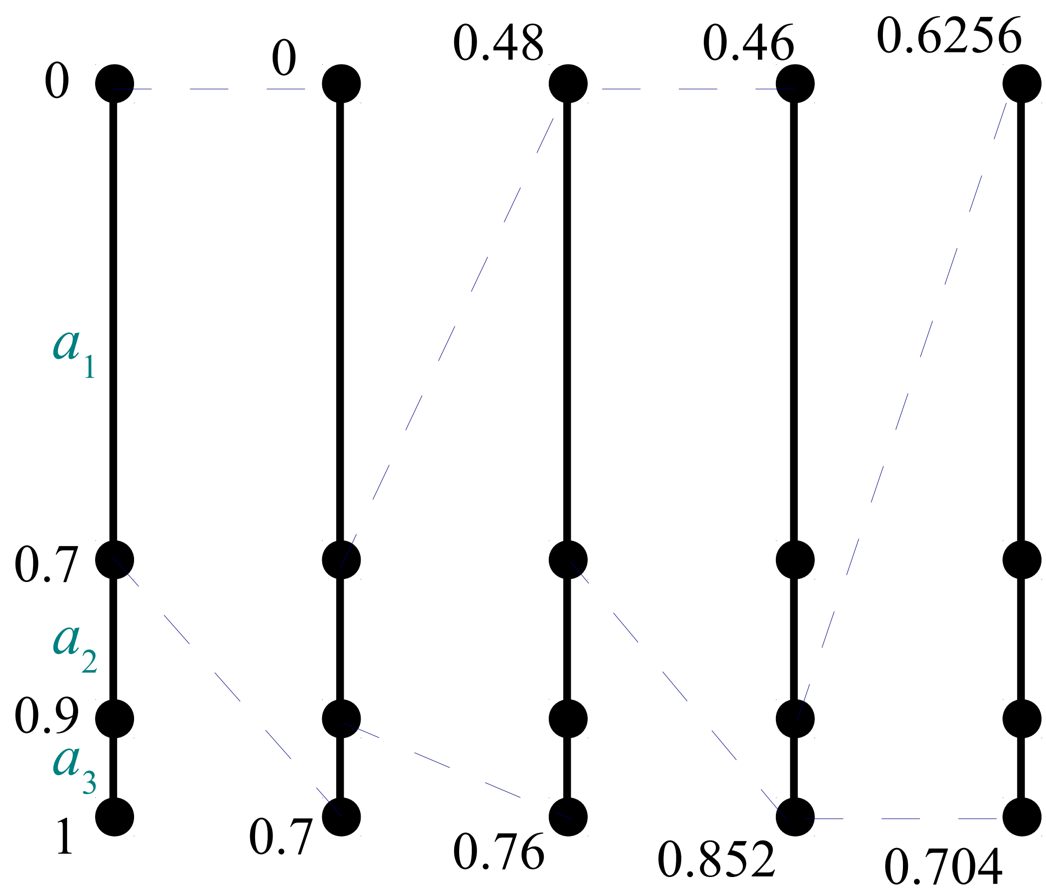
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



