Information Theory

Mohamed Hamada

Software Engineering Lab
The University of Aizu

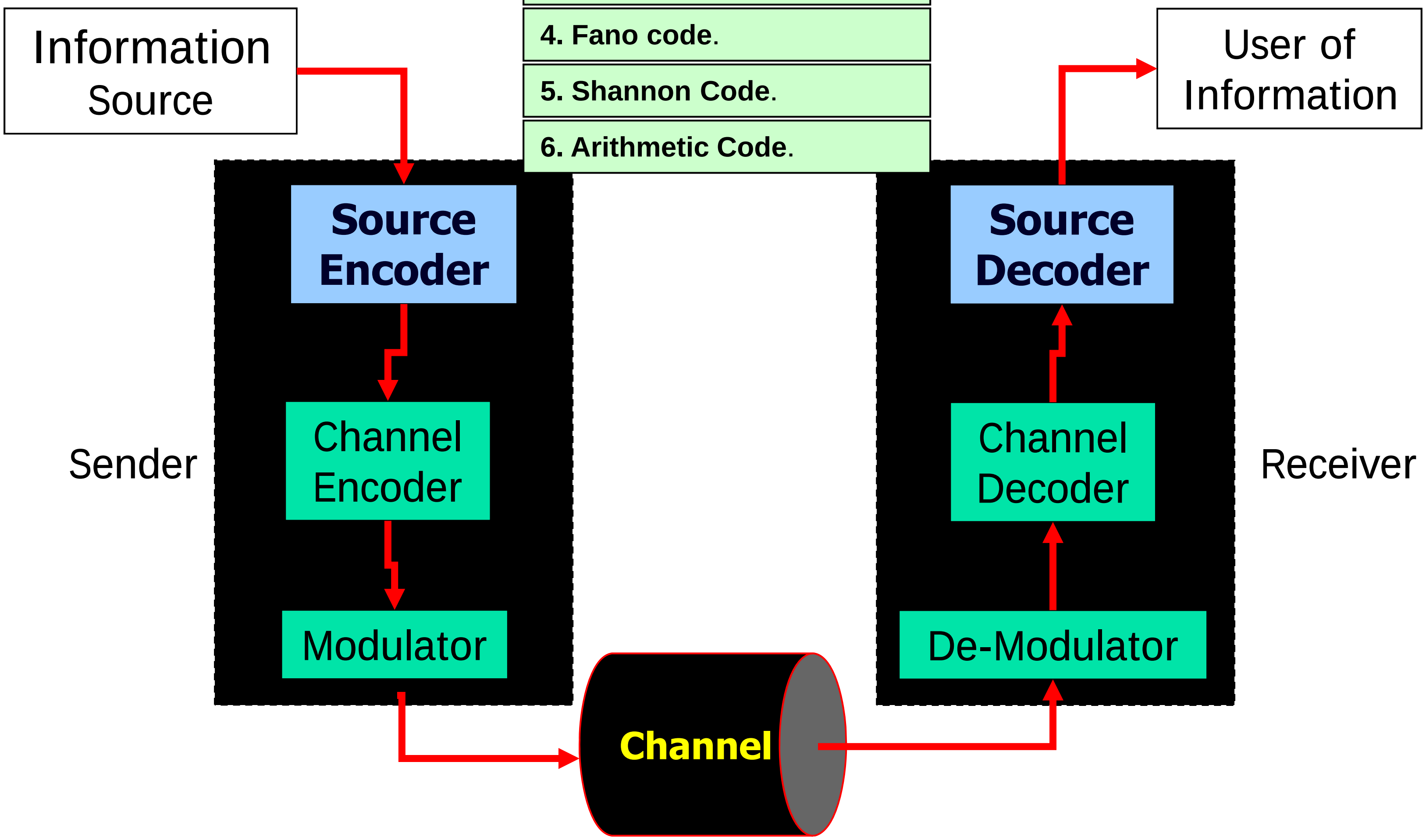
Email: hamada@u-aizu.ac.jp
URL: http://www.u-aizu.ac.jp/~hamada

