Expression Trees and Huffman Encoding: Key Data Structures in Computing

This presentation explores fundamental data structures: expression trees, Huffman code trees, and threaded binary trees. We will cover their construction and diverse applications. These structures are crucial for computation and data compression.

Constructing Expression Trees Using Stacks

Expression Parsing

- Infix: 3 + 4 * 5
- Postfix: 3 4 5 * +
- Prefix: + 3 * 4 5

Algorithm with Stack

Convert postfix to a binary expression tree. Operands are pushed onto the stack. Operators pop operands, form a subtree, then push the root back.

Evaluating Expression Trees

1

Postorder Traversal

Evaluate left subtree, then right, then root. This ensures operands are processed before operators.

2

Example: "3 4 + 5 *"

First, 3 and 4 are added. Then, the result (7) is multiplied by 5, yielding 35.

3

Key Benefits

Efficient parsing. Supports complex expressions. Handles operator precedence naturally. Versatile for interpreters.

Introduction to Huffman Encoding for Data Compression

Lossless Compression

Reduces file size without losing data. Perfect for text, executable files.

Variable-Length Codes

Frequently occurring symbols get shorter binary codes. Less frequent symbols get longer codes.

Invented in 1952

David A. Huffman developed this elegant algorithm during his PhD studies.

Building the Huffman Code Tree

1

Create Leaf Nodes

Each unique symbol becomes a leaf node. Its frequency is its weight.

2

Combine Lowest Frequencies

Select two nodes with the smallest weights. Combine them into a new parent node.

3

Assign Parent Weight

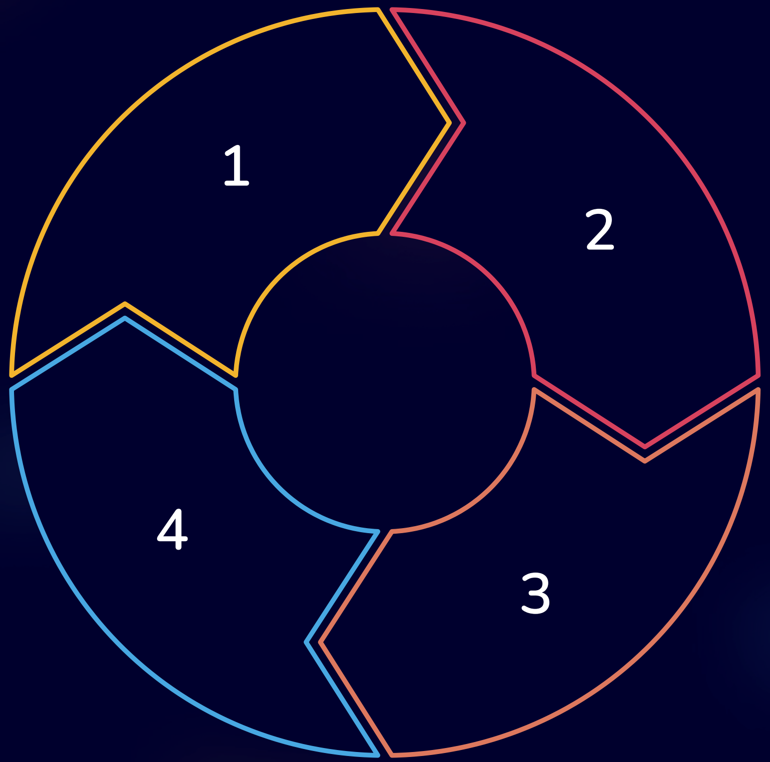
The new parent's weight is the sum of its children's weights.

4

Repeat Until Root

Continue combining nodes until only one root node remains. This forms the complete Huffman tree.

Generating Huffman Codes from the Tree



Traverse the Tree

Start from the root node. Follow paths to each leaf symbol.