Scale and emit 100

$i = 4$

$W = 0.392$
$L = 0.6256$
$R = 0.704$

$C(x_{i-1}) = 0.9$
$P(x_i) = 0.1$

$C = 0$

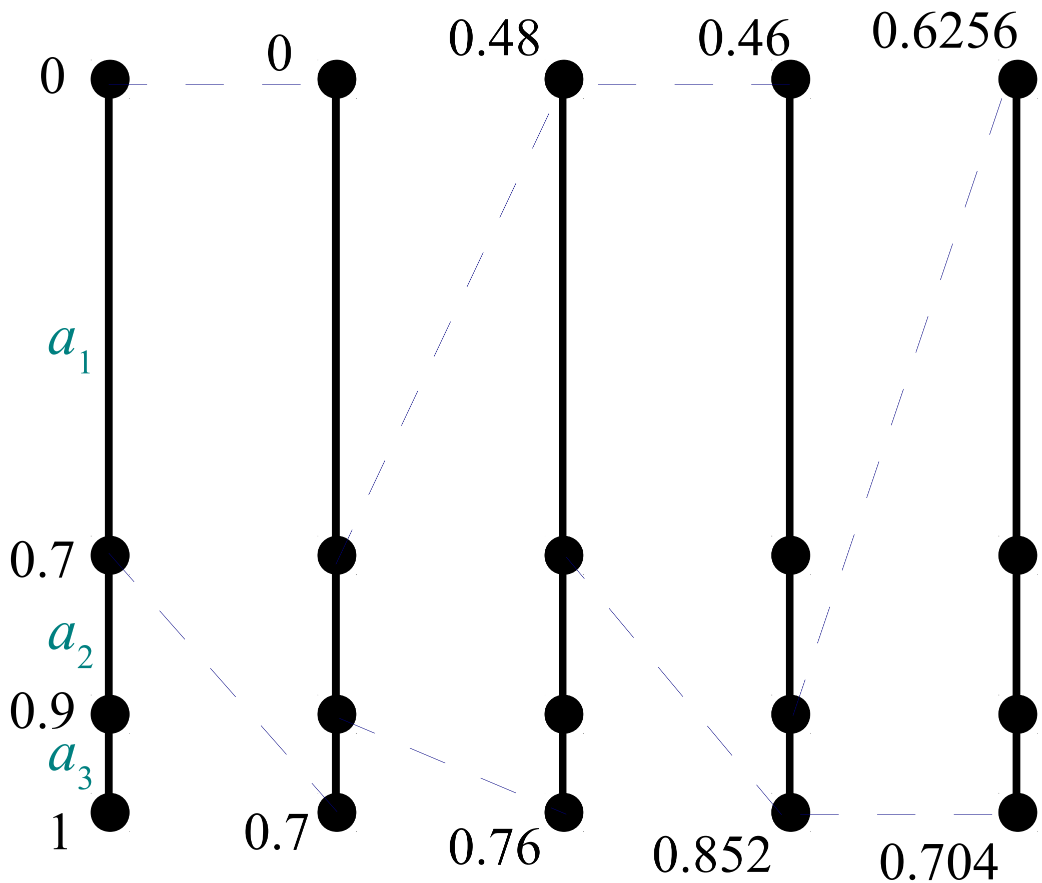
Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



L = 0.6256
R = 0.704

Already emitted 100, now choose any value within the interval e.g. 0.6875 = 0.1011