Today's Topics

- **Source Coding Techniques**
- **Arithmetic Coding**
- **Arithmetic Vs. Huffman Coding**
- **Coding Examples**
- **Arithmetic Decoding**
- **Decoding Examples**

Source Coding Techniques

1. Huffman Code.
2. Two-pass Huffman Code.
3. Lemple-Ziv Code.
4. Fano code.
5. Shannon Code.
6. Arithmetic Code.



Source Coding Techniques

1. Huffman Code.

2. Two-pass Huffman Code.

3. Lemple-Ziv Code.

4. Fano code.

5. Shannon Code.

6. Arithmetic Code.

Source Coding Techniques

1. Huffman Code.

2. Two-path Huffman Code.

3. Lemple-Ziv Code.

4. Shannon Code.

5. Fano Code.

6. Arithmetic Code.

Arithmetic Coding

- String of characters with occurrence probabilities make up a message
- A complete message may be fragmented into multiple smaller strings
- A codeword corresponding to each string is found separately

Arithmetic Code

Coding

Arithmetic coding is a form of variable length entropy encoding that
Converts a message into another representation that represents
Frequently used characters using fewer bits and infrequently used
Characters using more bits, with the goal of using fewer bits in total

Arithmetic coding is notable for extremely high coding efficiency

Application: recent generation standards including
JPEG2000 and H.264
utilize arithmetic coding

Arithmetic Vs. Huffman Coding

The most common statistical compression methods are Huffman and Arithmetic coding.

Huffman utilizes a static table to represent all the characters and their frequencies, then generates a code table accordingly.

More frequent characters will be assigned shorter code,
and by doing so the source can be effective compressed.

Arithmetic coding works a bit differently from Huffman.
It also uses a statistical table for coding, but this table is
Adaptive:
it is modified from time to time to reflect the real time distribution statistics.

While a new character is being processed, the table will re-calculate frequencies until the end of the text stream.

Arithmetic Vs. Huffman Coding

Huffman uses a static table for the whole coding process, so it is rather fast, but does not produce an efficient compression ratio.

Arithmetic coding, on the other hand, has different features. It can generate a high compression ratio, but all the complex calculation takes much more time, resulting in a slower implementation.

The table below presents a simple comparison between these compression methods.

Compression Method	Arithmetic	Huffman
Compression Ratio	Very Good	Poor
Compression Speed	Slow	Fast
Decompression Speed	Slow	Fast
Memory Space	Very Low	Low
Compressed Pattern Matching	No	Yes
Permits Random Access	No	Yes

Arithmetic Vs. Huffman Coding

An ideal compression method should satisfy all those features given in the table.

Compression Method	Arithmetic	Huffman
Compression Ratio	Very Good	Poor
Compression Speed	Slow	Fast
Decompression Speed	Slow	Fast
Memory Space	Very Low	Low
Compressed Pattern Matching	No	Yes
Permits Random Access	No	Yes

The last two items are important considerations in information retrieval, as both features are key in a system's ability to search documents directly and randomly.

Without compressed pattern matching, a system would need to decompress the entire document prior to processing a user's query. Without random access, a system could not retrieve any part of a document until it completely decompressed the document from the very beginning.

Arithmetic Code

Coding

In arithmetic coding a message is encoded as a number from the interval [0, 1).

The number is found by expanding it according to the probability of the currently processed letter of the message being encoded.

This is done by using a set of interval ranges IR determined by the probabilities of the information source as follows:

$$IR = \{ [0, p_1), [p_1, p_1 + p_2), [p_1 + p_2, p_1 + p_2 + p_3), \ldots [p_1 + \ldots + p_{n-1}, p_1 + \ldots + p_n) \}$$

Putting $q_j = \sum_{i=1}^{j} p_i$ **we can write** $IR = \{ [0, q_1), [q_1, q_2), \ldots [q_{n-1}, 1) \}$

