15-750:Algorithms in the Real World

Data Compression

PROBABILITY CODING

Assumptions and Definitions

Communication (or a file) is broken up into pieces called **messages**.

Each message come from a **message set** $S = \{s_1, \ldots, s_n\}$ with a **probability distribution** $p(s)$.

Code $C(s)$: A mapping from a message set to **codewords**, each of which is a string of bits

Message sequence: a sequence of messages

Variable length codes and Unique Decodability

A **variable length code** assigns a bit string (codeword) of variable length to every message value

e.g. `a = 1, b = 01, c = 101, d = 011`

What if you get the sequence of bits `1011` ?

Is it `aba, ca,` or, `ad`?

A **uniquely decodable code** is a variable length code in which bit strings can always be uniquely decomposed into its codewords.

Prefix Codes

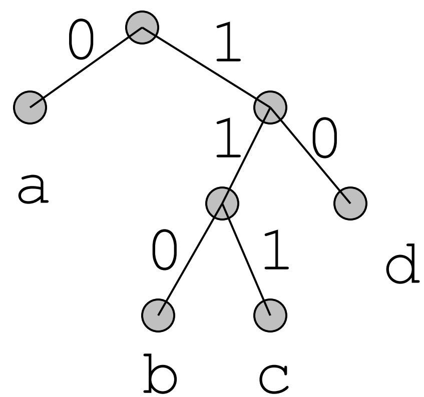
A **prefix code** is a variable length code in which no codeword is a prefix of another word.

e.g., a = 0, b = 110, c = 111, d = 10

All prefix codes are uniquely decodable

Prefix Codes: as a tree

Prefix codes can be viewed as a binary tree with 0s or 1s on the edges and message values at the leaves:



a = 0, b = 110, c = 111, d = 10

Average Length

For a code *C* with associated probabilities *p(c)* the **average length** is defined as

$$l_a(C) = \sum_{c \in C} p(c) l(c)$$

$l(c)$ = length of the codeword *c* (a positive integer)

We say that a prefix code *C* is **optimal** if for all prefix codes *C'*, $l_a(C) \leq l_a(C')$

Relationship to Entropy

Theorem (lower bound): For any probability distribution p(S) with associated uniquely decodable code C,

$$H(S) \leq l_a(C)$$

Theorem (upper bound): For any probability distribution p(S) with associated <u>optimal</u> prefix code C,

$$l_a(C) \leq H(S) + 1$$

Kraft McMillan Inequality

Theorem (Kraft-McMillan): For any uniquely decodable code C,

$$\sum_{c \in C} 2^{-l(c)} \leq 1$$

Conversely, for any set of lengths L such that $\sum_{l \in L} 2^{-l} \leq 1$

there is a prefix code C such that

$$l(c_i) = l_i (i = 1, \ldots, |L|)$$

We will use Kraft McMillan for proving the upper bound theorem.

Proof of the Upper Bound (Part 1)

Assign each message a length: $l(s) = \lceil \log(1/p(s)) \rceil$

We then have

$$\begin{aligned}
\sum_{s \in S} 2^{-l(s)} &= \sum_{s \in S} 2^{-\lceil \log(1/p(s)) \rceil} \\
&\leq \sum_{s \in S} 2^{-\log(1/p(s))} \\
&= \sum_{s \in S} p(s) \\
&= 1
\end{aligned}$$

Then, by the converse part of Kraft-McMillan inequality there is a prefix code with lengths *l(s)*.

Proof of the Upper Bound (Part 2)

Now we can calculate the average length given l(s)

$$
\begin{aligned}
l_a(S) &= \sum_{s \in S} p(s) l(s) \\
&= \sum_{s \in S} p(s) \cdot \left\lceil \log\left(1/p(s)\right) \right\rceil \\
&\leq \sum_{s \in S} p(s) \cdot (1 + \log(1/p(s))) \\
&= 1 + \sum_{s \in S} p(s) \log(1/p(s)) \\
&= 1 + H(S)
\end{aligned}
$$

Another property of optimal codes

Theorem: If C is an optimal prefix code for the probabilities $\{p_1, \ldots, p_n\}$, then $p_i > p_j$ implies $l(c_i) \leq l(c_j)$

Proof: (by contradiction: **switching technique**)

Assume $l(c_i) > l(c_j)$ (for the sake of contradiction).

