Multimedia Networking ECE 599

Prof. Thinh Nguyen
School of Electrical Engineering and Computer Science

Based on B. Lee's lecture notes.

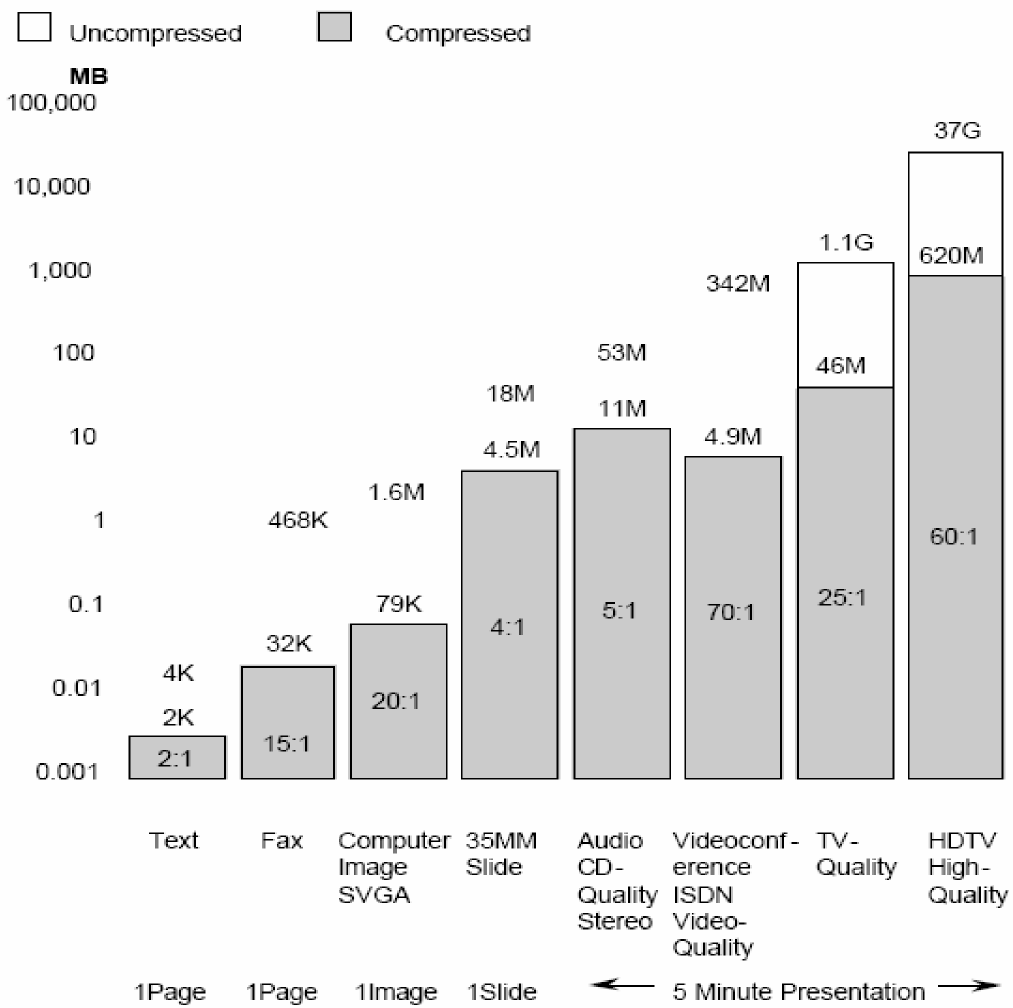
Outline

- Compression basics
- Entropy and information theory basics
- Lossless coding
  - Run-Length
  - Huffman
  - Arithmetic
- Lossy coding
  - DPCM