Arithmetic Code

Coding

In arithmetic coding these subintervals also determine the proportional division of any other interval [L, R) contained in [0, 1) into subintervals $IR_{[L,R]}$ as follows:

$$IR_{[L,R]} = \{ [L, L+(R-L)\, q_1), [L+(R-L)\, q_1, L+(R-L)\, q_2), [L+(R-L)\, q_2, L+(R-L)\, q_3), \ldots , [L+(R-L)\, P_{n-1}, L+(R-L)) \}$$

Using these definitions the arithmetic encoding is determined by the Following algorithm:

ArithmeticEncoding (Message)

1. CurrentInterval = [0, 1);

While the end of message is not reached

2. Read letter x_i from the message;

3. Divid CurrentInterval into subintervals $IR_{CurrentInterval}$;

Output any number from the CurrentInterval (usually its left boundary);

This output number uniquely encoding the input message.

Arithmetic Code

Coding

Example 1 **Consider the information source**

A	B	C	#
0.4	0.3	0.1	0.2

Then the input message ABBC#
has the unique encoding number 0.23608.

As we will see the explanation In the next slides

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Coding

Example 1

2. Read X_i

input message: A B B C #

1. CurrentInterval = [0, 1);

X_i	Current interval	Subintervals
A	[0, 1)	

A	B	C	#
0.4	0.3	0.1	0.2

Arithmetic Code

Coding

2. Read X_i

Example 1

input message: A B B C #

3. Divid CurrentInterval into subintervals $IR_{CurrentInterval}$;

X_i	Current interval	
A	[0, 1)	

$IR_{[0,1)}$ = {
[0, 0.4) , [0.4, 0.7),
[0.7, 0.8), [0.8, 1)
}

$$q_j = \sum_{i=1}^{j} p_i$$

$[L+(R-L)\, q_i,\ L+(R-L)\, q_{i+1})$

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Coding

2. Read X_i

Example 1

input message: A B B C #

3. Divid CurrentInterval into subintervals $IR_{CurrentInterval}$;

X_i	Current interval	Subintervals
A	[0, 1)	[0, 0.4) , [0.4, 0.7), [0.7, 0.8), [0.8, 1)

$IR_{[0,1)}$= {
[0, 0.4) , [0.4, 0.7),
[0.7, 0.8), [0.8, 1)
}

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Coding

Example 1

input message: A B B C #

No. 1

A	B	C	#
0.4	0.3	0.1	0.2

X_i	Current interval	Subintervals
A	[0, 1)	[0, 0.4) , [0.4, 0.7), [0.7, 0.8), [0.8, 1)
	[0, 0.4)	

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Coding

Example 1

2. Read X_i

input message: A B B C #

3. Divid CurrentInterval into subintervals $IR_{CurrentInterval}$;

X_i	Current interval	
A	[0, 1)	[0, 0.4) , [0.4, 0.7
B	[0, 0.4)	

$IR_{[0,0.4)}$ = {
[0, 0.16) , [0.16, 0.28),
[0.28, 0.32), [0.32, 0.4)
}

$$q_j = \sum_{i=1}^{j} p_i$$

$[L+(R-L)\, q_i,\ L+(R-L)\, q_{i+1})$

A	B	C	#
0.4	0.3	0.1	0.2

Arithmetic Code

Coding

2. Read X_i

Example 1

input message: A B B C #

3. Divid CurrentInterval into subintervals $IR_{CurrentInterval}$;

X_i	Current interval	Subintervals			
A	[0, 1)	[0, 0.4) ,	[0.4, 0.7),	[0.7, 0.8),	[0.8, 1)
B	[0, 0.4)	[0, 0.16) ,	[0.16, 0.28),	[0.28, 0.32),	[0.32, 0.4)

$IR_{[0,0.4)}$ = {
[0, 0.16) , [0.16, 0.28),
[0.28, 0.32), [0.32, 0.4)
}

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Coding

Example 1

input message: A B B C #

No. 2

A	B	C	#
0.4	0.3	0.1	0.2

X_i	Current interval	Subintervals			
A	[0, 1)	[0, 0.4) ,	[0.4, 0.7),	[0.7, 0.8),	[0.8, 1)
B	[0, 0.4)	[0, 0.16) ,	[0.16, 0.28),	[0.28, 0.32),	[0.32, 0.4)
	[0.16, 0.28)				