Consider switching codes c_i and c_j.

If l_a is the average length of the original code, the length of the new code is

$$
\begin{aligned}
l_a' &= l_a + p_j(l(c_i) - l(c_j)) + p_i(l(c_j) - l(c_i)) \\
&= l_a + (p_j - p_i)(l(c_i) - l(c_j)) \\
&< l_a
\end{aligned}
$$

This is a contradiction since l_a is not optimal

Huffman Codes

Invented by Huffman as a class assignment in 1950.

Used in many, if not most, compression algorithms

Properties:

- Generates optimal prefix codes
- Cheap to generate codes
- Cheap to encode and decode
- $l_a = H$ if probabilities are powers of 2

Huffman Codes

Huffman Algorithm:

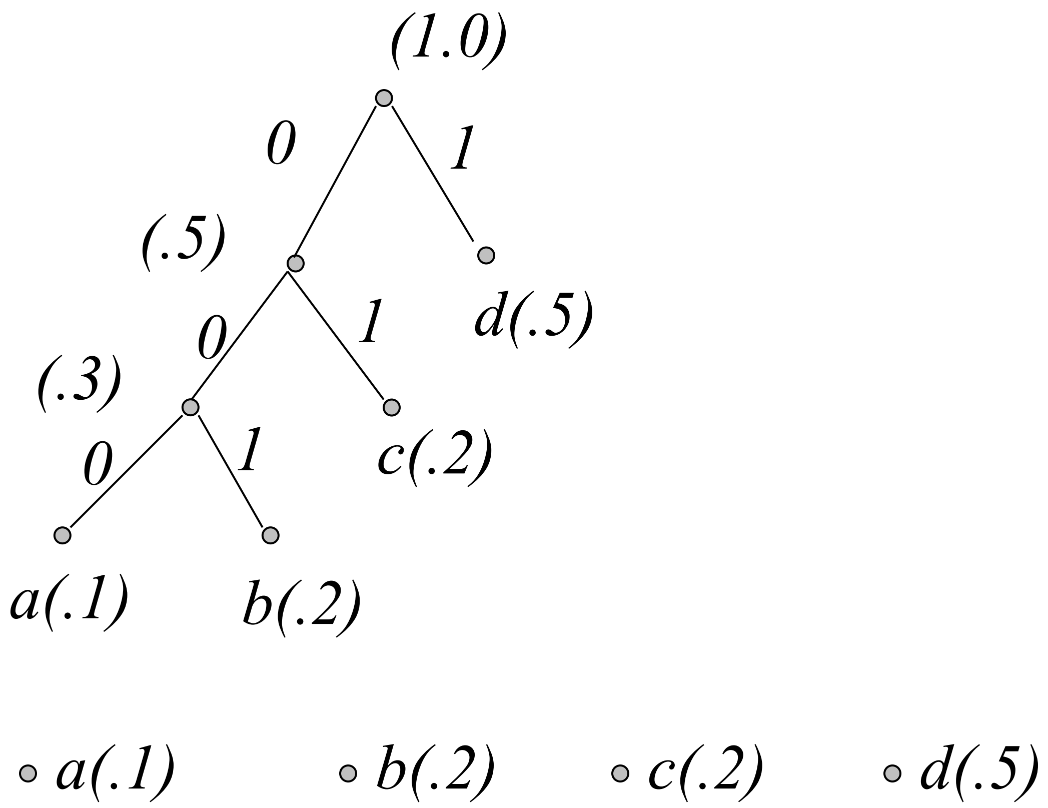
Start with a forest of trees each consisting of a single vertex corresponding to a message *s* and with weight *p(s)*

Repeat until one tree left:

- Select two trees with minimum weight roots p_1 and p_2
- Join into single tree by adding root with weight $p_1 + p_2$

Example

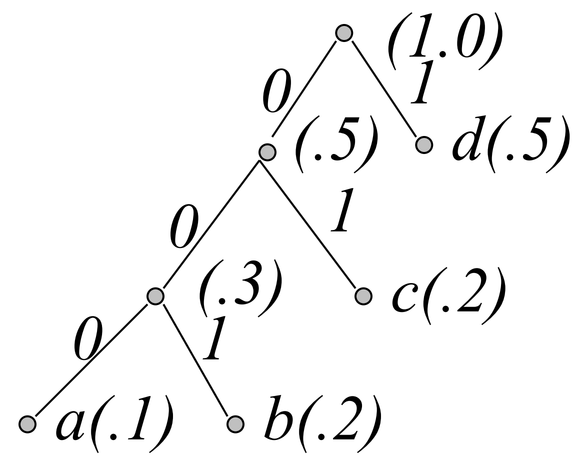
$p(a) = .1, \; p(b) = .2, \; p(c) = .2, \; p(d) = .5$



Encoding and Decoding

Encoding: Start at leaf of Huffman tree and follow path to the root. Reverse order of bits and send.