Compression basics

- Redundancy exists in many places
  - Texts
    - Redundancy(German) > Redundancy(English)
  - Video and images
    - Redundancy (videos) > redundancy(images)
  - Audio
    - Redundancy(music) ? Redundancy(speech)
- Eliminate redundancy – keep essential information
  - Assume 8 bits per character
  - Uncompressed: aaaaaaaaab: 10x8 = 80 bits
  - Compressed: 9ab = 3x8 = 24 bits
- Reduce the amount of bits to store the data
  - Small storage, small network bandwidth, low storage devices.
    - Ex: 620x560 pixels/frame
    - 24 bits/pixel → 1 MB
    - 30 fps → 30 MB/s (CD-ROM 2x 300KB/s)
    - 30 minutes → 50 GB

Compression basics



Compression methods

- JPEG (DCT), JPEG-2000 (Wavelet)
  - Images
- JBIG
  - Fax
- LZ (gzip)
  - Text
- MPEG
  - Video

16:1 compression ratio

Compression Classification

Compression

Lossless

- Decoded data = original data
- Comp. ratio < lossy comp. ratio
- Eliminate redundancy
- Used where errors are not allowed, e.g, computer programs.
- LZ, JBIG

Lossy

- Decoded data ~ original data
- Comp. ratio > lossless comp. ratio
- Keeping important information
- Used where small errors are allowed, e.g, images, videos.
- JPEG, MPEG

Information theory basics

Amount of information within data is defined as the number of bits to represent different patterns in the data (Hartley 1928).

A message of l symbols and every symbol has a choice of n possibilities.

Number of possible patterns is n^l

Number of bits to present different patterns is $l \log\{n\}$

Information theory basics

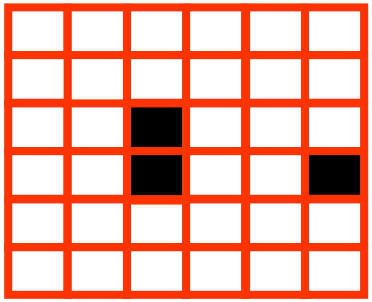
- Shannon's measure of information is the number of bits to represent the amount of uncertainty (randomness) in a data source, and is defined as entropy

$$H = -\sum_{i=1}^{n} p_i \log(p_i)$$

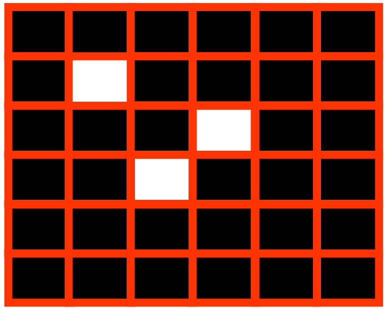
Where there are n symbols 1, 2, ... n, each with probability of occurrence of p_i

Examples

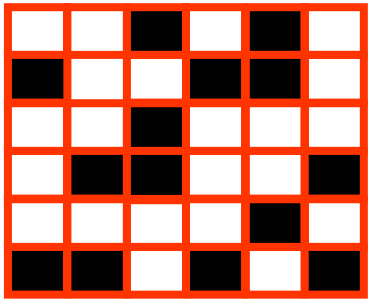
An binary image can be considered a random source of two symbols “black” and “white”.



P(black) < p(white)



P(black) > p(white)



More uncertainty = more information

Entropy Intuition

Why $$H = -\sum_{i=1}^{n} p_i \log(p_i)$$

Suppose you have a long random string of two binary symbols 0 and 1, and the probability of symbols 1 and 0 are p_0 and p_1

Ex: 00100100101101001100001000100110001 ....

If any string is long enough say N, it is likely to contain Np_0 0's and Np_1 1's. The probability of this string equal to

$$p = p_0^{Np_0} p_1^{Np_1}$$

Hence, # of possible patterns is $1/p = p_0^{-Np_0} p_1^{-Np_1}$

bits to represent all possible patterns is $\log(p_0^{-Np_0} p_1^{-Np_1}) = -\sum_{i=0}^{1} Np_i \log p_i$

The average # of bits to represent the symbol is therefore

$$-\sum_{i=0}^{1} p_i \log p_i$$

Entropy and compression

- Entropy is the minimum number of bits per symbols to completely represent a random data source.

