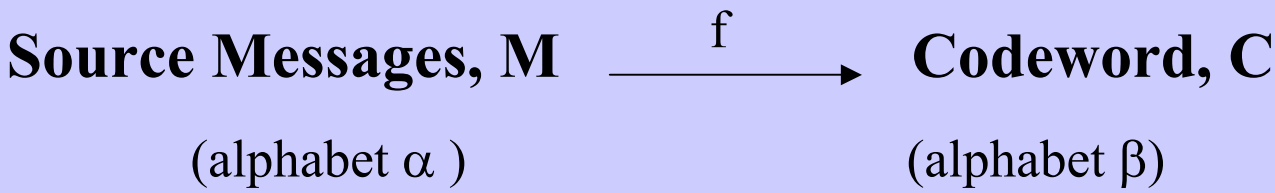
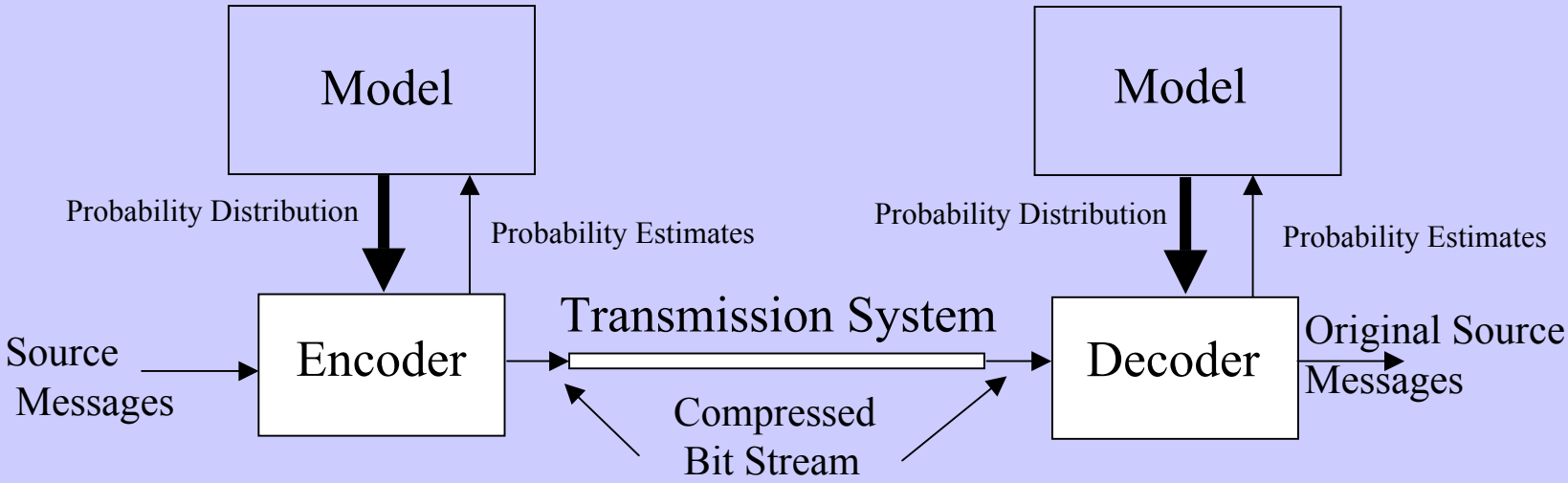
Coding



Properties

- **Distinct**
- **Uniquely Decipherable (Prefix)**
- **Instantaneously Decodable**
- **Minimal Prefix**

Modeling and Coding



- Model predicts next symbol
- Probability distribution and static codes
- Probability estimates and dynamic codes

Entropy as a Measure of Information

- Given a set of possible events with known probabilities $p_1, p_2, \ldots, p_n$, that sum to 1.
- *Entropy* $E(p_1, p_2, \ldots, p_n)$ (Shannon, 1940's): how much choice in selecting an event.
  - E should be a continuous function of p_i.
  - If $p_i=p_j$ for all $1\leq i,j\leq n$, then E should be an increasing function of n.
  - If choice is made in stages, E should be the weighted sum of the entropies at each stage (weights are the probabilities of each stage).

