Adaptive Huffman Coding

- One pass
- During the pass calculate the frequencies
- Update the Huffman tree accordingly
  - Coder – new Huffman tree computed after transmitting the symbol
  - Decoder – new Huffman tree computed after receiving the symbol
- Symbol set and their initial codes must be known ahead of time.
- Need NYT (not yet transmitted symbol) to indicate a new leaf is needed in the tree.

Optimal Tree Numbering

\[
\begin{align*}
& a : 5, b : 2, c : 1, d : 3 \\
\end{align*}
\]

Weight the Nodes

\[
\begin{align*}
& a : 5, b : 2, c : 1, d : 3 \\
\end{align*}
\]

Number the Nodes

\[
\begin{align*}
& a : 5, b : 2, c : 1, d : 3 \\
\end{align*}
\]

Adaptive Huffman Principle

- In an optimal tree for n symbols there is a numbering of the nodes \( Y_1 < Y_2 < \ldots < Y_{2n-1} \) such that their corresponding weights \( x_1, x_2, \ldots, x_{2n-1} \) satisfy:
  - \( x_1 < x_2 < \ldots < x_{2n-1} \)
  - siblings are numbered consecutively
- And vice versa
  - That is, if there is such a numbering then the tree is optimal. We call this the node number invariant.
Initialization

• Symbols $a_1, a_2, \ldots, a_m$ have a basic prefix code, used when symbols are first encountered.
• Example: a, b, c, d, e, f, g, h, i, j

C s e 4 9 0 g z - L e c t u r e 3 - W i n t e r 2 0 0 4

Coding Algorithm
1. If a new symbol is encountered then output the code for NYT followed by the fixed code for the symbol. Add the new symbol to the tree.
2. If an old symbol is encountered then output its code.
3. Update the tree to preserve the node number invariant.

Decoding Algorithm
1. Decode the symbol using the current tree.
2. If NYT is encountered then use the fixed code to decode the symbol. Add the new symbol to the tree.
3. Update the tree to preserve the node number invariant.

Updating the Tree
1. Let $y$ be leaf (symbol) with current weight $x$.
2. If $y$ the root update $x$ by 1, otherwise.
3. Exchange $y$ with the largest numbered node with the same weight (unless it is the parent).
4. Update $x$ by 1
5. Let $y$ be the parent with its weight $x$ and go to 2.

Example
- $a b c d a d$ in alphabet {a, b, ..., j}
- Output = 000

We never update the weight of NYT
** This exchange will preserve the node number invariant
Example

- aabcdad

Output = 000

Example

- aabcdad

Output = 0001

Example

- aabcdad

Output = 00010001

NYT

fixed code for b

Example

- aabcdad

Output = 00010001
Example

• aabcdad

output = 00010001

Example

• aabgdad

output = 0001000100010

Example

• aabcdad

output = 0001000100010

Example

• aabgdad

output = 0001000100010

Example

• aabcdad

output = 0001000100010
Example

- aabc\text{g}ad

\[ \begin{array}{c}
\text{output} = 00010001000100000011 \\
\end{array} \]

Example

- aabc\text{g}ad

\[ \begin{array}{c}
\text{NYT} \\
\text{output} = 00010001000100000011 \\
\end{array} \]

Example

- aabc\text{g}ad

\[ \begin{array}{c}
\text{NYT} \\
\text{exchange!} \\
\end{array} \]

Example

- aabc\text{g}ad

\[ \begin{array}{c}
\text{NYT} \\
\text{output} = 00010001000100000011 \\
\end{array} \]

Example

- aabc\text{g}ad

\[ \begin{array}{c}
\text{NYT} \\
\text{exchange!} \\
\end{array} \]

Example

- aabc\text{g}ad

\[ \begin{array}{c}
\text{NYT} \\
\text{output} = 00010001000100000011 \\
\end{array} \]
Example

- aabcdad

output = 000100010001000111

Note: the first a is coded as 000, the second as 1, and the third as 0

- aabcdad

output = 0001000100010001110

- aabcdad

output = 000100010001000111101101

- aabcdad

exchange!

output = 0001000100010001111101101

- aabcdad

output = 00010001000100011111111111

- aabcdad

output = 00010001000100011111111111

Note: the first a is coded as 000, the second as 1, and the third as 0
Example
• aabcdag

Example
• aabcdad

Example
• aabcdad

Data Structure for Adaptive Huffman
1. Fixed code table
2. Binary tree with parent pointers
3. Table of pointers nodes into tree
4. Doubly linked list to rank the nodes

In Class Exercise
• Decode using adaptive Huffman coding assuming the following fixed code

Huffman Summary
• Statistical compression algorithm
• Prefix code
• Fixed-to-variable rate code
• Optimization to create a best code
• Symbol merging
• Context
• Adaptive coding
• Decoder and encoder behave almost the same
• Need for data structures and algorithms