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 basic 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 \( y_1 < y_2 < ... < y_{2n-1} \) such that their corresponding weights \( x_1, x_2, ..., x_{2n-1} \) satisfy:
  - \( x_1 \leq x_2 \leq ... \leq x_{2n-1} \)
  - 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: $a, b, c, d, e, f, g, h, i, j$

**In Discussion**

- 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.

**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.

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

**Example**

- $abcdad$ in alphabet {a,b,..., j}

- NYT

- Output = 000

- Fixed code for a
Example

• aabcdad

output = 000

Example

• aabcdad

output = 0001

Example

• aabcdad

output = 00010001

Example

• aabcdad

output = 00010001
Example

• aabcdad

output = 00010001

Example

• aabcdad

output = 0001000100010

Example

• aabcdad

output = 0001000100010

Example

• aabcdad

output = 0001000100010
Example

- aabcdad

fixed code for d

output = 00010001000100000111

Example

- aabcdad

output = 00010001000100000111

Example

- aabcdad

output = 00010001000100000111

Example

- aabcdad

exchange!

output = 00010001000100000111

Example

- aabcdad

exchange!

output = 00010001000100000111
Example

• aabcdad

output = 000100010001000011

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

output = 0001000100010000110

Example

• aabcdad

output = 0001000100010000110

exchange!

Example

• aabcdad

output = 00010001000100001101

Example

• aabcdad

output = 0001000100010000110110

Example

• aabcdad

output = 00010001000100001101101

Example

• aabcdad

output = 000100010001000011011011

Example

• aabcdad

output = 0001000100010000110110110
Example

- aabcdad

output = 000100010001000011011101

Example

- aabcdad

output = 000100010001000011011101

Data Structure for Adaptive Huffman

1. Fixed code table
2. Binary tree with parent pointers
3. Table of pointers nodes in tree
4. Doubly linked list to rank the nodes

In Class Exercise

- Decode using adaptive Huffman coding assuming the following fixed code

Example of Huffman

- 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