Entropy

- Shannon showed that only one function can satisfy these conditions.
  - *Self-information* of event A with probability P(A) is i(A) = - log P(A)
  - Entropy of a source is the sum of the self-information over all events

$$E(p_1, p_2, \ldots p_n) = -k \sum_{i=1}^{n} p_i \log p_i$$

Information and Compression

- Compression seeks a message representation that uses exactly as many bits as required for the information content (entropy is a lower bound on compression).
- However, computing entropy is difficult.
- Example: 1 2 1 2 3 3 3 3 1 2 3 3 3 3 1 2 3 3 1 2
  - One char at a time: P(1)=P(2)=¼, P(3)=½; entropy is 1.5 bits/symbol.
  - Two chars at a time: P(1 2)=P(3 3)=½; entropy is 1 bit/symbol.

Models Improve Entropy Computations

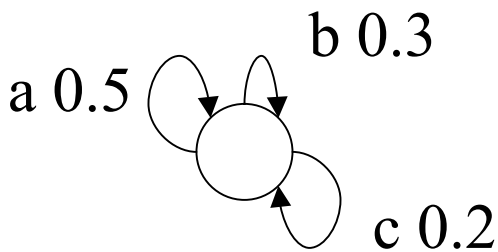
- Finite Context Models
- Finite State Models (Markov models)
- Grammar Models
- Ergodic Models

Finite Context Models

- Order k model: k preceding characters used as context in determining probability of next character.
- Examples:
  - Order -1 model: all characters have equal probability.
  - Order 0 model: probabilities do not depend on context.

Finite State Models (Markov Models)

- Probabilistic finite state machine.
- Fixed context models are a subclass.

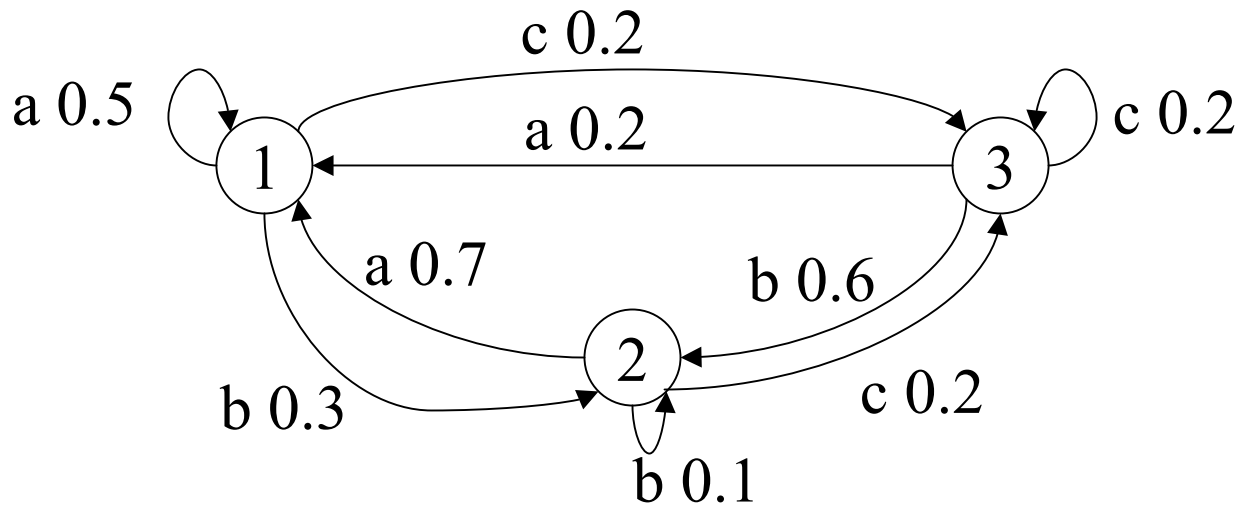


a	b	c	a	a	b
0.5	0.3	0.2	0.5	0.5	0.3

Msg. Prob. = 0.00225
(8.80 bits entropy)

