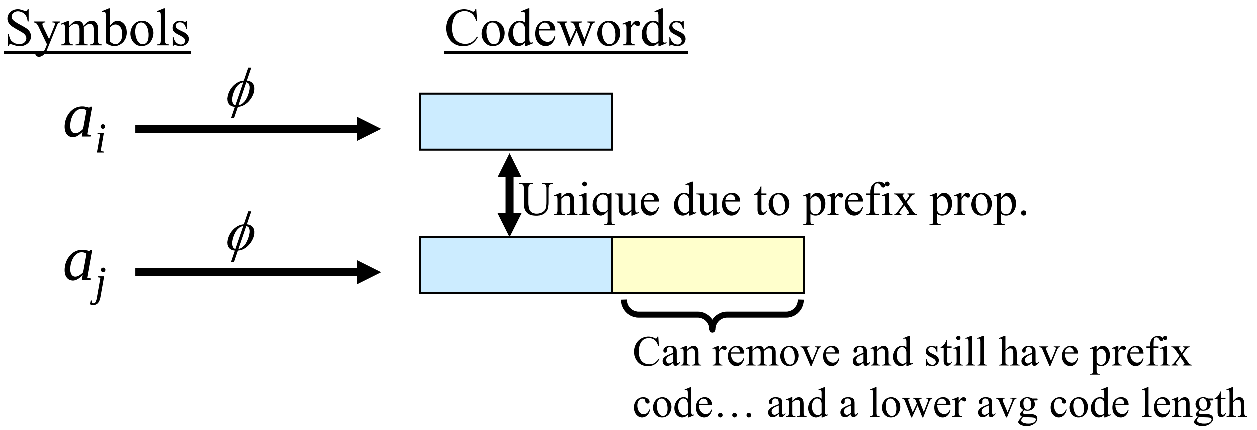
# Ch. 3 Huffman Coding

# Two Requirements for *Optimum* *Prefix* Codes

1. Likely Symbols → Short Codewords
   Unlikely Symbols → Long Codewords
   <Recall Entropy Discussion>
2. The two least likely symbols have codewords of the same length

**Why #2????**

Suppose two least likely symbols have different lengths:



Unique due to prefix prop.

Can remove and still have prefix code… and a lower avg code length

# Additional Huffman Requirement

The two least likely symbols have codewords that differ only in the last bit

These three requirements lead to a simple way of building a <u>binary tree</u> describing *<u>an</u>* optimum prefix code - *THE* Huffman Code