$a{=}000,\;\; b{=}001,\;\; c{=}01,\; d{=}1$



Decoding: Start at root of Huffman tree and take branch for each bit received. When at leaf can output message and return to root.

Huffman codes are "optimal" (prefix codes)

Theorem: The Huffman algorithm generates an optimal *prefix* code.

Proof outline:

Induction on the number of messages n.

Consider a message set S with $n + 1$ messages

1. Can make it so that least probable messages of S are neighbors in the Huffman tree
2. Replace the two messages with one message with probability $p(m_1) + p(m_2)$ making S'
3. Show that if S' is optimal, then S is optimal
4. S' is optimal by induction

(The proof is in the notes. This is a neat proof! Go through it.)

Problem with Huffman Coding

Consider a message with probability .999. The self information of this message is

$$-\log(.999) = .00144$$

If we were to send a 1000 such messages we might hope to use 1000*.0014 = 1.44 bits.

Using Huffman codes we require at least one bit per message, so we would require 1000 bits.

Need to “blend” bits among message symbols!

Discrete or Blended

Discrete: each message is a fixed set of bits

– E.g., Huffman coding, Shannon-Fano coding

01001	11	0001	011

message: 1 2 3 4

Blended: bits can be "shared" among messages

– E.g., Arithmetic coding

010010111010

message: 1,2,3, and 4

Arithmetic Coding: Introduction

Allows "blending" of bits in a message sequence.

Only requires 3 bits for the example

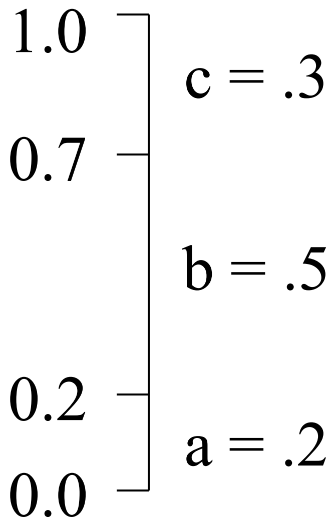
Can bound total bits required based on sum of self information:

$$l < 2 + \sum_{i=1}^{n} s_i$$

Used in many compression algorithms as building block

Arithmetic Coding: message intervals

Assign each message to an interval range from 0 (inclusive) to 1 (exclusive) based on the probabilities.

