Huffman Coding

• Huffman (1951)
• Uses frequencies of symbols in a string to build a variable rate prefix code.
  – Each symbol is mapped to a binary string.
  – More frequent symbols have shorter codes.
  – No code is a prefix of another.
• Example:
  a 0
  b 100
  c 101
  d 11

Variable Rate Code Example

• Example: a 0, b 100, c 101, d 11
• Coding:
  – aabddca = 16 bits
  – 0 0 100 11 11 101 0 0 = 14 bits
• Prefix code ensures unique decodability.
  – 0010011110100
  – a b d d c a

Cost of a Huffman Tree

• Let $p_1, p_2, \ldots, p_m$ be the probabilities for the symbols $a_1, a_2, \ldots, a_m$, respectively.
• Define the cost of the Huffman tree $T$ to be

$$C(T) = \sum_{i=1}^{m} p_i r_i$$

where $r_i$ is the length of the path from the root to $a_i$.
• $C(T)$ is the expected length of the code of a symbol coded by the tree $T$. $C(T)$ is the bit rate of the code.

Example of Cost

• Example: a 1/2, b 1/8, c 1/8, d 1/4

$$C(T) = 1 \times \frac{1}{2} + 3 \times \frac{1}{8} + 3 \times \frac{1}{8} + 2 \times \frac{1}{4} = 1.75$$

Huffman Tree

• Input: Probabilities $p_1, p_2, \ldots, p_m$ for symbols $a_1, a_2, \ldots, a_m$, respectively.
• Output: A tree that minimizes the average number of bits (bit rate) to code a symbol. That is, minimizes

$$HC(T) = \sum_{i=1}^{m} p_i r_i$$

where $r_i$ is the length of the path from the root to $a_i$. This is the Huffman tree or Huffman code.
Optimality Principle 1
• In an Huffman tree a lowest probability symbol has maximum distance from the root.
  – If not exchanging a lowest probability symbol with one at maximum distance will lower the cost.

\[ C(T) = C(T) + h_p \cdot h_q + k_p = C(T) + (h-k)(p \cdot q) < C(T) \]

Optimality Principle 2
• The second lowest probability is a sibling of the the smallest in some Huffman tree.
  – If not, we can move it there not raising the cost.

\[ C(T) = C(T) + h - h_q + k_q = C(T) + (h-k)(q - p) < C(T) \]

Optimality Principle 3
• Assuming we have a Huffman tree T whose two lowest probability symbols are siblings at maximum depth, they can be replaced by a new symbol whose probability is the sum of their probabilities.
  – The resulting tree is optimal for the new symbol set.

\[ C(T) = C(T) + (h-1)(p+q) - h_p - h_q = C(T) + (p+q) \]

Optimality Principle 3 (cont’)
• If T’ were not optimal then we could find a lower cost tree T”.
  This will lead to a lower cost tree T’’ for the original alphabet.

\[ C(T’’) = C(T’’) + p + q < C(T) + p + q = C(T) \] which is a contradiction

Recursive Huffman Tree Algorithm
1. If there is just one symbol, a tree with one node is optimal. Otherwise
2. Find the two lowest probability symbols with probabilities p and q respectively.
3. Replace these with a new symbol with probability p + q.
4. Solve the problem recursively for new symbols.
5. Replace the leaf with the new symbol with an internal node with two children with the old symbols.

Iterative Huffman Tree Algorithm
form a node for each symbol a with weight p;
insert the nodes in a min priority queue ordered by probability;
while the priority queue has more than one element do
  min1 := delete-min;
  min2 := delete-min;
  create a new node n;
  n.weight := min1.weight + min2.weight;
  n.left := min1;
  n.right := min2;
  insert(n);
return the last node in the priority queue.
Example of Huffman Tree Algorithm (1)
- \( P(a) = .4, P(b) = .1, P(c) = .3, P(d) = .1, P(e) = .1 \)

Example of Huffman Tree Algorithm (2)

Example of Huffman Tree Algorithm (3)

Example of Huffman Tree Algorithm (4)

Huffman Code
- average number of bits per symbol is 
  \( .4 \times 1 + .1 \times 4 + .3 \times 2 + .1 \times 3 + .1 \times 4 = 2.1 \)

Optimal Huffman Code vs. Entropy
- \( P(a) = .4, P(b) = .1, P(c) = .3, P(d) = .1, P(e) = .1 \)
  
  Entropy
  \[ H = - (.4 \times \log_2(.4) + .1 \times \log_2(.1) + .3 \times \log_2(.3) + .1 \times \log_2(.1) + .1 \times \log_2(.1)) \]
  \[ = 2.05 \text{ bits per symbol} \]

Huffman Code
- \( HC = .4 \times 1 + .1 \times 4 + .3 \times 2 + .1 \times 3 + .1 \times 4 \)
  \[ = 2.1 \text{ bits per symbol} \]
  pretty good!
In Class Exercise
- P(a) = 1/2, P(b) = 1/4, P(c) = 1/8, P(d) = 1/16, P(e) = 1/16
- Compute the Huffman tree and its bit rate.
- Compute the Entropy
- Compare
- Hint: For the tree change probabilities to be integers: a:8, b:4, c:2, d:1, e:1. Normalize at the end.