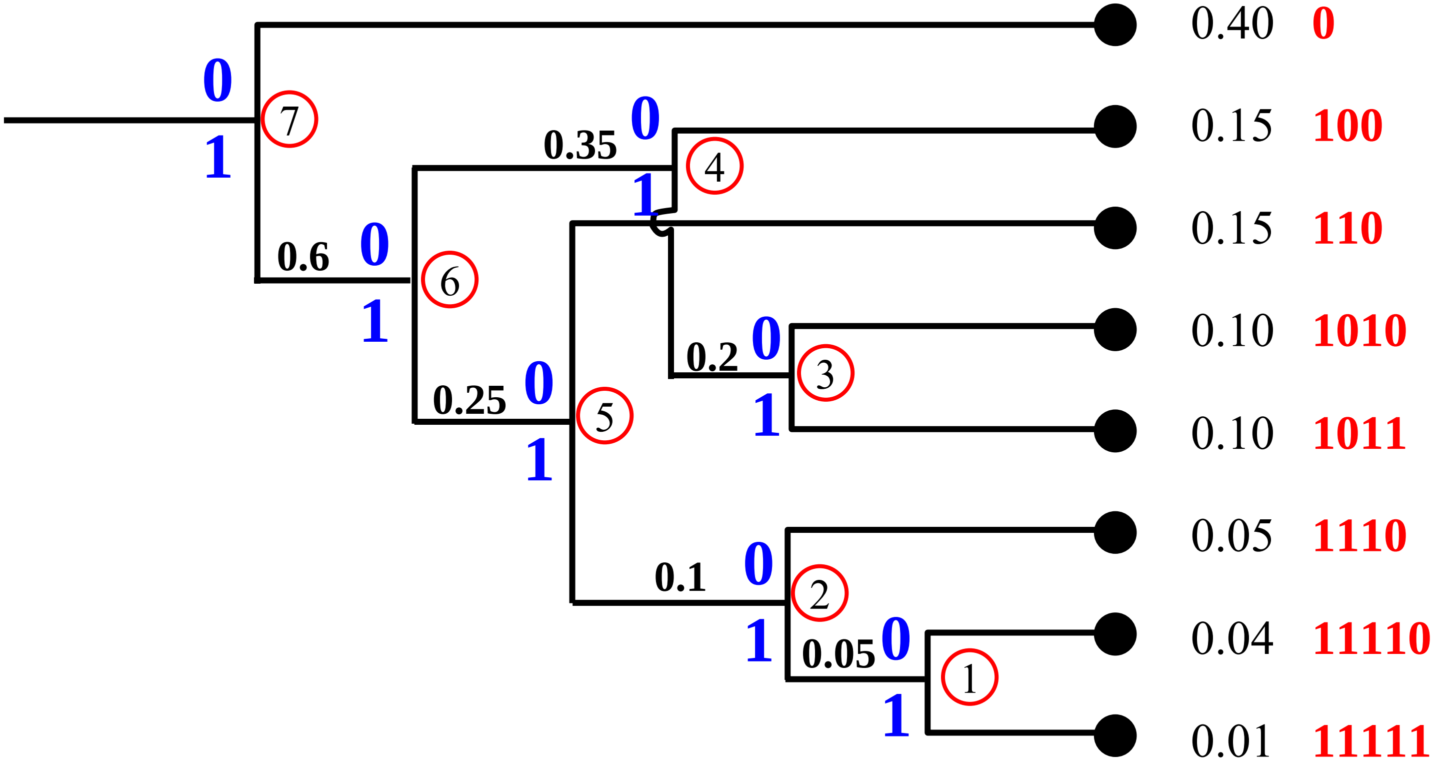
- Build it from bottom up, starting w/ the two <u>least</u> likely symbols
- The external nodes correspond to the symbols
- The internal nodes correspond to “super symbols” in a “reduced” alphabet

# Huffman Design Steps

1. Label each node w/ one of the source symbol probabilities
2. Merge the nodes labeled by the two smallest probabilities into a parent node
3. Label the parent node w/ the sum of the two children's probabilities
   - This parent node is now considered to be a "super symbol" (it replaces its two children symbols) in a reduced alphabet
4. Among the elements in reduced alphabet, merge two with smallest probs.
   - If there is more than one such pair, choose the pair that has the "lowest order super symbol" (this assure the minimum variance Huffman Code – see book)
5. Label the parent node w/ the sum of the two children probabilities.
6. Repeat steps 4 & 5 until only a single super symbol remains

# Example of Huffman Design Steps

1. Label each node w/ one of the source symbol probabilities
2. Merge the nodes labeled by the two smallest probabilities into a parent node
3. Label the parent node w/ the sum of the two children's probabilities
4. Among the elements in reduced alphabet, merge two with smallest probs.
5. Label the parent node w/ the sum of the two children probabilities.
6. Repeat steps 4 & 5 until only a single super symbol remains



# Performance of Huffman Codes

Skip the details, State the results

**How close to entropy $H(S)$ can Huffman get?**

**Result #1**: If all symbol probabilities are powers of two then $\bar{l} = H_1(S)$

Info of each symbol is
an integer # of bits

**Result #2**: $\underbrace{H_1(S) \le \bar{l}}_{\bar{l} - H_1(S) \,=\, \text{Redundancy}} < H_1(S) + 1$

**Result #3**: Refined Upper Bound $\bar{l} < \begin{cases} H_1(S) + P_{max}, & P_{\max} < 0.5 \\ H_1(S) + P_{max} + 0.086, & P_{\max} \ge 0.5 \end{cases}$

**Note: Large alphabets tend to have small $P_{max}$ ➔ Huffman Bound Better**

**Small alphabets tend to have large $P_{max}$ ➔ Huffman Bound Worse**

# Applications of Huffman Codes

### Lossless Image Compression Examples

Directly: $1.14 \leq CR \leq 1.67$

Differences: $1.66 \leq CR \leq 2.03$

Not That Great!