- A compression scheme is optimal if its average number of bits/symbol equals to the entropy.

- Example: in an 8-bit image with uniform distribution of pixel values, i.e. $p_i = 1/256$, the entropy of this image is 8 bits.

$$-\sum_{i=0}^{255} p_i \log p_i = -\sum_{i=0}^{255} \frac{1}{256} \log \frac{1}{256} = 8$$

Entropy and compression

- We are interested in finding a map from the symbols to the code such that the average bits/symbols equals to the entropy.
- Clearly, this depends on the distribution, i.e. the probability of occurrences of each symbol.
- Intuitively, we want to use fewer bits to represent the frequent symbols, and longer bits to represent rare symbols in order to minimize the average bits/symbols.
- Example: suppose the source contains 3 symbols a, b, and c with p(a) = .8, p(b) = .15 and p(c) = .05.

a = 0

b = 10 average bits/symbols = .8 + .15*2 + .05*2

c = 11

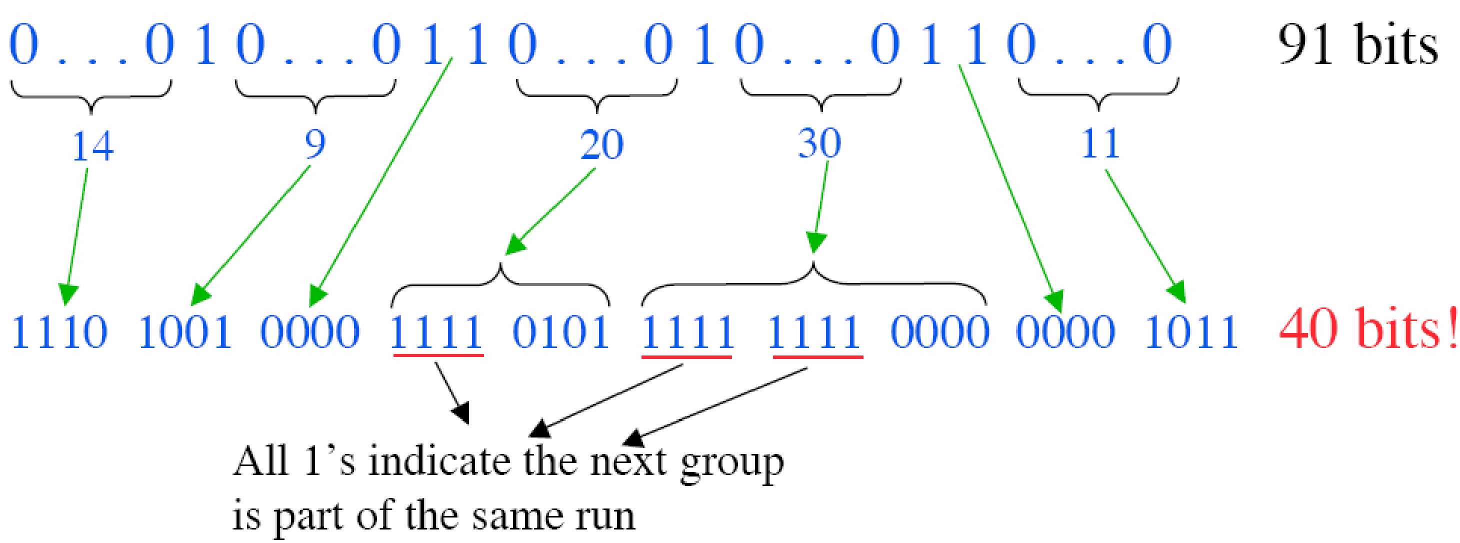
Lossless coding: Run-Length encoding (RLE)

- Redundancy is removed by not transmitting consecutive pixels or character values that are equal.
- The repeated value can be coded once, along with the number of times it repeats.
- Useful for coding black and white images e.g. fax.

Binary RLE

- Code the run length so 0's using k bits. Transmit the code.
- Do not transmit runs of 1's.
- Two consecutive 1's are implicitly separately by a zero-length run of zero.