Order 0 Fixed Context Model as a Finite State Model

Order 1 Fixed Context Model as a Finite State Model



Message:	a	b	c	a	a	b
States:	1 1	2	3	1	1	2
Probabilities:	0.5	0.3	0.2	0.2	0.5	0.3

Msg. Prob. = 0.0009; entropy = 10.1 bits

Grammar Models

- Use a grammar as the underlying structure.
  - Grammars have more expressive power than finite state machines.
  - Assign probabilities to each production rule.
  - However, does not appear useful to exploit this for natural language.

Example:

1. message --> string "`.`" (probability 1)
2. string --> substring string (probability 0.6)
3. string --> empty-string (probability 0.4)
4. substring --> "`a`" (probability 0.5)
5. substring --> "`(`" string "`)`" (probability 0.5)

String `((a)(a)).` has entropy 7.80 bits.

Ergodic Models

- *Ergodic*: as a sequence grows in length, it better represents the entire model.
- Usually assume this for natural language models.

Entropy and Redundancy

Entropy of Source H = $\sum_{i=1}^{n}[-p_i \log p_i]$

=Average number of bits per symbol for an optimal encoding scheme for an alphabet of size n, i-th symbol having a probability of p_i.

Entropy of a message of m symbols '$a_1 a_2 \ldots a_m$' = $\sum_{i}^{m} [-p_i \log p_i]$

=Total number of bits in a message

$\geqq$ mH = length of message∗entropy of source

Redundancy= $\sum_{i}^{m} l_i p_i$ - H

where l_i is the length of the code for the source symbol a_i.

Entropy and Encoding

- Measure of uncertainty/choice/information in a symbol sequence, with respect to a given model.
- Shannon's "Noiseless Source Coding Theorem": Entropy is optimal lower bound on average length of compressed message.
- Optimal symbol length for character encoding: $E_i = -\log_2 p_i$ bits.
- **Minimum redundancy code** has minimum average code length for a given probability distribution.

Example

Message = aa_bbb_cccc_ddddd_eeeeee_fffffffgggggggg

Symbol	Probability	Huffman Code
a	2/40	1001
b	3/40	1000
c	4/40	011
d	5/40	010
e	6/40	111
f	7/40	110
g	8/40	00
space	5/40	101

H= 2.894 bits/symbol

mH= 116 bits

Message Length = 117

Redundancy= 117/40 - 2.894

= 2.925- 2.894

= 0.031 bits/symbol

$$\textbf{Compression Ratio} = \frac{\text{average symbol length in bits}}{\text{average codeword length in bits}}$$

Using 6-bit ASCII codes for source symbols, this ratio is 6/2.89=2.076 or 48.16% compression (the compressed file is about 48.16% of the original file size.)

Shannon-Fano Code

- List the source symbols in non-increasing probability order.
- Divide the list in two lists of nearly equal total probability.
- Assign a bit 0 to the first list and a bit 1 to the second list.
- Recurse the process over the two lists until each list has only one symbol.

Symbol	Probability	Code
a	1/2	0
b	1/4	10
c	1/8	110
d	1/16	1110
e	1/32	11110
f	1/32	11111