Assign Binary Values

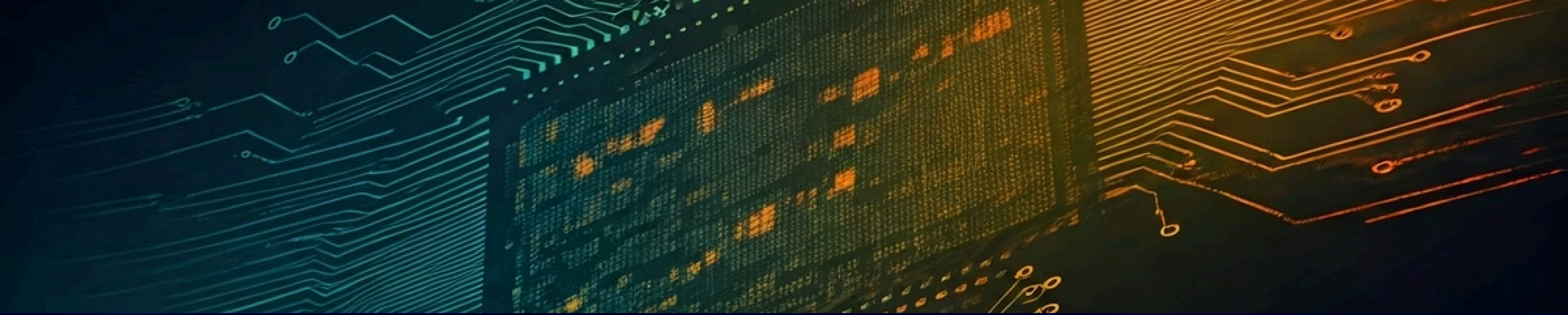
Assign '0' to every left branch. Assign '1' to every right branch.

Record Path

The sequence of 0s and 1s from root to leaf forms the symbol's code.

Example Codes

'a' = 0 (if left), 'b' = 10 (right then left), 'c' = 11 (right then right).



Decoding with Huffman Code Tree

Shared Tree

The same Huffman tree used for encoding is used for decoding. It acts as a lookup table.

Prefix-Free Property

No Huffman code is a prefix of another code. This ensures unambiguous decoding.

Efficient Recovery

Read incoming bits one by one. Traverse the tree following the bits. When a leaf is reached, a symbol is decoded.

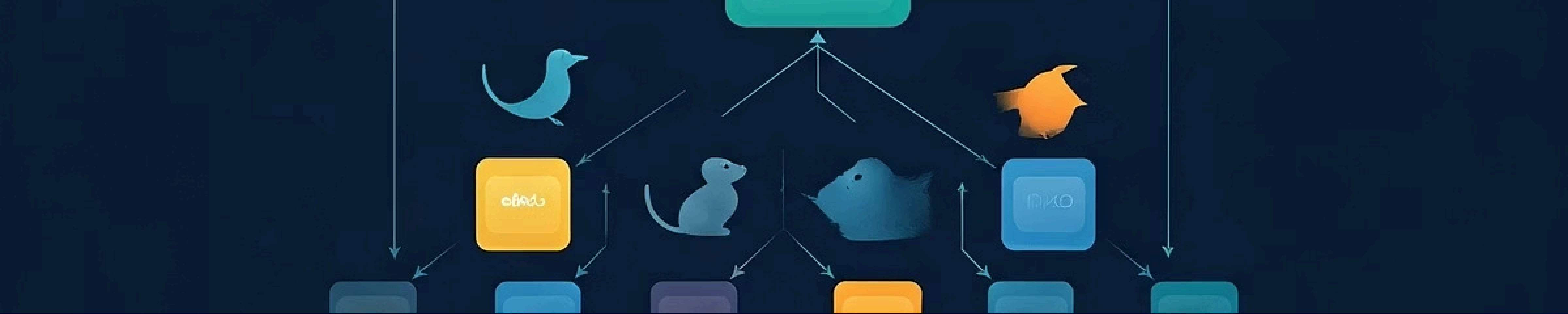
Threaded Binary Trees: Motivation and Basics

Threads, Not Nulls

Null pointers in a binary tree are replaced. They point to in-order predecessors or successors.

Traversal Efficiency

Eliminates the need for a stack or recursion during in-order traversal. This saves memory.



In-order Traversal of Threaded Binary Trees & Array Storage

1. ### Threaded Traversal

   In-order traversal is simplified. Follow 'threads' to find the next node. Achieves O(N) time complexity.

2. ### Complete Binary Tree

   A tree where every level, except possibly the last, is fully filled. All nodes are as far left as possible.

3. ### Array Storage

   Can be stored efficiently in an array. Left child is at index 2i+1. Right child at 2i+2.

4. ### Benefits

   Efficient memory usage. Fast indexing of nodes. Reduces memory overhead.

Summary and Practical Applications

Expression Trees

Used in interpreters, compilers, and database query optimizers for parsing complex expressions.

Huffman Trees

Found in data compression standards. Examples include ZIP files, JPEG images, and audio codecs.

Threaded Trees

Enhance database indexing and search algorithms. Critical for efficient memory management.