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Coding

Example 1

2. Read X_i

input message: A B B C #

3. Divid CurrentInterval into subintervals $IR_{CurrentInterval}$;

X_i	Current interval	Subintervals			
A	[0, 1)	[0, 0.4) ,	[0.4, 0.7),	[0.7, 0.8),	[0.8, 1)
B	[0, 0.4)	[0, 0.16) ,	[0.16, 0.28),	[0.28, 0.32),	[0.32, 0.4)
B	[0.16, 0.28)	[0.16, 0.208) ,	[0.208, 0.244),	[0.244, 0.256),	[0.256, 0.28)

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Coding

Example 1

input message: A B B C #

No. 2

A	B	C	#
0.4	0.3	0.1	0.2

X_i	Current interval	Subintervals
A	[0, 1)	[0, 0.4) , [0.4, 0.7), [0.7, 0.8), [0.8, 1)
B	[0, 0.4)	[0, 0.16) , [0.16, 0.28), [0.28, 0.32), [0.32, 0.4)
B	[0.16, 0.28)	[0.16, 0.208) , [0.208, 0.244), [0.244, 0.256), [0.256, 0.28)
	[0.208, 0.244)	

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Coding

Example 1

2. Read X_i

input message: A B B C #

3. Divid CurrentInterval into subintervals $IR_{CurrentInterval}$;

X_i	Current interval	Subintervals			
A	[0, 1)	[0, 0.4) ,	[0.4, 0.7),	[0.7, 0.8),	[0.8, 1)
B	[0, 0.4)	[0, 0.16) ,	[0.16, 0.28),	[0.28, 0.32),	[0.32, 0.4)
B	[0.16, 0.28)	[0.16, 0.208) ,	[0.208, 0.244),	[0.244, 0.256),	[0.256, 0.28)
C	[0.208, 0.244)	[0.208, 0.2224) ,	[0.2224, 0.2332),	[0.2332, 0.2368),	[0.2368, 0.244)

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Coding

Example 1

input message: A B B C #

No. 3

A	B	C	#
0.4	0.3	0.1	0.2

X_i	Current interval	Subintervals			
A	[0, 1)	[0, 0.4) ,	[0.4, 0.7),	[0.7, 0.8),	[0.8, 1)
B	[0, 0.4)	[0, 0.16) ,	[0.16, 0.28),	[0.28, 0.32),	[0.32, 0.4)
B	[0.16, 0.28)	[0.16, 0.208) ,	[0.208, 0.244),	[0.244, 0.256),	[0.256, 0.28)
C	[0.208, 0.244)	[0.208, 0.2224) ,	[0.2224, 0.2332),	[0.2332, 0.2368),	[0.2368, 0.244)
	[0.2332, 0.2368)				

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Coding

2. Read X_i

Example 1

input message: A B B C #

3. Divid CurrentInterval into subintervals $IR_{CurrentInterval}$;

X_i	Current interval	Subintervals			
A	[0, 1)	[0, 0.4) ,	[0.4, 0.7),	[0.7, 0.8),	[0.8, 1)
B	[0, 0.4)	[0, 0.16) ,	[0.16, 0.28),	[0.28, 0.32),	[0.32, 0.4)
B	[0.16, 0.28)	[0.16, 0.208) ,	[0.208, 0.244),	[0.244, 0.256),	[0.256, 0.28)
C	[0.208, 0.244)	[0.208, 0.2224) ,	[0.2224, 0.2332),	[0.2332, 0.2368),	[0.2368, 0.244)
#	[0.2332, 0.2368)	[0.2332, 0.23464) ,	[0.23464, 0.23572),	[0.23572, 0.23608),	[0.23608, 0.2368)

