Adaptive Huffman Coding Implementation Report

Course: DSAI 325 - Introduction to Information Theory

---

1. Introduction

Adaptive Huffman Coding is a dynamic data compression algorithm that builds and updates a Huffman tree in real-time during encoding and decoding.
This project implements the algorithm in Java, **featuring:**

- Dynamic tree adaptation (insertion, frequency updates, node swapping)
- Efficient encoding/decoding with NYT (Not Yet Transmitted) symbol handling
- Visualization (JavaFX) of the Huffman tree
- Compression ratio tracking

---

2. Implementation

2.1 Project Structure

```
adaptive-huffman-coding/
├── src/
│   ├── AdaptiveHuffman.java       # Driver class
│   ├── Node.java                  # Tree node structure
│   ├── HuffmanTree.java           # Tree operations
│   ├── Encoder.java               # Compression logic
│   ├── Decoder.java               # Decompression logic
├── test/
│   ├── AdaptiveHuffmanTest.java   # Unit tests
├── visualization/
│   ├── HuffmanTreeVisualizer.java # JavaFX GUI
└── report.pdf                     # This document
```

2.2 Core Components

(1) Node.java

```
char symbol;  // '\0' for NYT/internal nodes
int weight;   // Frequency count
int nodeNumber; // Creation order
Node left, right, parent;
```

- **Methods:**
  - isLeaf(): Checks if node is a symbol leaf
  - isNYT(): Checks if node is NYT

(2) HuffmanTree.java

- **Key Methods:**
  - insert(char c): Splits NYT node → new NYT + symbol node
  - updateTree(Node n): Increments weights + rebalances tree
  - swapNodes(Node a, Node b): Maintains sibling property
  - getCode(Node n): Generates Huffman code (0=left, 1=right)

(3) Encoder.java

- **For each symbol:**
  - New symbol: Output [NYT code] + [8-bit ASCII]
  - Existing symbol: Output its Huffman code
- **Example:**

  'a' (new) → "0" (NYT) + "01100001" (ASCII 97) → "01100001"

  'a' (existing) → "1"

(4) Decoder.java

- **Traverses tree using input bits:**
  - NYT node: Reads next 8 bits as new symbol
  - Leaf node: Outputs symbol + updates tree

(5) Visualization (HuffmanTreeVisualizer.java)

- **Node Colors:**

  NYT: Light green

  Leaf: Light blue (with symbol)

  Internal: White
- **Edge Labels:** "0" (left), "1" (right)

---

3. Test Cases & Results

3.1 Test Case 1: "aacbdad"

Input: a a c b d a d
OUTPUT Steps:

```
Code for node NYT (node #1):
After encoding the character 'a' (character #1), the compressed stream is '01100001'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 1
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 1)


After encoding the character 'a' (character #2), the compressed stream is '011000011'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 2
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 2)


After encoding the character 'c' (character #3), the compressed stream is '011000011001100011'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 2
symbol 'c' with code 01 and count 1
NYT node with code 00 and count 0
NYT: 00 (weight: 0)
'c': 01 (weight: 1)
'a': 1 (weight: 2)


After encoding the character 'b' (character #4), the compressed stream is
'0110000110011000110001100010'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 2
symbol 'b' with code 01 and count 1
symbol 'c' with code 001 and count 1
NYT node with code 000 and count 0
NYT: 000 (weight: 0)
'c': 001 (weight: 1)
'b': 01 (weight: 1)
'a': 1 (weight: 2)
```

After encoding the character 'd' (character #5), the compressed stream is
'011000011001100011000110001000001100100'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 2
symbol 'b' with code 0001 and count 1
symbol 'c' with code 001 and count 1
symbol 'd' with code 01 and count 1
NYT node with code 0000 and count 0
NYT: 0000 (weight: 0)
'b': 0001 (weight: 1)
'c': 001 (weight: 1)
'd': 01 (weight: 1)
'a': 1 (weight: 2)

After encoding the character 'a' (character #6), the compressed stream is
'0110000110011000110001100010000011001001'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 3
symbol 'b' with code 0001 and count 1
symbol 'c' with code 001 and count 1
symbol 'd' with code 01 and count 1
NYT node with code 0000 and count 0
NYT: 0000 (weight: 0)
'b': 0001 (weight: 1)
'c': 001 (weight: 1)
'd': 01 (weight: 1)
'a': 1 (weight: 3)

After encoding the character 'd' (character #7), the compressed stream is
'011000011001100011000110001000001100100101'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 3
symbol 'b' with code 0001 and count 1
symbol 'c' with code 001 and count 1
symbol 'd' with code 01 and count 2
NYT node with code 0000 and count 0
NYT: 0000 (weight: 0)
'b': 0001 (weight: 1)