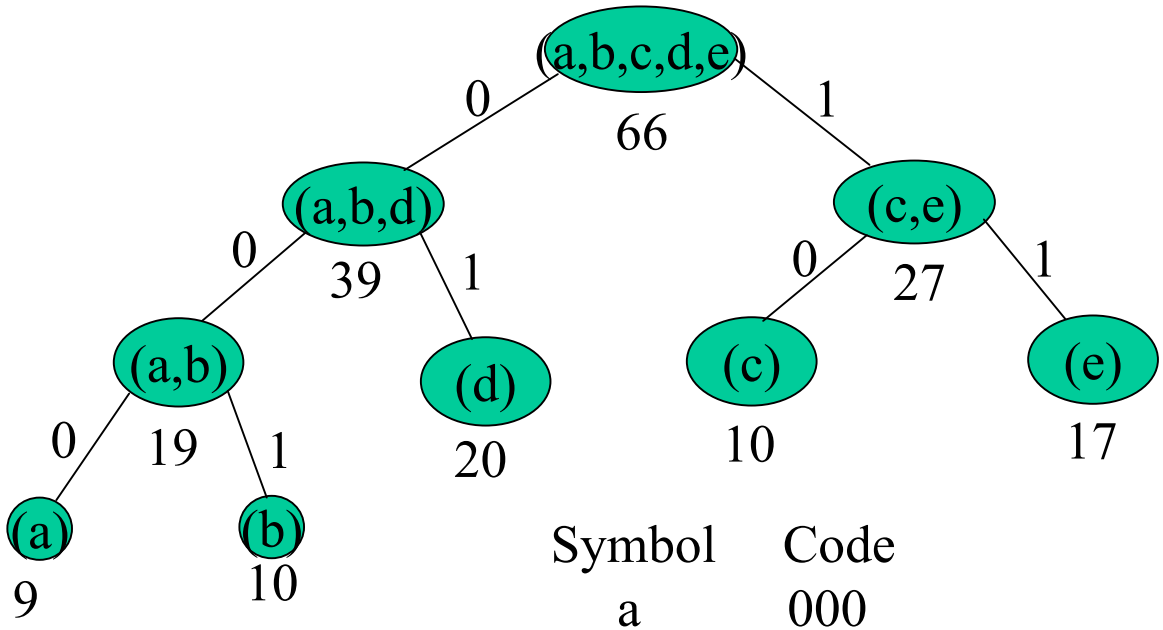
Huffman Code

1. Construct a Binary Tree of Sets of Source Symbols.

- Sort the set of symbols with non-decreasing probabilities.
- Form a set including two symbols of smallest probabilities.
- Replace these by a single set containing both the symbols whose probability is the sum of the two component sets.
- Repeat the above steps until the set contains all the symbols.
- Construct a binary tree whose nodes represent the sets. The leaf nodes representing the source symbols.

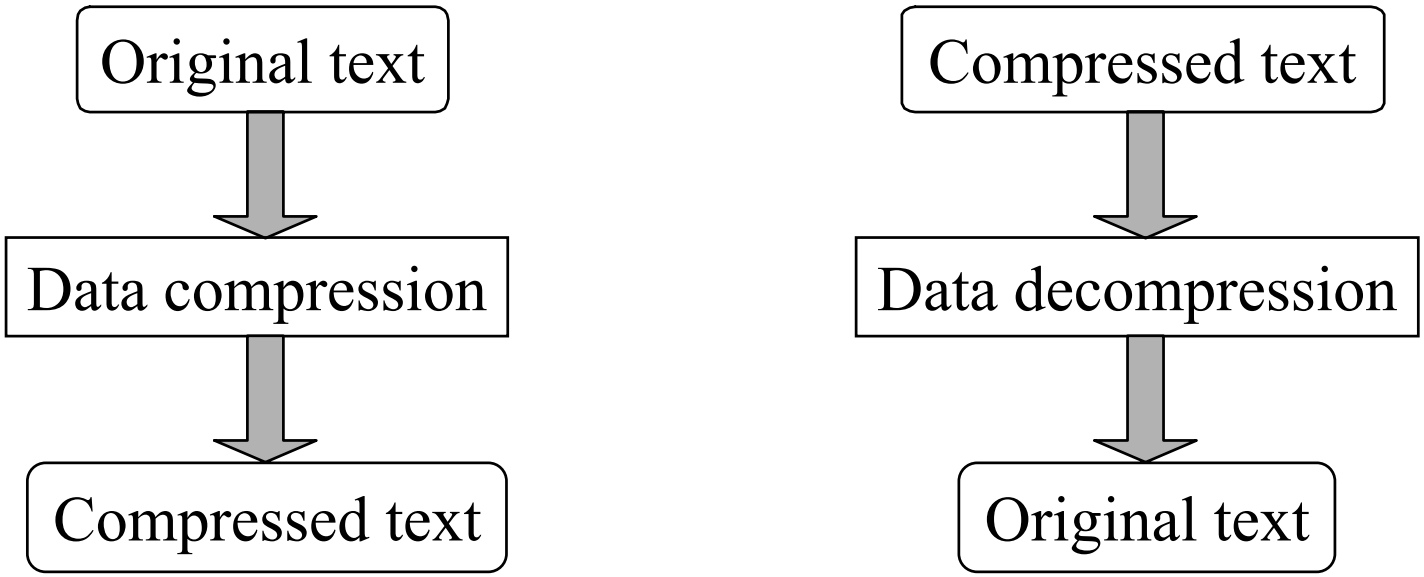
2. Traverse each path of the tree from root to a symbol, assigning a code 0 to a left branch and 1 to a right branch. The sequence of 0's and 1's thus generated is the code for the symbol.