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Coding

2. Read X_i

Example 1

input message: A B B C #

No. 4

A	B	C	#
0.4	0.3	0.1	0.2

X_i	Current interval	Subintervals
A	[0, 1)	[0, 0.4) , [0.4, 0.7), [0.7, 0.8), [0.8, 1)
B	[0, 0.4)	[0, 0.16) , [0.16, 0.28), [0.28, 0.32), [0.32, 0.4)
B	[0.16, 0.28)	[0.16, 0.208) , [0.208, 0.244), [0.244, 0.256), [0.256, 0.28)
C	[0.208, 0.244)	[0.208, 0.2224) , [0.2224, 0.2332), [0.2332, 0.2368), [0.2368, 0.244)
#	[0.2332, 0.2368)	[0.2332, 0.23464) , [0.23464, 0.23572), [0.23572, 0.23608), [0.23608, 0.2368)
	[0.23608, 0.2368)	

A	B	C	#
0.4	0.3	0.1	0.2

Arithmetic Code

Coding

2. Read X_i

Example 1

input message: A B B C #

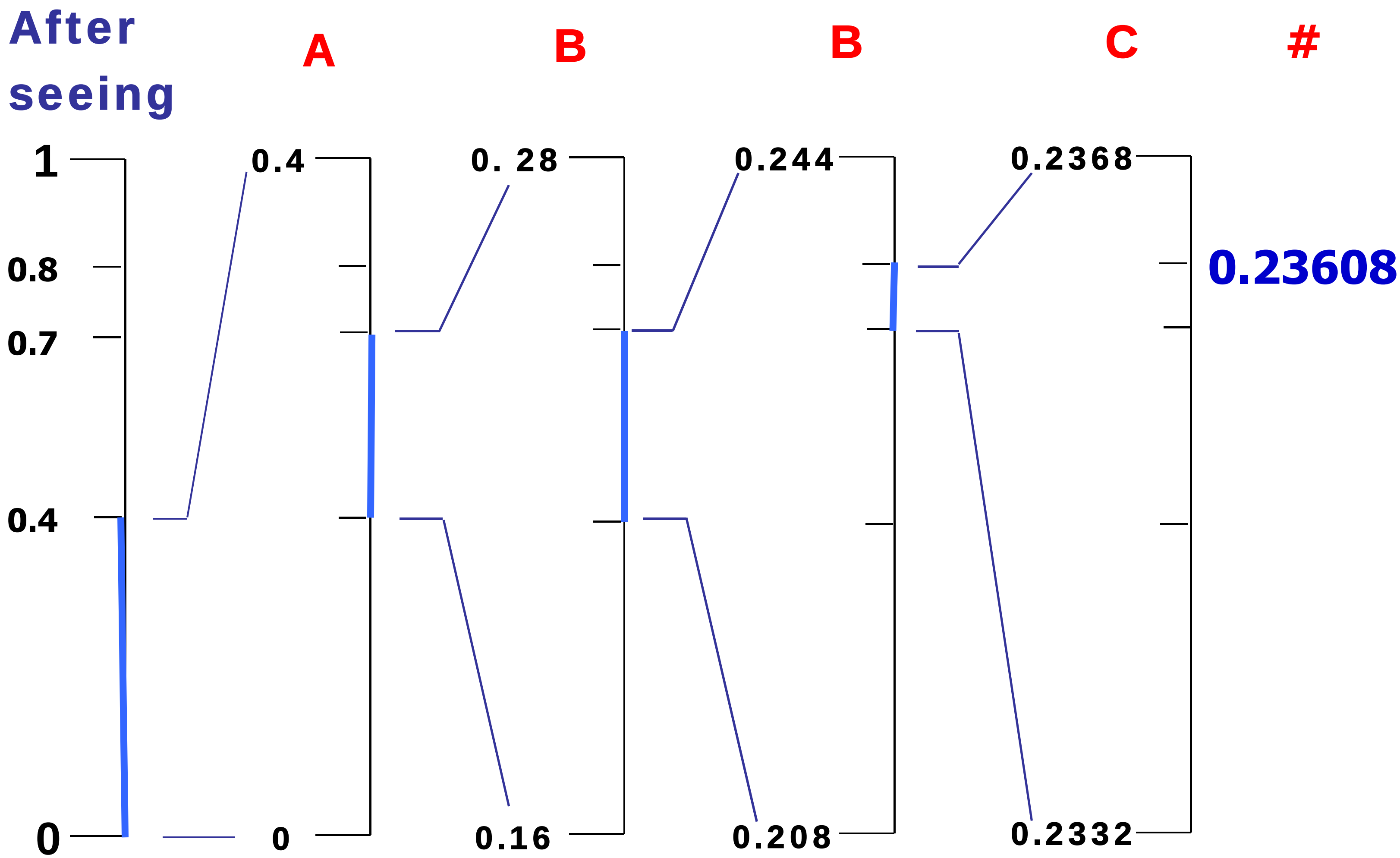
is the end of input message **Stop** Return current interval [0.23608, 0.2368)