The codeword then becomes 10010110 = 0.5859

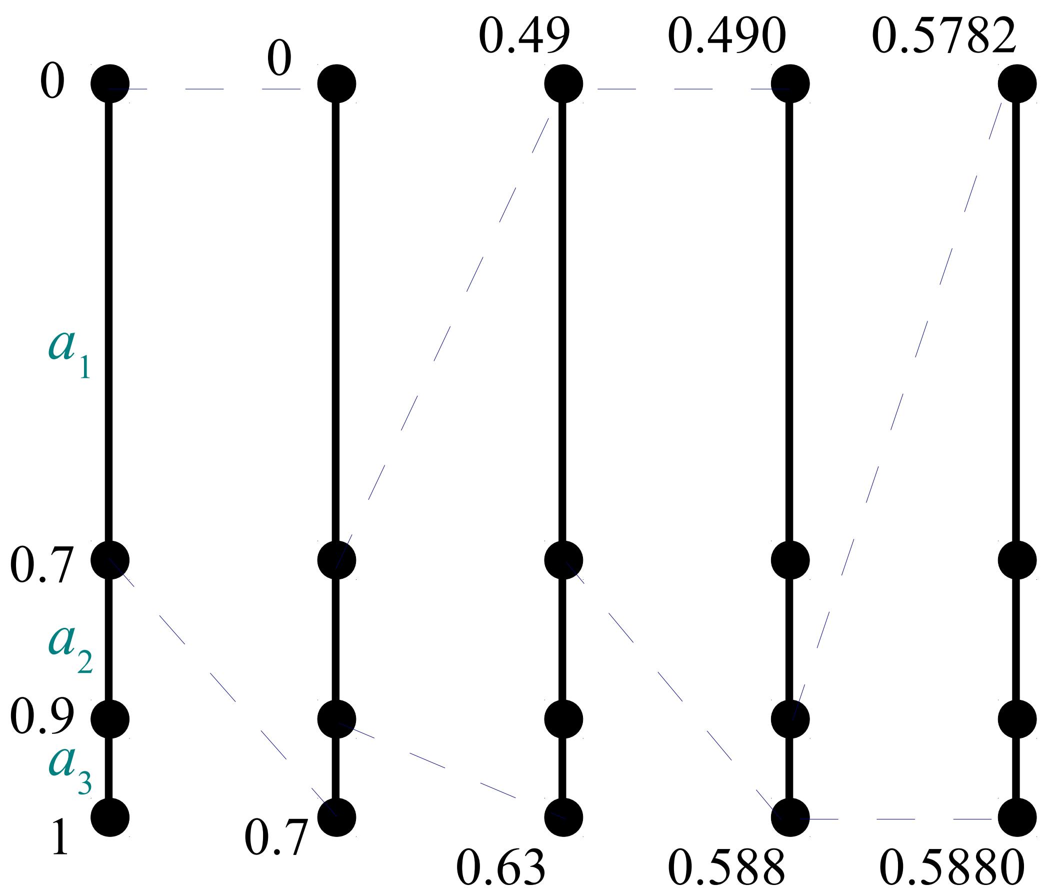
Previous Example

$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R= 1;
for i = 1 to n do
    W = R - L;
    L = L + W * C(x_{i-1});
    R = L + W * P(x_i);
t = (L+R)/2;
choose code for the tag
```



L = 0.5782
R = 0.5880

t = 0.5859 lies within the interval!

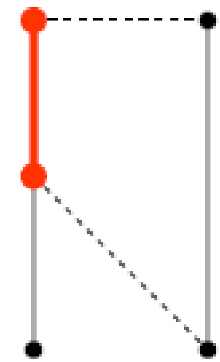
Decoding with Scaling

- The decoder behaves exactly the same as the encoder, except that it doesn't keep track of the C values
- Instead, the input stream is *consumed* during the scaling

Decoding with Scaling

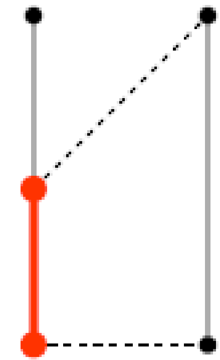
Lower half

```
If [L,R) is contained in [0,.5) then
    L = 2L; R = 2R
    consume 0 from the input stream
```



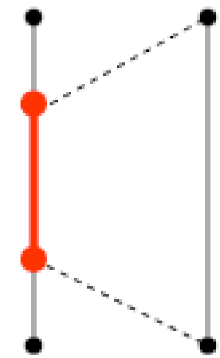
Upper half

```
If [L,R) is contained in [.5,1) then
    L = 2L - 1, R = 2R - 1
    consume 1 from the input stream
```



Middle Half

