Huffman Codes

Uniquely Decodable Codes

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

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`?

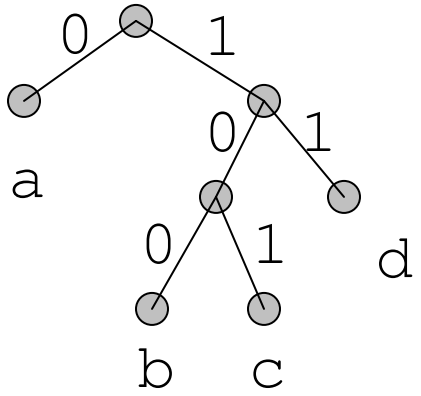
A **uniquely decodable code** is a variable length code in which bit strings can always be uniquely decomposed into its codewords.

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

Can be viewed as a binary tree with symbol values at the leaves and 0 or 1s on the edges.



Some Prefix Codes for Integers

n	Binary	Unary
1	..001	0
2	..010	10
3	..011	110
4	..100	1110
5	..101	11110
6	..110	111110

Many other fixed prefix codes:
Golomb, phased-binary, subexponential, ...

Average Length

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

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

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

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

A 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)

Assume $l(c_i) > l(c_j)$. Consider switching codes c_i and c_j. If ACL is the average length of the original code, the length of the new code is

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

This is a contradiction since ACL is not optimal

Huffman Codes

Invented by Huffman as a class assignment in 1950.

Used in many, if not most, compression algorithms

- gzip, bzip, jpeg (as option), fax compression,…

Properties:

- Generates optimal prefix codes
- Cheap to generate codes
- Cheap to encode and decode