Example: suppose we use 4 bits to encode the run length (maximum run length of 15) for following bit patterns



RLE Performance

- Worst case behavior: transition occurs on each bit. Since we use k bits to represent the transition, we wasted k-1 bits.

- Best case behavior: no transition and use k bits to represent run length then the compression ratio is (2^k-1)/k.

Lossless coding: Huffman Coding

- Input values are mapped to output values of varying length, called *variable-length code (VLC).*

- Most probable input values are coded with fewer bits.

- No code word can be prefix of another code word.

Huffman Code Construction

- Initialization: put all nodes (values, characters, or symbols) in an sorted list.

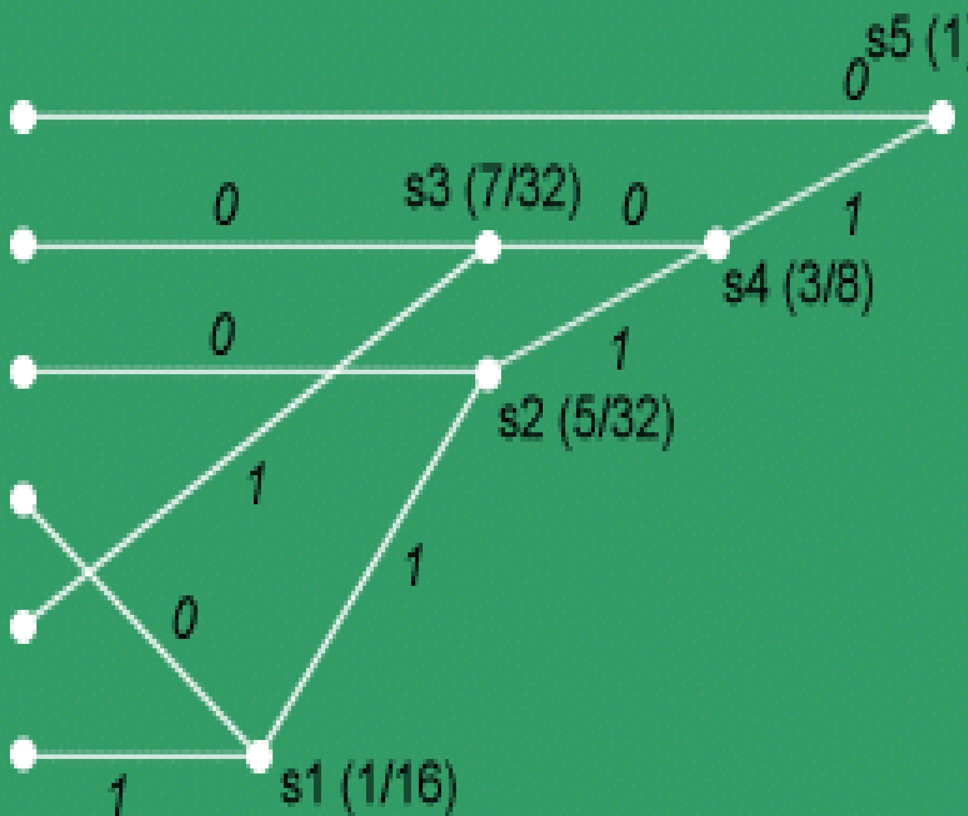
- Repeat until the list has only one node:

  - Combine the two nodes having the lowest frequency (probability). Create a parent node assigning the sum of the children's probability and insert it in the list.

  - Assign code 0, 1 to the two branches and delete children from the list.

Huffman example

Symbol	Codeword	Probability
a	0	5/8
b	100	3/32
c	110	3/32
d	1110	1/32
e	101	1/8
f	1111	1/32



Huffman Efficiency

- Entropy H of the example above

H = (5/8)log(5/8) + 2(3/32)log(3/32)+ 2(1/32)log(1/32) + (1/8)log(1/8) = 1.75 bits

- Huffman code:

5/8 + (3/32) 3 + 3 (3/32) + 4 (1/32)+3 (1/8) + 4 (1/32) = 1.81 bits

- Fixed length code: 3 bits.