```
If [L,R) is contained in [.25,.75) then
    L = 2L - 0.5, R = 2R - 0.5
    Replace 01 with 0 on the stream
    Replace 10 with 1 on the stream
```



Tag

What we are actually doing to the tag:

- Lower Half

$$0.0\mathrm{b}_1 b_2 \cdots \times 10 = 0.\mathrm{b}_1 b_2 \cdots$$

- Upper Half

$$0.1\mathrm{b}_1 b_2 \cdots \times 10 - 1 = 0.\mathrm{b}_1 b_2 \cdots$$

- Middle Half

$$0.10\mathrm{b}_1 b_2 \cdots \times 10 - 0.1 = 0.1\mathrm{b}_1 b_2 \cdots$$

$$0.01\mathrm{b}_1 b_2 \cdots \times 10 - 0.1 = 0.0\mathrm{b}_1 b_2 \cdots$$

Example

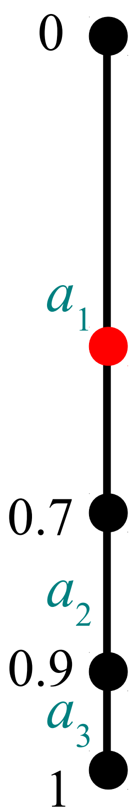
$P(a_1) = 0.7$, $P(a_2) = 0.2$, $P(a_3) = 0.1$

$C(a_1) = 0.7$, $C(a_2) = 0.9$, $C(a_3) = 1$

Code: 10010110

Decoded message:

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



i = 1

W = 1
L = 0
R = 1

t = 0.5859375

Example

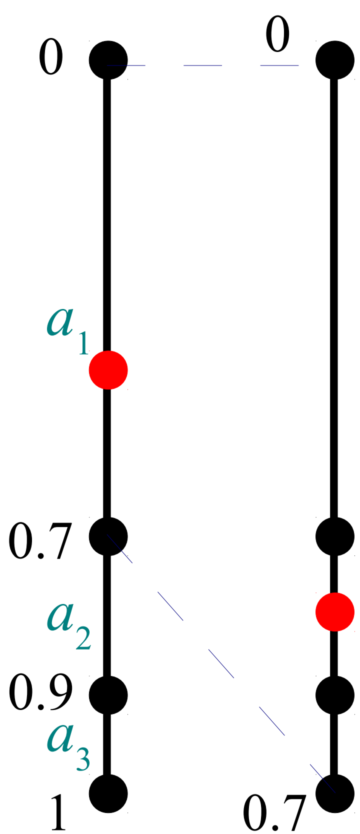
$P(a_1) = 0.7$, $P(a_2) = 0.2$, $P(a_3) = 0.1$

$C(a_1) = 0.7$, $C(a_2) = 0.9$, $C(a_3) = 1$