X_i	Current interval	Subintervals			
A	[0, 1)	[0, 0.4) ,	[0.4, 0.7),	[0.7, 0.8),	[0.8, 1)
B	[0, 0.4)	[0, 0.16) ,	[0.16, 0.28),	[0.28, 0.32),	[0.32, 0.4)
B	[0.16, 0.28)	[0.16, 0.208) ,	[0.208, 0.244),	[0.244, 0.256),	[0.256, 0.28)
C	[0.208, 0.244)	[0.208, 0.2224) ,	[0.2224, 0.2332),	[0.2332, 0.2368),	[0.2368, 0.244)
#	[0.2332, 0.2368)	[0.2332, 0.23464) ,	[0.23464, 0.23572),	[0.23572, 0.23608),	[0.23608, 0.2368)
	[0.23608, 0.2368)				

Example 1

ABBC#

A	B	C	#
0.4	0.3	0.1	0.2



A	B	C	#
0.4	0.3	0.1	0.2

Arithmetic Code

Coding

Example 1

input message: A B B C #

 # is the end of input message **Stop** **Return current interval** **[0.23608, 0.2368)**

Return the lower bound of the currentinterval as the codeword of the input message

Input message	Codeword
ABBC#	0.23608

Example 1

A	B	C	#
0.4	0.3	0.1	0.2

- **The size of the final range is**

  0.2368 - 0.23608 = 0.00072,

  that is also exactly the multiplication of the probabilities of the five symbols in the message ABBC#

:

  (0.4)* (0.3)* (0.3)* (0.1)* (0.2) = 0.00072.

  it takes 5 decimal digits to encode the message.

- **According to Shannon :**

  The best compression code is the output length contains a contribution of –log(p) bits from the encoding of each symbol whose probability of occurrence is p.

  The entropy of ABBC# is :

  -log 0.4 -log 0.3 -log 0.3 -log 0.1 -log 0.2 =-log 0.00072 = 3.14

Decoding

Arithmetic decoding can be determined by the following algorithm:

ArithmeticDecoding (Codeword)

0. CurrentInterval = [0, 1);

While(1)

1. Divid CurrentInterval into subintervals $IR_{CurrentInterval}$;
2. Determine the $subinterval_i$ of CurrentInterval to which Codeword belongs;
3. Output letter x_i corresponding to this subinterval;
4. If x_i is the symbol '#'

   Return;
5. CurrentInterval = $subinterval_i$ in $IR_{CurrentInterval}$;

Arithmetic Code

Decoding

Example **Consider the information source**

Symbol	Probability
A	0.4
B	0.3
C	0.1
#	0.2

Then the input code word 0.23608 can be decoded to the message ABBC#

As we will see the explanation In the next slides

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Decoding

Example **input codeword: 0.23608**

0. CurrentInterval = [0, 1);

Current interval	Subintervals	Output
[0, 1)		

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Decoding

Example input codeword: 0.23608

1. Divid CurrentInterval into subintervals $IR_{CurrentInterval}$;

Current interval	Subinterv
[0, 1)	