- Huffman code is much better than fixed length code.

Huffman coding

- Optimal if symbols are coded one at a time!

A A B D C D A E ....

00 00 011

$$H(x) \leq R(x) \leq H(x) + 1$$

Arithmetic Coding

- Huffman coding method is optimal when and only when the symbol probabilities are integral powers of ½, which rarely is the case.

- In arithmetic coding, each symbol is coded by considering prior data

  - Relies on the fact that coding efficiency can be improved when symbols are combined.

  - Yields a single code word for each string of characters.

  - Each symbol is a portion of a real number between 0 and 1.

Arithmetic vs. Huffman

- Arithmetic encoding does not encode each symbol separately; Huffman encoding does.
- Arithmetic encoding transmits only length of encoded string; Huffman encoding transmits the Huffman table.
- Compression ratios of both are similar when the entropy is high. When entropy is low, arithmetic coding outperforms Huffman coding.

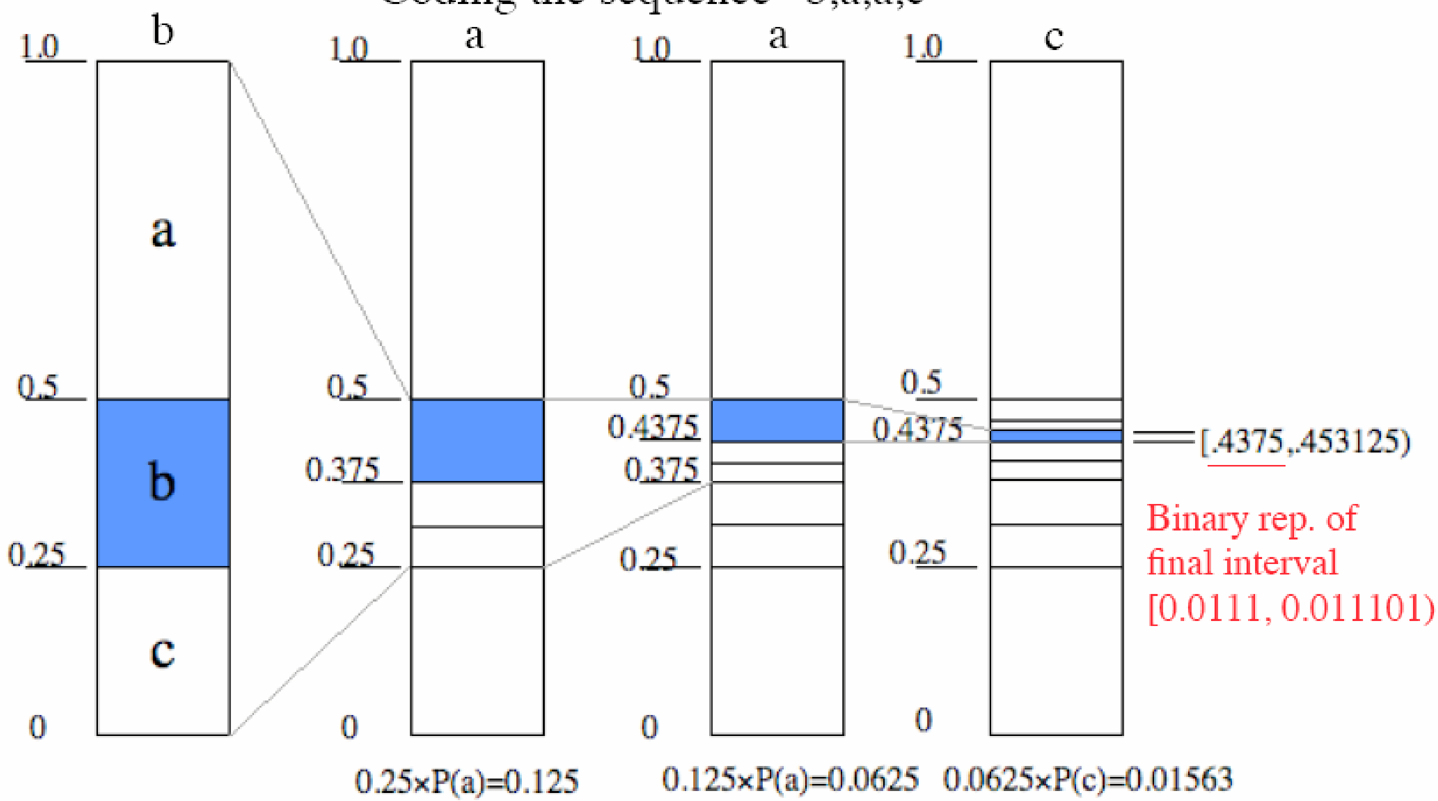
Arithmetic Coding Algorithm

```
BEGIN
    low = 0.0;   high = 1.0;      range = 1.0;
    while (symbol != terminator){
        low = low + range * Range_low(symbol);
        high = low + range * Range_high(symbol);
        range = high - low;
    }
    output a code so that low <= code < high;
END
```