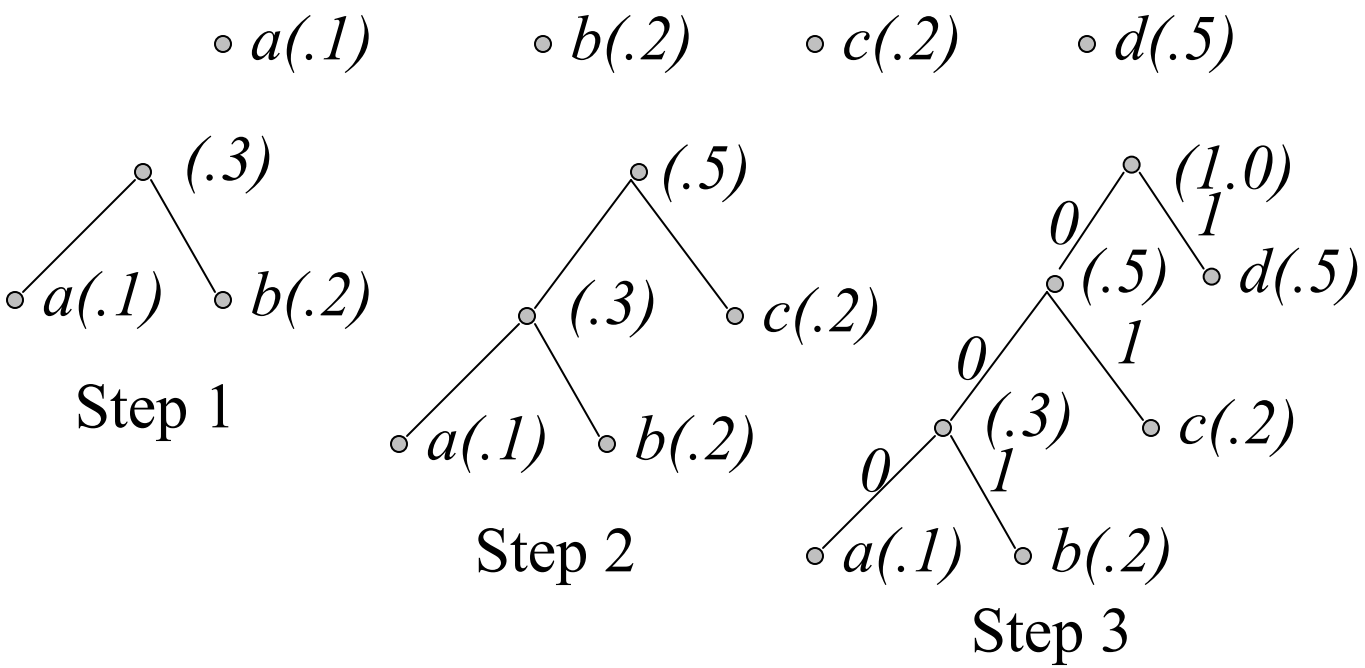
Huffman Codes

Huffman Algorithm

- Start with a forest of trees each consisting of a single vertex corresponding to a symbol *s* and with weight *p(s)*
- ***Repeat:***
  - 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



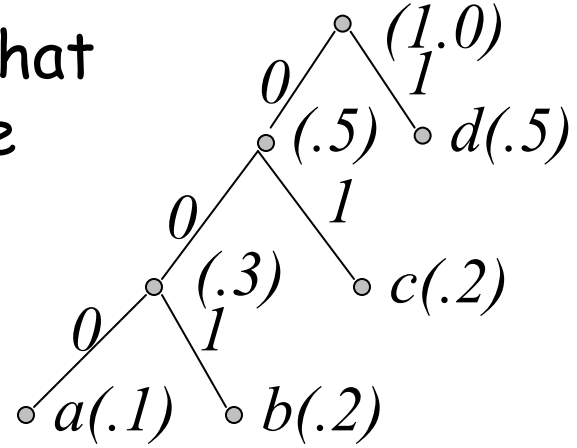
a=000, b=001, c=01, d=1

Encoding and Decoding

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

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

There are even faster methods that can process 8 or 32 bits at a time



Huffman codes are optimal

Lemma 1: *There is an optimal prefix code tree where the two symbols with smallest probabilities are siblings in the last level.*

Proof.

- *An optimal tree is a full binary tree*
- *The two symbols with smallest probability must be in the last level*
- *It is possible to interchange probabilities in the same level without affecting the cost of the tree.*

Huffman codes are optimal

Notation

ABL(T): Average code length of the prefix code induced by tree T.

Huffman codes are optimal

Lemma 2: *Let S' be an alphabet obtained from S by replacing its two symbols of smallest probabilities, say x and y, by na new symbol w and setting p(w)=p(x)+p(y). In addition, let T (T') be the tree constructed by Huffman algorithm for alphabet S(S'). Then, ABL(T)=ABL(T') + p(w)*

Proof.

$$ABL(T) = \sum_{x \in S} p(x) Depth_T(x) =$$

$$p(y) Depth_T(y) + p(z) Depth_T(z) + \sum_{x \in S, x \neq y, x \neq z} p(x) Depth_T(x) =$$

$$p(w)(Depth_{T'}(w) + 1) + \sum_{x \in S, x \neq y, x \neq z} p(x) Depth_T(x) =$$

$$p(w) + \sum_{x \in S'} p(x) Depth_{T'}(x) = p(w) + ABL(T')$$

Huffman codes are optimal

Theorem: *The Huffman Algorithm construct the optimal prefix code.*

Proof. Induction on the size of the alphabet.

For |S|=2 the result holds.

Assume by induction that result holds when |S|<k.

Induction Step: |S|=k.

For the sake of contradiction, assume that result does not hold. Then, there is a tree Z such that ABL(Z) < ABL(T).

By Lmma 1, we can assume that the two symbols x and y with smallest probabilities are siblings at the last level of Z.

Huffman codes are optimal

Let Z' be tree obtained from Z removing these two siblings and replacing its parent by a leaf with probability p(x)+p(y).

From Lemma 2, ABL(Z)=ABL(Z')+p(w)

Since

$$ABL(Z)=ABL(Z')+p(w)<ABL(T)=ABL(T')+p(w),$$

it follows that

$$ABL(Z')<ABL(T'),$$

which contradicts the induction hypothesis.

Implementation

- *Maintain a heap with storing the probabilities of the active nodes.*
- *Running time O(n log n)*