Code: 10010110

Decoded message: a_1

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



$i = 2$

$W = 0.7$
$L = 0$
$R = 0.7$

$j = 1$

$t = 0.5859375$

Example

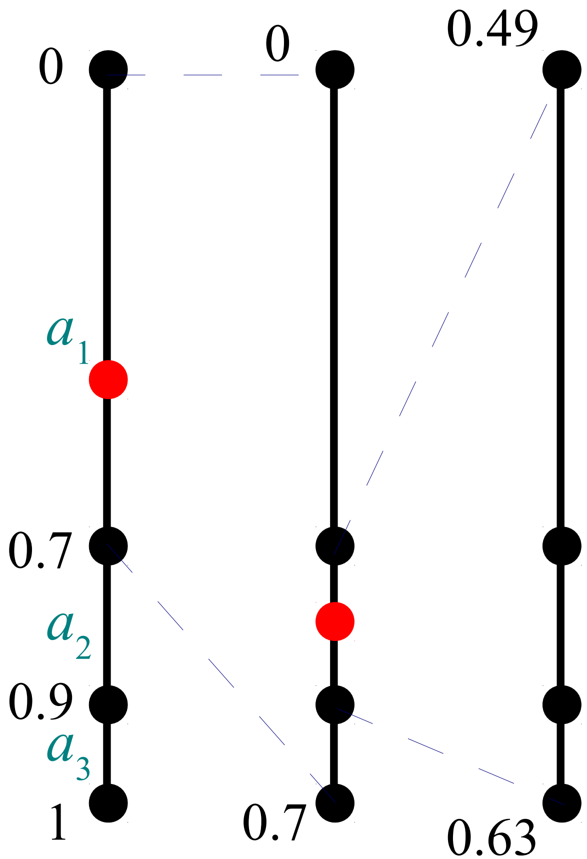
$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Code: 10010110

Decoded message: $a_1 a_2$

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



[L,R) within [0.25, 0.75)

i = 3

W = 0.14
L = 0.49
R = 063

j = 2

t = 0.5859375

Example

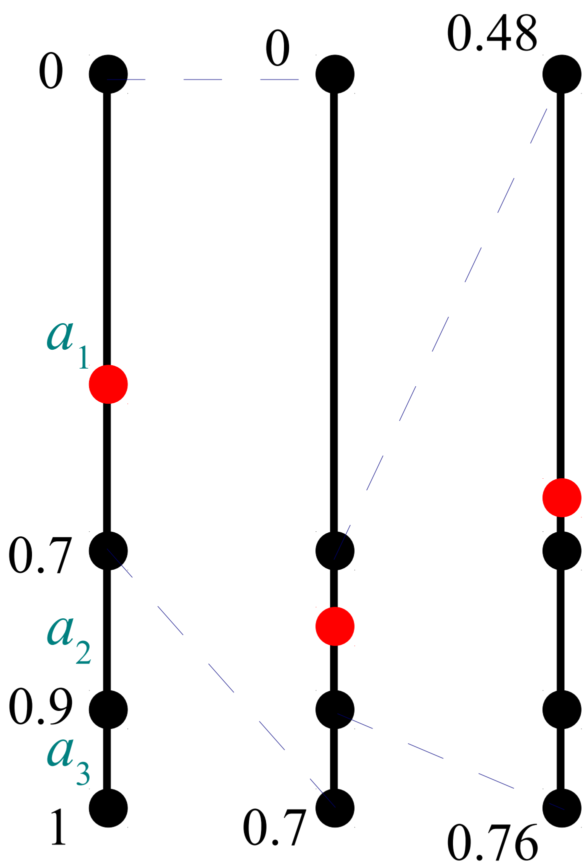
$P(a_1) = 0.7$, $P(a_2) = 0.2$, $P(a_3) = 0.1$

$C(a_1) = 0.7$, $C(a_2) = 0.9$, $C(a_3) = 1$

Code: ~~10~~ 1010110

Decoded message: $a_1 a_2$

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



Scale and change 10 to 1 and update the tag t

$i = 3$

$W = 0.28$
$L = 0.48$
$R = 0.76$

$j = 1$

$t = 0.671875$

Example

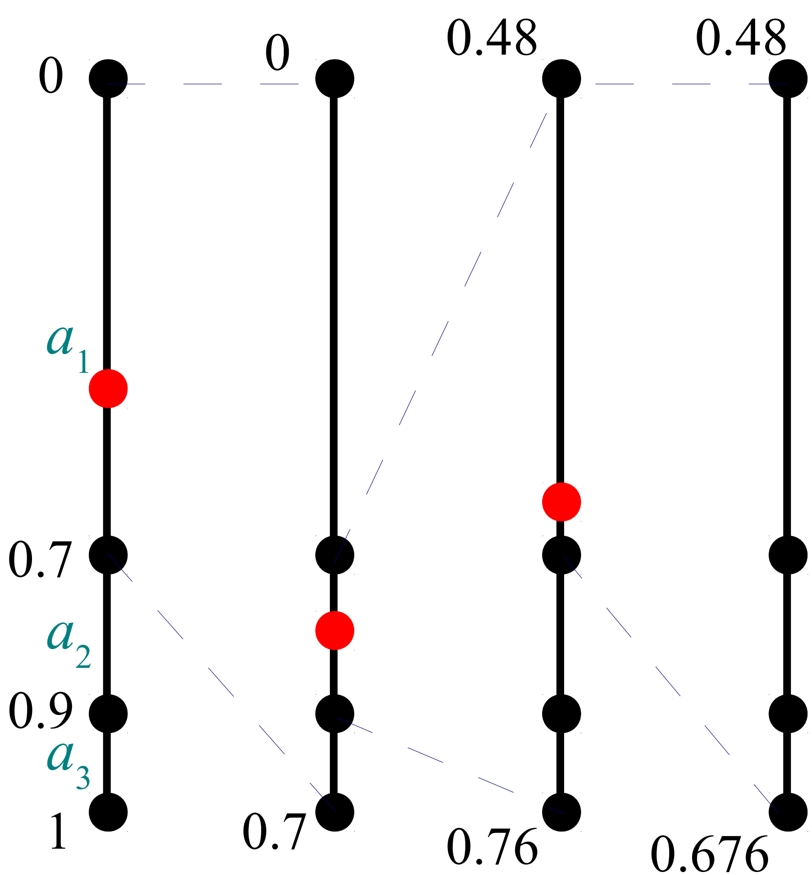
$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Code: ~~10~~ 1010110

Decoded message: $a_1 a_2 a_1$