Example



Symbol	Code
a	000
b	001
c	10
d	01
e	11

Text compression



Decoding

Decoding is performed by reading the code word left to right. The tree is traversed beginning from the root of the tree, moving left or right at each node corresponding to code bit being 0 or 1, respectively. Once it reaches the leaf node, a symbol of the source code is decoded and the traversal path starts again from the root of the tree .

Thus, if the input bit is '000011111110101111001', the decoded message is 'ggee_ca', assuming the decoder also has the same Huffman tree as the encoder has. In fact, the same probability distribution is used by both the encoder and decoder based on the distribution of symbol probabilities for a given text corpus. Note, the Huffman code is a prefix code which makes it possible to decode messages in only one pass over the codeword.

Properties of Huffman Code

- Huffman codes are minimum redundancy codes for a given probability distribution of the message.
- Huffman code is not ‘optimal’ unless the probability distribution of all possible n-grams are used to build the tree, which is unrealistic.
- Huffman code does not achieve ‘minimum redundancy’ because it does not allow fractional bits.

Models

- *Static*: probabilities are determined using some representative text corpus.
- *Semi-static*: two-pass, one to gather statistics and the second to encode data. The model must also be transmitted with compressed text; appropriate for fixed text.
- *Adaptive*: probabilities are adjusted based on already processed input symbols.