### Text Compression Example

Applied to Ch. 3: $CR = 1.63$

Not That Great!

### Lossless Audio Compression Examples

Directly: $1.16 \leq CR \leq 1.3$

Differences: $1.47 \leq CR \leq 1.65$

Not That Great!

So… why have we looked at something so bad???

- Provides good intro to compression ideas
- Historical result & context
- Huffman is often used as building block in more advanced methods
  - Group 3 FAX (Lossless)
  - JPEG Image (Lossy)
  - Etc…

# Block Huffman Codes (or "Extended" Huffman Codes)

- Useful when Huffman not effective due to large $P_{max}$
- Example: IID Source w/ $P(a_1) = 0.8 \quad P(a_2) = 0.02 \quad P(a_3) = 0.18$
- Book shows that Huffman gives 47% more bits than the entropy!!
- Block codes allow better performance
  - Because they allow noninteger # bits/symbol
- Note: assuming IID… means that no context can be exploited
  - If source is not IID we can do better by exploiting context model
- Group into $n$-symbol blocks ➔
  - map between original alphabet & a new "extended" alphabet

$$\{a_1, a_2, \ldots a_m\} \rightarrow \underbrace{\left\{ \underbrace{(a_1 a_1 \cdots a_1)}_{n \text{ times}}, (a_1 \cdots a_1 a_2), \cdots, (a_m \cdots a_m a_m) \right\}}_{m^n \text{ elements in new alphabet}}$$

Need $m^n$ codewords… use Huffman procedure on probs of blocks

Block probs determined using IID: $P(a_i, a_j, \ldots a_p) = P(a_i)P(a_j)\cdots P(a_p)$

# Performance of Block Huffman Codes

- Let $S^{(n)}$ denote the block source (with the scalar source IID)

  $R^{(n)}$ denote the rate of the block Huffman code (bits/block)

  $H(S^{(n)})$ be the entropy of the block source
- Then, using bounds discussed earlier

$$H(S^{(n)}) \leq R^{(n)} < H(S^{(n)}) + 1 \quad \Rightarrow \quad \boxed{\frac{H(S^{(n)})}{n} \leq R < \frac{H(S^{(n)})}{n} + \frac{1}{n}}$$

\# bits per $n$ symbols
➔ $R = R^{(n)}/n$ # bits/symbol

- Now, how is $H(S^{(n)})$ related to $H(S)$?
  - See p. 53 of 3rd edition, which uses independence & properties of log
  - After much math manipulation we get $\boxed{H(S^{(n)}) = nH(S)}$

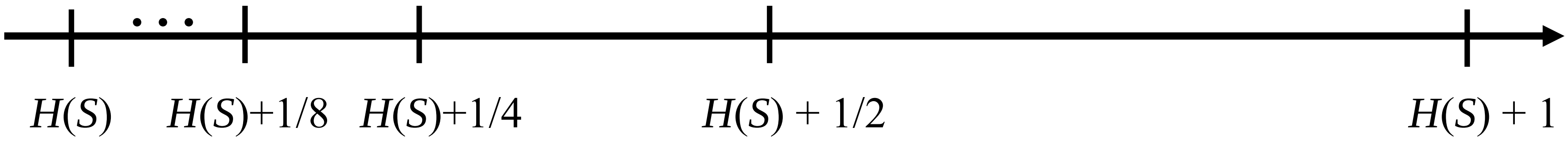
Makes Sense: - Each symbol in block gives $H(S)$ bits of info
- Indep. ➔ no “shared” info between sequence
- Info is additive for Indep. Seq.

$$\begin{aligned} H(S^{(n)}) &= H(S) + H(S) + \cdots + H(S) \\ &= nH(S) \end{aligned}$$

# Final Result for Huffman Block Codes w/ IID Source

$$\boxed{H(S) \leq R < H(S) + \frac{1}{n}}$$

$n = 1$ is case of “ordinary” single symbol Huffman case we looked at earlier



- As blocks get larger, Rate approaches $H(S)$
- Thus, longer blocks lead to the “Holy Grail” of compressing down to the entropy…

**BUT**… $\underbrace{\textbf{\# of codewords grows Exponentially: } m^n}_{\textbf{Impractical!!!}}$