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



[L,R) within [0.25, 0.75)

i = 4

W = 0.28
L = 0.48
R = 0.676

j = 1

t = 0.671875

Example

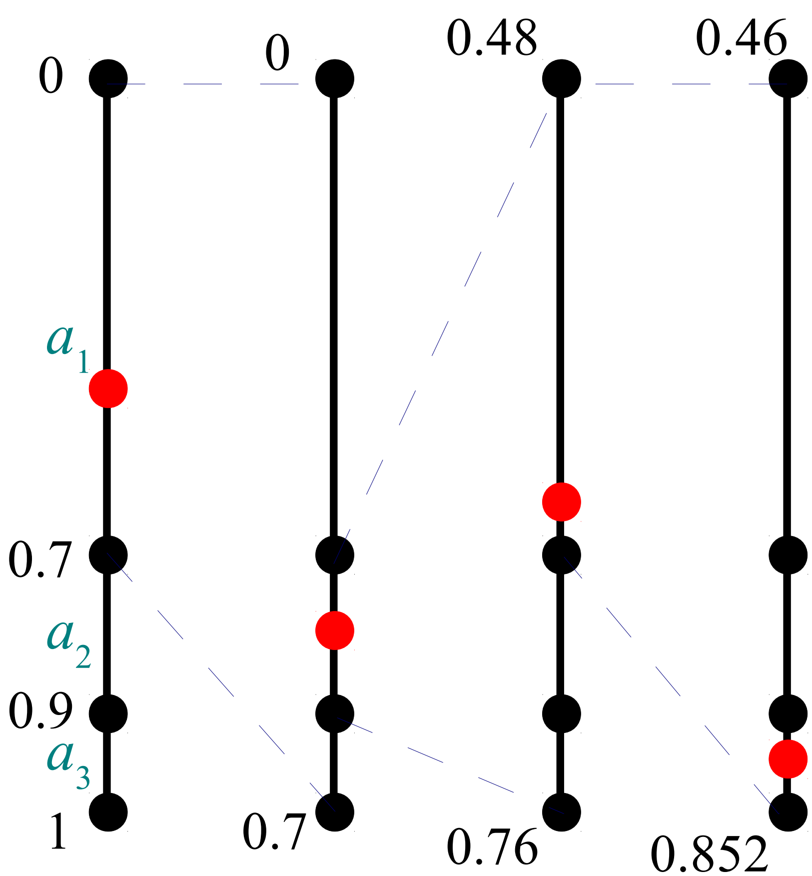
$P(a_1) = 0.7$, $P(a_2) = 0.2$, $P(a_3) = 0.1$

$C(a_1) = 0.7$, $C(a_2) = 0.9$, $C(a_3) = 1$

Code: ~~101~~ 110110

Decoded message: $a_1 a_2 a_1$

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



Scale and change 10 to 1 and update the tag t

$i = 4$

$W = 0.392$
$L = 0.46$
$R = 0.852$

$j = 1$

$t = 0.84375$

Example

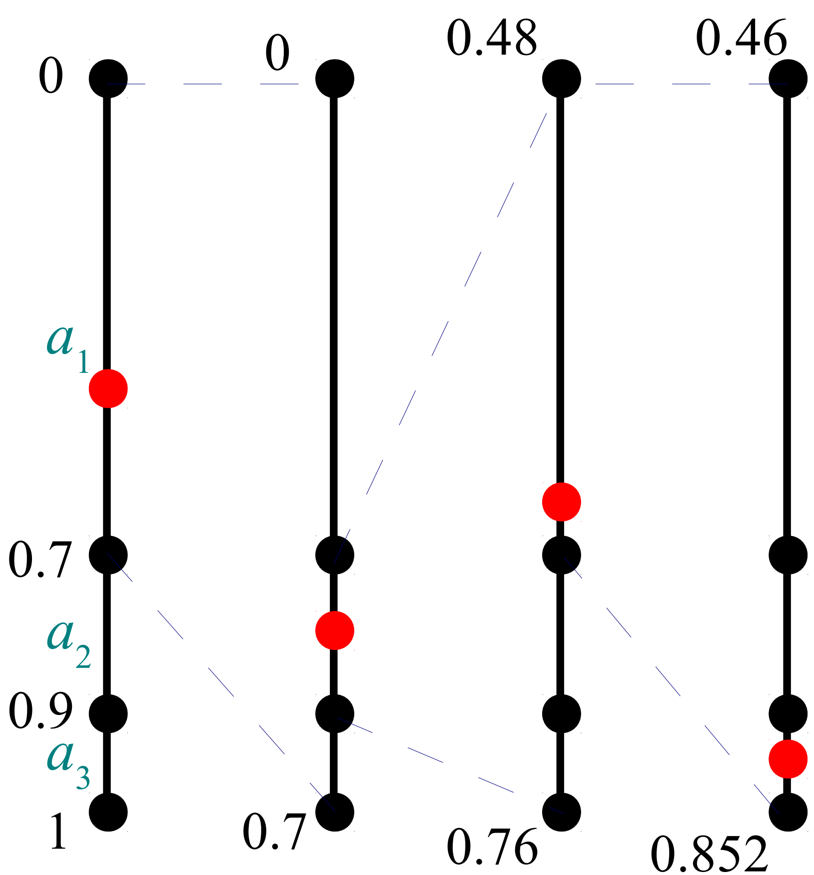
$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Code: ~~101~~ 110110

Decoded message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



$i = 4$

$W = 0.392$
$L = 0.46$
$R = 0.852$

$j = 3$

$t = 0.84375$

Example

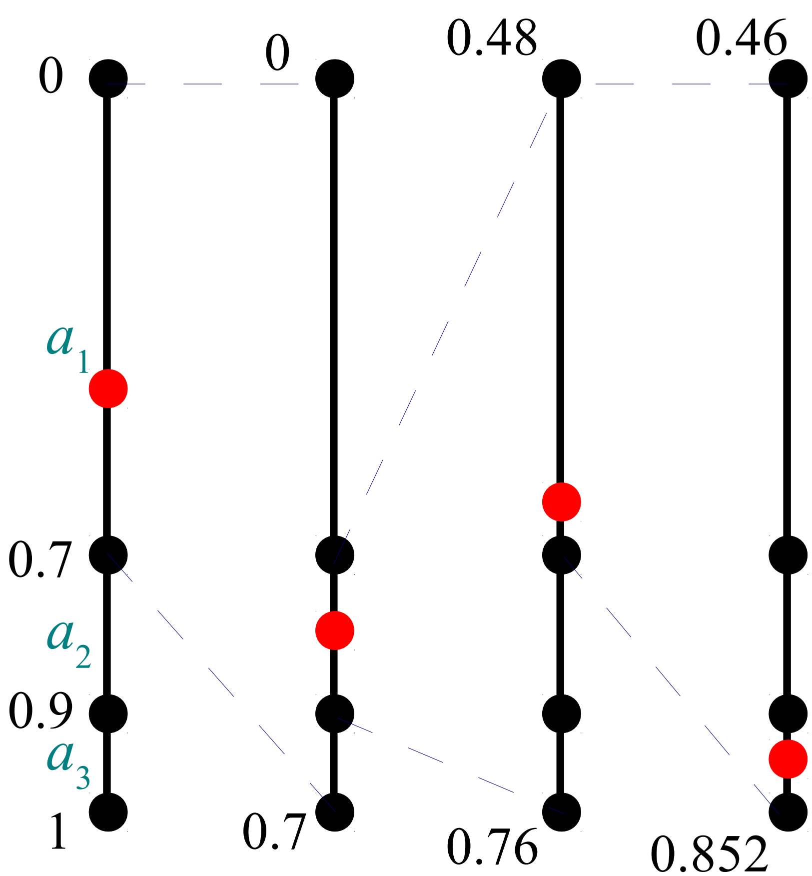
$P(a_1) = 0.7,\ P(a_2) = 0.2,\ P(a_3) = 0.1$

$C(a_1) = 0.7,\ C(a_2) = 0.9,\ C(a_3) = 1$

Code: ~~101~~ 110110

Decoded message: $a_1 a_2 a_1 a_3$

