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

- a : 5, b : 2, c : 1, d : 3

Weight the Nodes

- a : 5, b : 2, c : 1, d : 3

Number the Nodes

- a : 5, b : 2, c : 1, d : 3

Adaptive Huffman Principle

- In an optimal tree for n symbols there is a numbering of the nodes \( y_1 < y_2 < \ldots < y_{2^n - 1} \) such that their corresponding weights \( x_1, x_2, \ldots, x_{2^n - 1} \) satisfy:
  - \( x_1 \leq x_2 \leq \ldots \leq x_{2^n - 1} \)
  - 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 $a_1, a_2, ..., a_m$ have a basic prefix code, used when symbols are first encountered.
- Example: $a, b, c, d, e, f, g, h, i, j$

The tree will encode up to $m + 1$ symbols including NYT.
- We reserve numbers 1 to $2m + 1$ for node numbering.
- The initial Huffman tree consists of a single node

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.

Example

- $aabcdad$ 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

Example

• aabcdad

output = 00010001

Example

• aabcdad

output = 00010001
Example

• aabcdad

Example

• aabcdad

Example

• aabcdad

Example

• aabcdad

Example

• aabcdad

Example

• aabcdad
Example

• aabcd

\[ \begin{align*}
\text{NYT} & \rightarrow 15 \\
17 & \rightarrow 18 \\
19 & \rightarrow 20
\end{align*} \]

output = 0001000100010000011

exchange!

Example

• aabcd

\[ \begin{align*}
\text{NYT} & \rightarrow 15 \\
17 & \rightarrow 18
\end{align*} \]

output = 0001000100010000011

Example

• aabcd

\[ \begin{align*}
\text{NYT} & \rightarrow 15 \\
17 & \rightarrow 18
\end{align*} \]

output = 0001000100010000011

Example

• aabcd

\[ \begin{align*}
\text{NYT} & \rightarrow 15 \\
17 & \rightarrow 18
\end{align*} \]

output = 0001000100010000011

Example

• aabcd

\[ \begin{align*}
\text{NYT} & \rightarrow 15 \\
17 & \rightarrow 18
\end{align*} \]

output = 0001000100010000011

Example

• aabcd

\[ \begin{align*}
\text{NYT} & \rightarrow 15 \\
17 & \rightarrow 18
\end{align*} \]

output = 0001000100010000011
Example

• **aabcd**

```
output = 0001000100010000011
```

Example

• **aabcd**

```
output = 0001000100010000011
```

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

Example

• **aabcd**

```
output = 00010001000100000110
```

Exchange!

Example

• **aabcd**

```
output = 000100010001000001101101
```

Example

• **aabcd**

```
output = 000100010001000001101101
```

Example

• **aabcd**

```
output = 000100010001000001101101
```
**Example**

- aabcdad

```
0 1
1 0
```

Output = 000100010001000001101101

**Example**

- aabcdad

```
0 1
1 0
```

Output = 000100010001000001101101

**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

```
0 1 1
1 1 0
```

- 00110000

**Huffman Summary**

- Statistical compression algorithm
- Prefix code
- Fixed-to-variable length 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