Quality of the Huffman Code
- The Huffman code is within one bit of the entropy lower bound.
  \[ H \leq HC \leq H + 1 \]
- Huffman code does not work well with a two symbol alphabet.
  Example: P(0) = 1/100, P(1) = 99/100
  \[ HC = 1 \text{ bit/symbol} \]
  \[ H = -(1/100) \log_2(1/100) + (99/100) \log_2(99/100) = .08 \text{ bit/symbol} \]

Powers of Two
- If all the probabilities are powers of two then \( HC = H \)
- Proof by induction on the number of symbols.
  Let \( p_1 \leq p_2 \leq \ldots \leq p_n \) be the probabilities that add up to 1.
  If \( n = 1 \) then \( HC = H \) (both are zero).
  If \( n > 1 \) then \( p_i = 2^k \) for some \( k \), otherwise the sum cannot add up to 1.
  Combine the first two symbols into a new symbol of probability \( 2^k + 2^k = 2^{k+1} \).

Powers of Two (Cont.)
By the previous page,
\[ HC(p_i, p_2, \ldots, p_n) = H(p_i, p_2, \ldots, p_n) - (p_i + p_2) \]
By the properties of Huffman trees (principle 3),
\[ HC(p_1, p_2, \ldots, p_n) = HC(p_1, p_2, p_3, \ldots, p_n) + (p_1 + p_2) \]
Hence,
\[ HC(p_1, p_2, \ldots, p_n) = H(p_1, p_2, \ldots, p_n) \]

Powers of Two (Cont.)
By the induction hypothesis
\[ HC(p_1, p_2, \ldots, p_n) = H(p_1, p_2, \ldots, p_n) \]
\[ = -(p_1 + p_2) \log_2(p_1 + p_2) + \sum_{i=3}^{n} p_i \log_2(p_i) \]
\[ = -2^{-n} \log_2(2^{-n}) - \left( \sum_{i=3}^{n} p_i \log_2(p_i) \right) \]
\[ = \sum_{i=3}^{n} p_i \log_2(p_i) - (p_1 + p_2) \]
\[ = H(p_1, p_2, \ldots, p_n) - (p_1 + p_2) \]
Extending the Alphabet
- Assuming independence \( P(ab) = P(a)P(b) \), so we can lump symbols together.
  Example: P(0) = 1/100, P(1) = 99/100
  \[ P(00) = 1/10000, P(01) = P(10) = 99/10000, \]
  \[ P(11) = 9801/10000. \]
  \[ HC = 1.03 \text{ bits/symbol (2 bit symbol)} \]
  \[ = .515 \text{ bit/bit} \]
  Still not that close to \( H = .08 \text{ bit/bit} \)
Quality of Extended Alphabet

- Suppose we extend the alphabet to symbols of length k then
  \[ H \leq HC \leq H + \frac{1}{k} \]

- Pros and Cons of Extending the alphabet
  - Better compression
  - Padding needed to make the length of the input divisible by k

Huffman Codes with Context

- Suppose we add a one symbol context. That is in compressing a string \( x_1x_2...x_n \) we want to take into account \( x_k \) when encoding \( x_k \).
  - New model, so entropy based on just independent probabilities of the symbols doesn’t hold. The new entropy model (2nd order entropy) has for each symbol a probability for each other symbol following it.
  - Example: (a,b,c)

\[
\begin{array}{ccc}
\text{prev} & \text{next} \\
\hline
a & 4 & 2 \\
b & 1 & 9 \\
c & 1 & 3 \\
\end{array}
\]

Multiple Codes

- Time to design Huffman Code is \( O(n \log n) \) where \( n \) is the number of symbols.

- Each step consists of a constant number of priority queue operations (2 deletem’s and 1 insert)

Approaches to Huffman Codes

1. Frequencies computed for each input
   - Must transmit the Huffman code or frequencies as well as the compressed input
   - Requires two passes
2. Fixed Huffman tree designed from training data
   - Do not have to transmit the Huffman tree because it is known to the decoder.
   - H.263 video coder
3. Adaptive Huffman code
   - One pass
   - Huffman tree changes as frequencies change