```
Initialize L = 0 and R = 1;
t = .b1b2...bk000...
for i = 1 to n do
     W = R - L;
     find j such that:
          L + W * C(aj-1) <= t < L + W * (C(aj-1)+P(aj))
     output aj;
     L = L + W * C(aj-1);
     R = L + W * P(aj);
```



Now we are done decoding a sequence of 4 symbols. We get rid of the rest of the bits and work on a new batch of 4 symbols...

Integer Arithmetic Coding

- Using integer operations is faster than floating point operations
- Decide on the size of the integers to have m bits e.g. 8
- Map the fractions in the interval $[0, 1]$ into values $[0, 2^m-1]$
- For example with $m = 8$:

  $0 \rightarrow 0 \quad = 0000\ 0000$

  $1 \rightarrow 255 \quad = 1111\ 1111$

  $0.5 \rightarrow 128 \quad = 1000\ 0000$

- Represent the probabilities by their frequencies
  - Let n_i be the number of times symbol a_i occurs

Integer Arithmetic Coding

- Represent the *probabilities* by symbol *frequencies*
  - Let n_i be the number of times symbol a_i occurs
  - Define $C_i = \sum_{j=1}^{i} n_i$
  - Let N be the total size of the message
- Modify the encoding/decoding algorithm to use integer operations with frequencies instead of probabilities, together with scaling

Integer AC Encoder