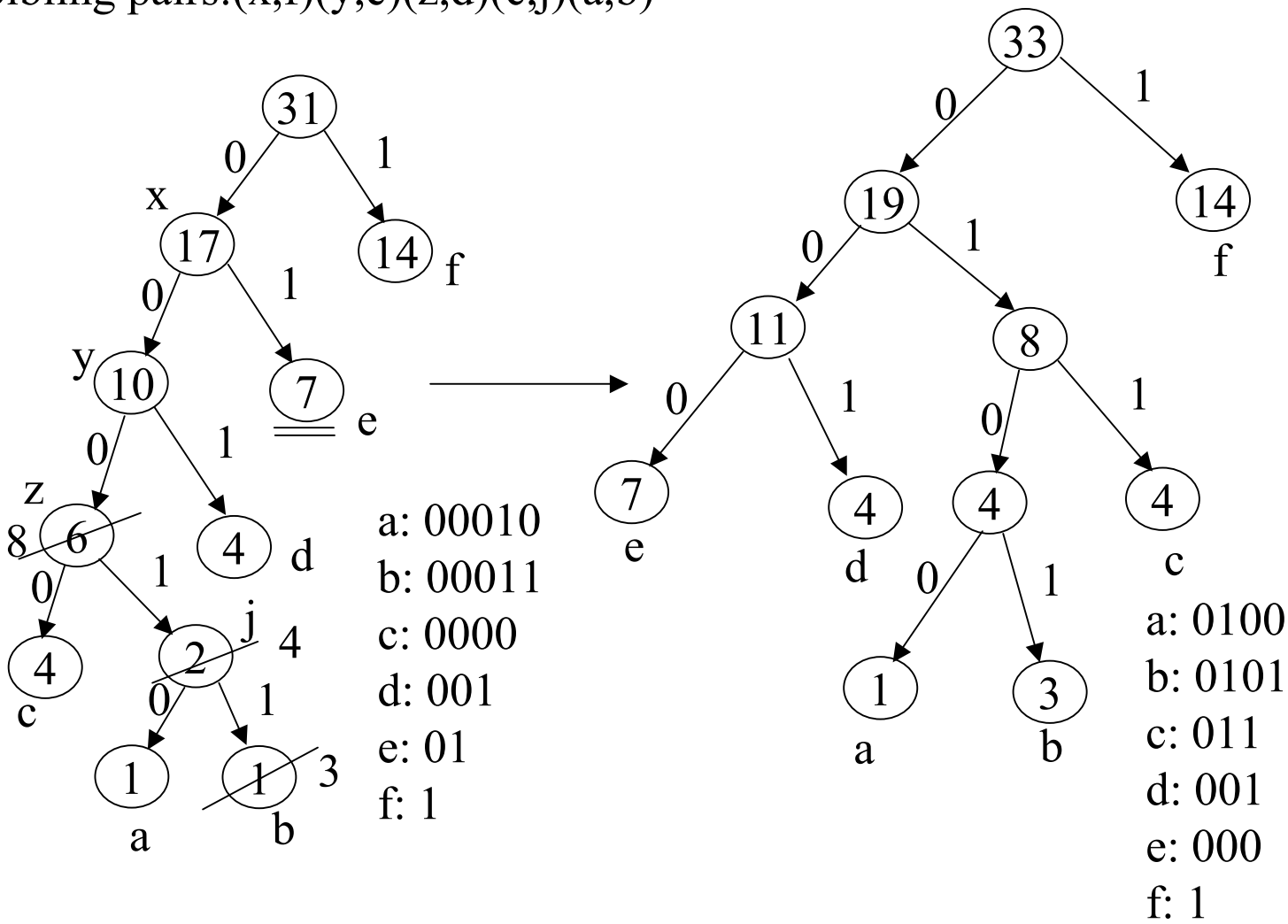
Adaptive Huffman Code

Huffman tree has the following properties

- ***Each node except root node has a sibling.***
- ***If the nodes (excluding root) are listed in order of non-increasing weight,then each node is adjacent to its sibling.***

Procedure: Whenever the count of a node is incremented, the new count is compared with the two counts of the next higher sibling pair (if any) in the ordered list. If the new count becomes larger than any one of these two counts, the two nodes must be interchanged (or the two subtrees must be interchanged) and the new counts computed until no further interchange can take place.

Sibling pairs:(x,f)(y,e)(z,d)(c,j)(a,b)