IR[0,1)= {
[0, 0.4) , [0.4, 0.7),
[0.7, 0.8), [0.8, 1)
}

$$q_j = \sum_{i=1}^{j} p_i$$

$[L+(R-L)\, q_i,\ L+(R-L)\, q_{i+1})$

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Decoding

Example input codeword: 0.23608

Current interval	Subintervals	Output
[0, 1)	[0, 0.4) , [0.4, 0.7), [0.7, 0.8), [0.8, 1)	

$IR_{[0,1)} = \{$ [0, 0.4) , [0.4, 0.7), [0.7, 0.8), [0.8, 1) $\}$

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Decoding

Example **input codeword:** 0.23608

2. Determine the $subinterval_i$ of CurrentInterval to which Codeword belongs;

$0 \le 0.23608 < 0.4$

Current interval	Subintervals	Output
[0, 1)	[0, 0.4) , [0.4, 0.7), [0.7, 0.8), [0.8, 1)	

A	B	C	#
0.4	0.3	0.1	0.2

Arithmetic Code

Decoding

Example **input codeword: 0.23608**

2. Determine the subinterval$_i$ of CurrentInterval to which Codeword belongs;

$0 \leq 0.23608 < 0.4$

Current interval	Subintervals	Output
[0, 1)	[0, 0.4) , [0.4, 0.7), [0.7, 0.8), [0.8, 1)	

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Decoding

Example input codeword: 0.23608

3. Output letter x_i corresponding to this subinterval;

No. 1

A	B	C	#
0.4	0.3	0.1	0.2

No. 1

Current interval	Subintervals	Output
[0, 1)	[0, 0.4) , [0.4, 0.7), [0.7, 0.8), [0.8, 1)	A

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Decoding

Example input codeword: 0.23608

4. If x_i is the symbol '#'

Current interval	Subintervals	Output
[0, 1)	[0, 0.4) , [0.4, 0.7), [0.7, 0.8), [0.8, 1)	A

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Decoding

Example input codeword: 0.23608

4. If x_i is the symbol '#' **NO**

Current interval	Subintervals				Output
[0, 1)	[0, 0.4) ,	[0.4, 0.7),	[0.7, 0.8),	[0.8, 1)	A

A	B	C	#
0.4	0.3	0.1	0.2

Arithmetic Code

Decoding

Example input codeword: 0.23608

5. CurrentInterval = subinterval$_i$ in IR$_{CurrentInterval}$;

Current interval	Subintervals				Output
[0, 1)	[0, 0.4) ,	[0.4, 0.7),	[0.7, 0.8),	[0.8, 1)	A

A	B	C	#
0.4	0.3	0.1	0.2

Arithmetic Code

Decoding

Example **input codeword: 0.23608**

5. CurrentInterval = subinterval$_i$ in IR$_{CurrentInterval}$;

Current interval	Subintervals				Output
[0, 1)	[0, 0.4) ,	[0.4, 0.7),	[0.7, 0.8),	[0.8, 1)	A
[0, 0.4)					

Arithmetic Code

A	B	C	#
0.4	0.3	0.1	0.2

Decoding

Example input codeword: 0.23608

Similarly we repeat the algorithm steps 1 to 5 until the output symbol = '#'

Current interval	Subintervals				Output
[0, 1)	[0, 0.4) ,	[0.4, 0.7),	[0.7, 0.8),	[0.8, 1)	A
[0, 0.4)	[0, 0.16) ,	[0.16, 0.28),	[0.28, 0.32),	[0.32, 0.4)	B
[0.16, 0.28)	[0.16, 0.208) ,	[0.208, 0.244),	[0.244, 0.256),	[0.256, 0.28)	B
[0.208, 0.244)	[0.208, 0.2224) ,	[0.2224, 0.2332),	[0.2332, 0.2368),	[0.2368, 0.244)	C
[0.2332, 0.2368)	[0.2332, 0.23464) ,	[0.23464, 0.23572),	[0.23572, 0.23608),	[0.23608, 0.2368)	#

4. If x_i is the symbol '#' — **Yes** — **Stop**

Return the output message: A B B C #