```
Initialize L = 0 and R = 2^m - 1;
for i = 1 to n do
    W = R – L + 1;
    L = L + W * C_(i-1) / N;
    R = L + W * n_i / N - 1;

    // Scaling
    while(we need rescaling)
        // If [L, R) in [0, 0.5) or [0.5, 1)
        If MSB(L) = MSB(R) = b
            // Shift left
            L <<= 1;
            // Shift left and add 1 as LSB
            R <<= 1; R |= 1
            Emit b
            Emit !b C times

        // If [L, R) in [0.25, 0.75)
        // by checking second MSB
        Else SecondMSB(L)=1 and SecondMSB(R)=0
            // Shift left and complement MSB
            L <<= 1; L ^= 2^(m-1)
            R <<= 1; R ^= 2^(m-1); R |= 1;
            C += 1
```

Integer AC Decoder

```
Initialize L = 0 and R = 2^m - 1;
Read the first m bits into tag t

for i = 1 to n do
    Find j such that:
        L + W * C_{j-1} / N <= t < L + W * n_j / N - 1
    Output symbol a_j

    L = L + W * C_{j-1} / N;
    R = L + W * n_j / N - 1;
    W = R - L + 1;

    // Scaling
    while(we need rescaling)
        // If [L, R) in [0, 0.5) or [0.5, 1)
        If MSB(L) = MSB(R) = b
            // Shift left
            L <<= 1;
            // Shift left and add 1 as LSB
            R <<= 1; R |= 1
            // Shift left tag t
            t <<= 1 and put next bit into LSB

        // If [L, R) in [0.25, 0.75)by checking second MSB
        Else SecondMSB(L)=1 and SecondMSB(R)=0
            // Shift left and complement MSB
            L <<= 1; L ^= 2^(m-1)
            R <<= 1; R ^= 2^(m-1); R |= 1;
            t <<= 1; t ^= 2^(m-1);
```

Adaptive Arithmetic Coding

- What if the symbol *probabilities* are not known in advance?
- We can use an adaptive scheme:
  - Start with a *count* of 1 for every symbol in the alphabet
  - Run the integer encoder, and update the counts *after* every symbol is encoded
  - The decoder does the same
- No need to send probabilities/frequencies to receiver
- This can be done for both *integer* and *floating point* versions of Arithmetic Coding

Applications

- Arithmetic Coding is used in many lossless and lossy compression standards
- Example:

TABLE 4.7 Compression using adaptive arithmetic coding of pixel values.

Image Name	Bits/Pixel	Total Size (bytes)	Compression Ratio (arithmetic)	Compression Ratio (Huffman)
Sena	6.52	53,431	1.23	1.16
Sensin	7.12	58,306	1.12	1.27
Earth	4.67	38,248	1.71	1.67
Omaha	6.84	56,061	1.17	1.14

TABLE 4.8 Compression using adaptive arithmetic coding of pixel differences.

Image Name	Bits/Pixel	Total Size (bytes)	Compression Ratio (arithmetic)	Compression Ratio (Huffman)
Sena	3.89	31,847	2.06	2.08
Sensin	4.56	37,387	1.75	1.73
Earth	3.92	32,137	2.04	2.04
Omaha	6.27	51,393	1.28	1.26

Huffman vs. Arithmetic Coding

- Both compress very well. For *m* symbol blocks:
  - Huffman within $1/m$ of entropy
  - Arithmetic within $2/m$ of entropy
- Adaptation
  - Adaptive Huffman Coding is much more complicated
  - Adaptive Arithmetic Coding is much simpler
- Skewed distributions and small alphabets
  - Arithmetic Coding much better
- Conclusion
  - Arithmetic coding is more versatile and flexible

Recap

- Golomb Coding
- Arithmetic Coding
- Next:
  - Dictionary based techniques

- More information: Chapter 4 [**IDC**]