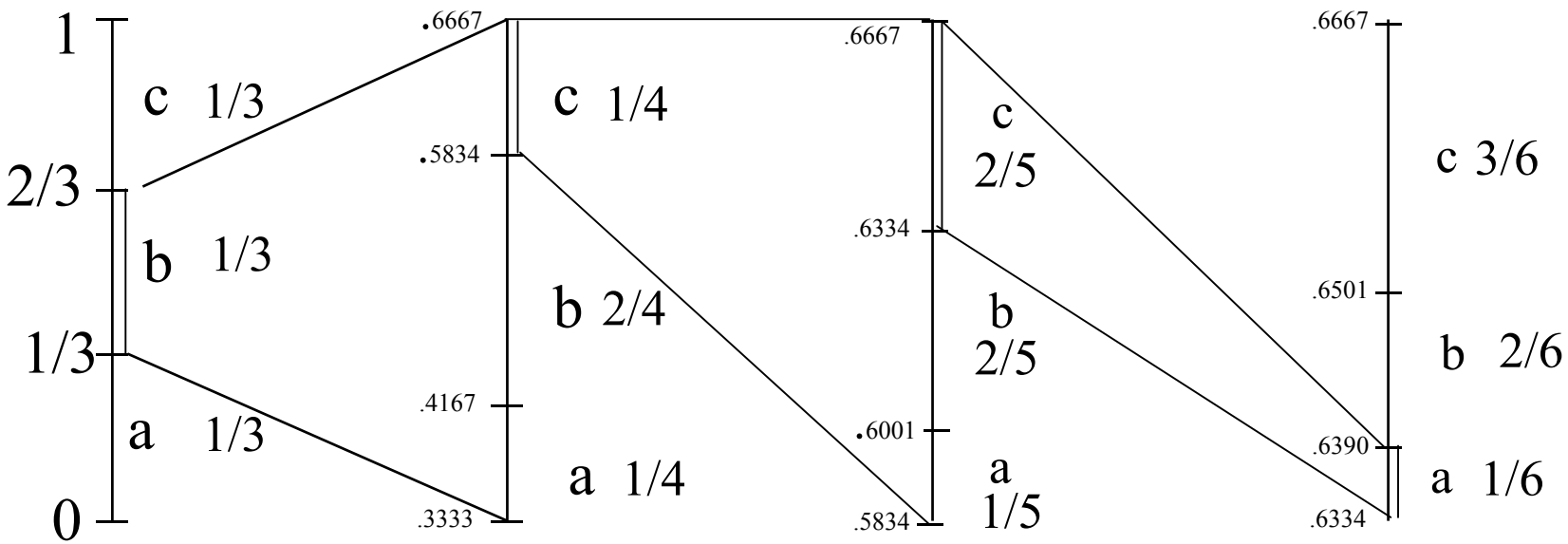


Entropy Coding

- Huffman coding: Create binary (Huffman) tree such that path lengths correspond to symbol probabilities. Use path labels as encodings.
- Arithmetic coding: Combine probabilities of subsequent symbols into a single fixed-point of high precision. Encode that number in binary.

Arithmetic Coding

Consider an half open interval [*low*,*high*). Initially, interval is set as [0,1) and *range*= *high -low* = 1-0 = 1.
Interval is divided into cumulative probabilities of n symbols, each having the same probability 1/n at the beginning.



Any value in the range [.6334,.6390) encodes ‘bcca’

Encoding Algorithm

Assume the source symbols have been numbered 1 through n and the probability of i -th symbol is p_i. The k-th symbol is encoded by a number between [*low,high*) as follows:

- ***Calculate the probabilities dynamically with the k-th symbol***
- ***low_bound=Cumulative(k-1)*** $\Leftarrow \sum p_i$
- ***high_bound=Cumulative(k)*** $\Leftarrow \sum p_i$
- ***range*** $\Leftarrow$ ***high - low (from the previous iteration or initial values[which is 1-0=1])***
- ***high*** $\Leftarrow$ ***low + range * high_bound***
- ***low*** $\Leftarrow$ ***low + range * low_bound***

Decoding Algorithm

Assume the initial probability of each symbol is 1/n.

1. **Calculate the vector (c_1,c_2,…,c_n) where c_i corresponds to cumulative probabilities up to and including the i-symbol within the *high/low* range (this is initialized to be 1).**
2. **Given the received value v, find c_k and c_{k+1} such that $c_k < v \leq c_{k+1}$.**
3. **Reset *low* to be c_k and *high* to be c_{k+1}.**
4. **Output symbol k and calculate the new probability distribution of all symbols.**
5. **Repeat the process until the numbers produced are within the bounds of the arithmetic precision agreed by both the encoder and decoder jointly.**

Properties of Arithmetic Coding

- The dynamic version is not more complex than the static version.

- The algorithm allocates $-\log p_i$ number of bits to a symbol of probability p_i whether or not this value is low or high. Unlike Huffman codes which is a fixed-to-variable coding scheme, arithmetic coding is variable -to-fixed coding scheme, and is capable of allocating non-integral number of bits to symbols, producing a near-optimal coding. It is not absolutely optimal due to limited precision of arithmetic operations.

- Incremental transmission of bits are possible, avoiding working with higher and higher precision numbers.