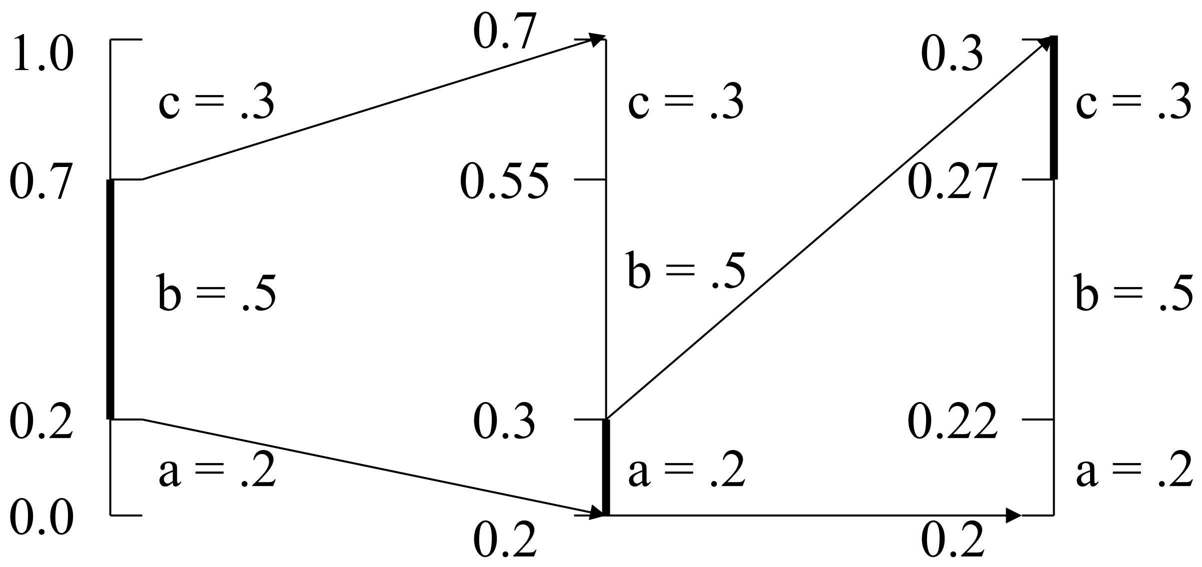


The interval for a particular message will be called the **message interval** (e.g for b the interval is [.2,.7))

Arithmetic Coding: Sequence intervals

Code a message sequence by composing intervals.

For example: **bac**



The final interval is **[.27,.3)**

Uniquely defining an interval

Important property: The sequence intervals for distinct message sequences of length *n* will never overlap

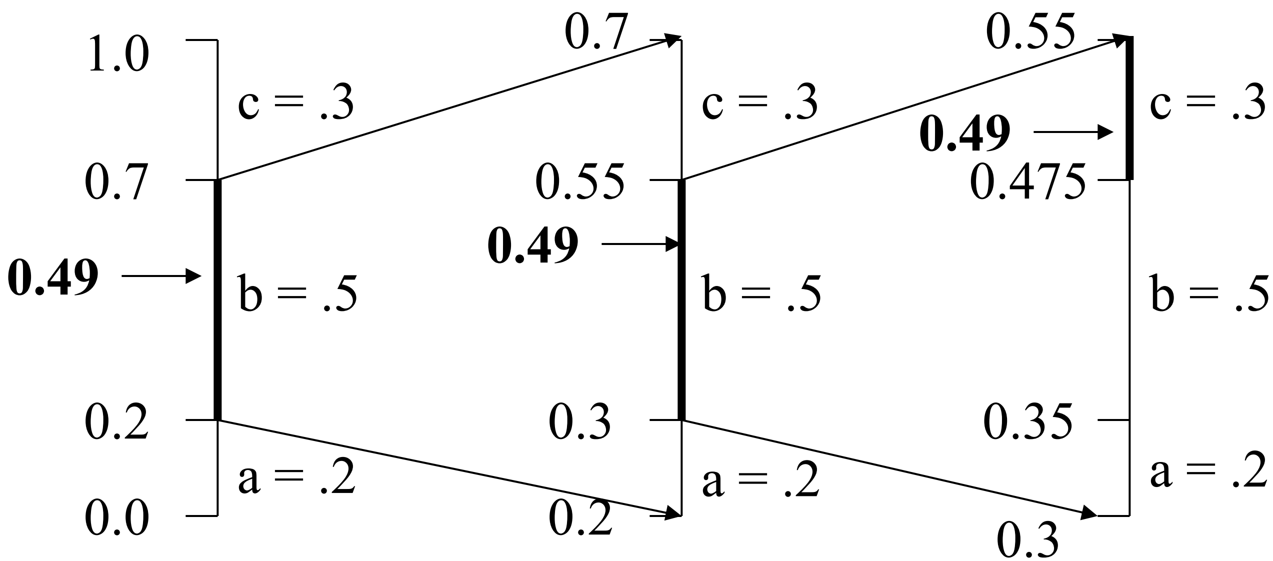
Therefore: specifying any number in the final interval uniquely determines the sequence.

Decoding for Arithmetic Codes:

Decoding is similar to encoding, but on each step need to determine what the message value is and then go backwards

Arithmetic Coding: Decoding Example

Decoding the number .49, knowing the message is of length 3:



The message is **bbc.**

Arithmetic codes: takeaways

- Blending messages into a sequence helps achieve better compression
- Takes closer to the information theoretic lower bound

$$l < 2 + \sum_{i=1}^{n} s_i$$

- Arithmetic codes are more expensive than Huffman coding
  - Due to fractions involved
  - Integer implementations exist and are not too bad (converting all fractions to equivalent integer representations)