Arithmetic Coding Example

Example: P(a) = 0.5, P(b) = 0.25, P(c) = 0.25

Coding the sequence "b,a,a,c"



Arithmetic Coding Example

- Size of the final subinterval, i.e., range s, is 0.4375 - 0.453125 = 0.015625, which is also determined by the product of the probabilities of the source message, P(b)×P(a)×P(a)×P(c).
- The number of bits needed to specify the final subinterval is given by at most $\lceil -\log_2 s \rceil = 6$ bits.
- Choose the smallest binary fraction that lies within [0.0111, 0.011101).
  - Code is 0111 (from 0.0111)!
- Entropy H is .5(1) + .25(2) + .25(2) = 1.5 bits/symbol
- 4 symbols encoded with 6 bits gives 1.5 bits/symbol!
- Fixed length code would have required 2 bits/symbol.

Code Generator for Encoder

```
BEGIN
    code = 0;
    k = 1;
    while (value(code) < low) {
            assign 1 to the kth binary fraction bit;
            if (value(code) > high)
                replace the kth bit by 0;
            k = k + 1;
    }
END
```

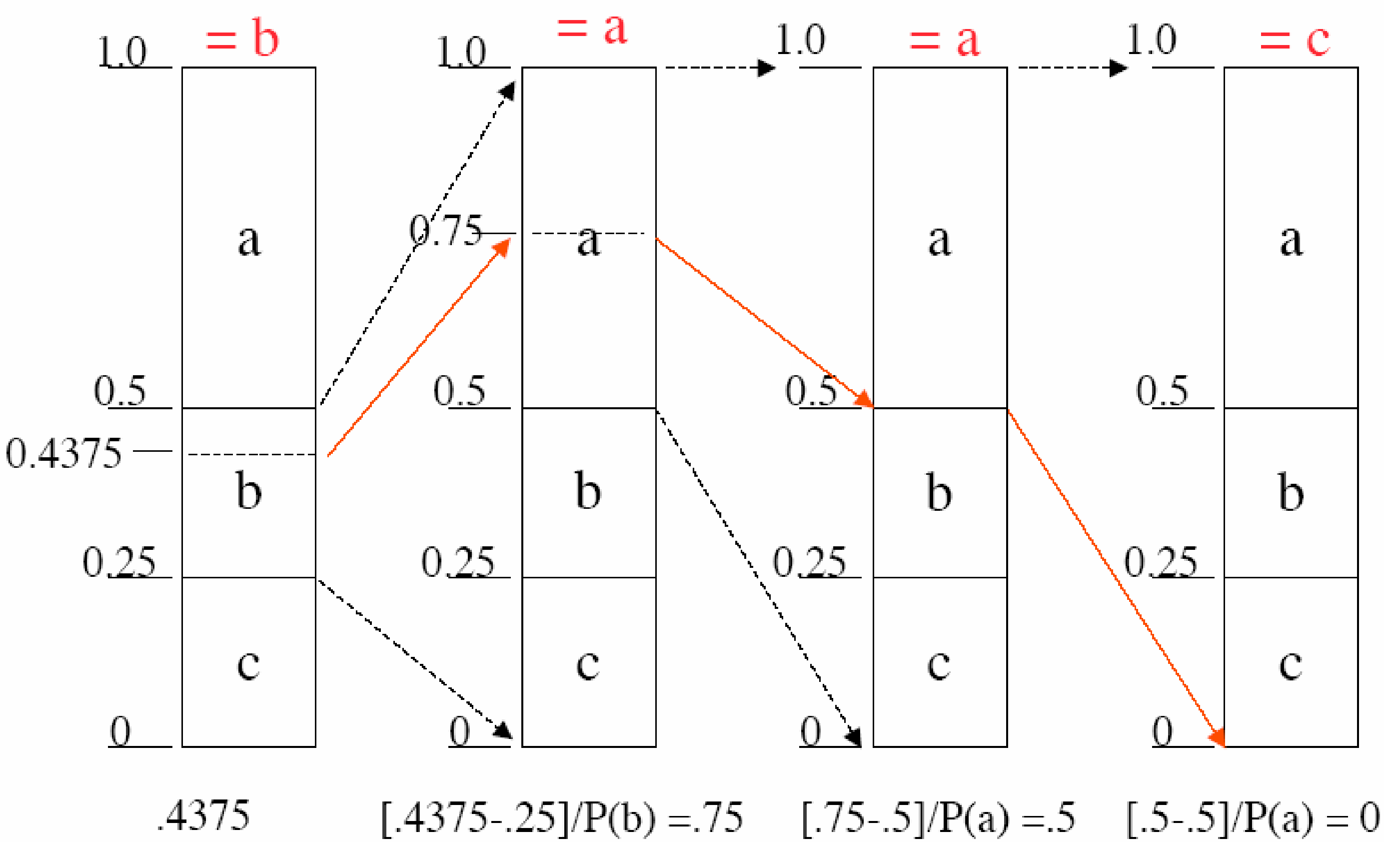
Code Generator Example

- For the previous example low = 0.4375 and high = 0.453125.
  - k = 1: 0.1 (0.5) > high & 0.0 < low => continue
  - k = 2: 0.01 (0.25) < high & 0.01 < low => continue
  - k = 3: 0.011 (0.375) < high & 0.011 < low => continue
  - k = 4: 0.0111 (0.4375) < high & 0.0111 = low => terminate
- Output code is 0111 (from 0.0111)

Decoding

```
BEGIN
    get binary code and convert to decimal value = value(code);
    do {
            find a symbol so that
                    Range_low(symbol) <= value < Range_high(symbol)
            output symbol;
            low = Range_low(symbol);
            high = Range_high(symbol);
            range = high - low;
            value = [value - low] / range
    }
    until symbol is a terminator
END
```

Decoding Example



Arithmetic Coding Issues

- Need to know when to stop, e.g., in the previous example, 0 can be c, cc, ccc, etc. Thus, a special character is included to signal end of message.
- Also, precision can grow without bound as length of message grows. Thus, fixed precision registers are used with detection and management of overflow and underflow.
- Thus, both use of message terminator and fixed-length registers reduces the degree of compression.

Arithmetic Coding Intuition

Sequence with larger probability is represented with larger range s, hence smaller number of bits.

Huffman Coding, Run-Lengh Coding, and Arithmetic Coding Question

A A A A A B A A A A A A A A A A A A A B B C C A

A A B A A B A B C A B A C B A C B C A B C C C A

Is there such thing as too much compression?

- What if an error happens in the encoded bit stream, e.g. a bit flips or lost packet?
  - What happens to the decoded data? Is it all corrupted.
  - How extensive is the damage to the data?

- Little redundancy can be a good thing! (for transmission over lossy channel)