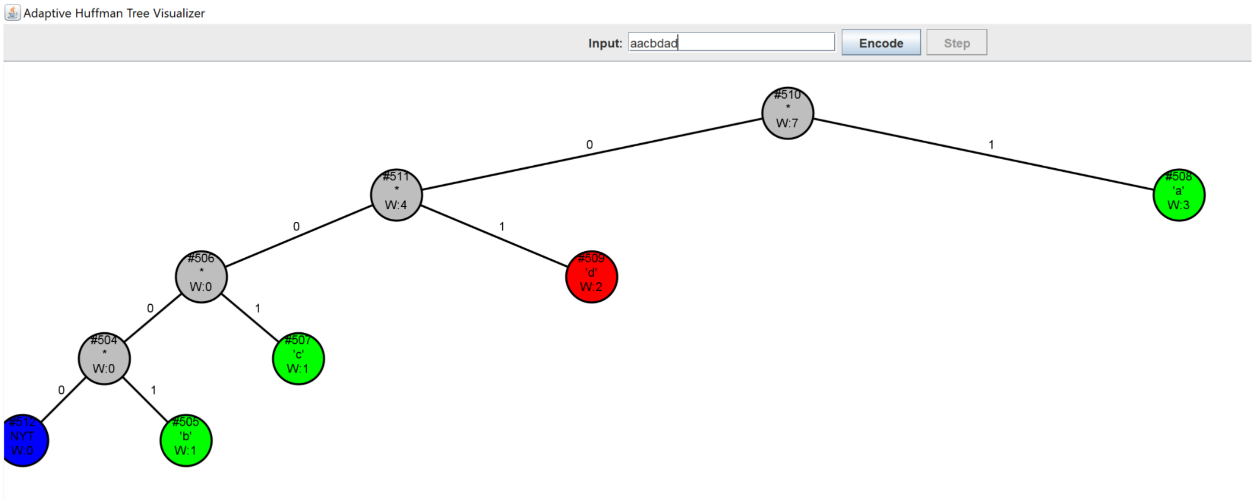
```
'c': 001 (weight: 1)
'd': 01 (weight: 2)
'a': 1 (weight: 3)


Final compressed stream: 0110000110011000110001100010000001100100101

=== DECODING PROCESS ===

After decoding character #1 ('a'), the current decoded message is 'a'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 1
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 1)


After decoding character #2 ('a'), the current decoded message is 'aa'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 2
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 2)


After decoding character #3 ('c'), the current decoded message is 'aac'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 2
symbol 'c' with code 01 and count 1
NYT node with code 00 and count 0
NYT: 00 (weight: 0)
'c': 01 (weight: 1)
'a': 1 (weight: 2)


After decoding character #4 ('b'), the current decoded message is 'aacb'The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 2
symbol 'b' with code 01 and count 1
symbol 'c' with code 001 and count 1
```

```
NYT node with code 000 and count 0
NYT: 000 (weight: 0)
'c': 001 (weight: 1)
'b': 01 (weight: 1)
'a': 1 (weight: 2)


After decoding character #5 ('d'), the current decoded message is 'aacbd'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 2
symbol 'b' with code 0001 and count 1
symbol 'c' with code 001 and count 1
symbol 'd' with code 01 and count 1
NYT node with code 0000 and count 0
NYT: 0000 (weight: 0)
'b': 0001 (weight: 1)
'c': 001 (weight: 1)
'd': 01 (weight: 1)
'a': 1 (weight: 2)


After decoding character #6 ('a'), the current decoded message is 'aacbda'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 3
symbol 'b' with code 0001 and count 1
symbol 'c' with code 001 and count 1
symbol 'd' with code 01 and count 1
NYT node with code 0000 and count 0
NYT: 0000 (weight: 0)
'b': 0001 (weight: 1)
'c': 001 (weight: 1)
'd': 01 (weight: 1)
'a': 1 (weight: 3)


After decoding character #7 ('d'), the current decoded message is 'aacbdad'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 3
symbol 'b' with code 0001 and count 1
symbol 'c' with code 001 and count 1
symbol 'd' with code 01 and count 2
NYT node with code 0000 and count 0
NYT: 0000 (weight: 0)
'b': 0001 (weight: 1)
'c': 001 (weight: 1)
'd': 01 (weight: 2)
'a': 1 (weight: 3)


Final decoded message: aacbdad
```

Results:

- **Original**: 56 bits (7 chars × 8 bits)
- **Compressed**: 43 bits
- **Compression Ratio**: 23.2% reduction
- **Decoded Output**: aacbdad (matches input)

Visualization:



3.2 Test Case 2: "aaaaaa"

Input: aaaaaa

OUTPUT STEPS:

```
Original message: aaaaaa
=== ENCODING PROCESS ===

After encoding the character 'a' (character #1), the compressed stream is '01100001'
The tree contains mainly the following nodes:
```

```
symbol 'a' with code 1 and count 1
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 1)


After encoding the character 'a' (character #2), the compressed stream is '011000011'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 2
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 2)


After encoding the character 'a' (character #3), the compressed stream is '0110000111'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 3
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 3)


After encoding the character 'a' (character #4), the compressed stream is '01100001111'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 4
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 4)


After encoding the character 'a' (character #5), the compressed stream is '011000011111'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 5
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 5)


After encoding the character 'a' (character #6), the compressed stream is '0110000111111'
The tree contains mainly the following nodes:
```

```
symbol 'a' with code 1 and count 6
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 6)


Final compressed stream: 0110000111111

=== DECODING PROCESS ===

After decoding character #1 ('a'), the current decoded message is 'a'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 1
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 1)


After decoding character #2 ('a'), the current decoded message is 'aa'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 2
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 2)


After decoding character #3 ('a'), the current decoded message is 'aaa'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 3
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 3)


After decoding character #4 ('a'), the current decoded message is 'aaaa'The tree contains mainly
the following nodes:
symbol 'a' with code 1 and count 4
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 4)
```

After decoding character #5 ('a'), the current decoded message is 'aaaaa'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 5
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 5)

After decoding character #6 ('a'), the current decoded message is 'aaaaaa'
The tree contains mainly the following nodes:
symbol 'a' with code 1 and count 6
NYT node with code 0 and count 0
NYT: 0 (weight: 0)
'a': 1 (weight: 6)

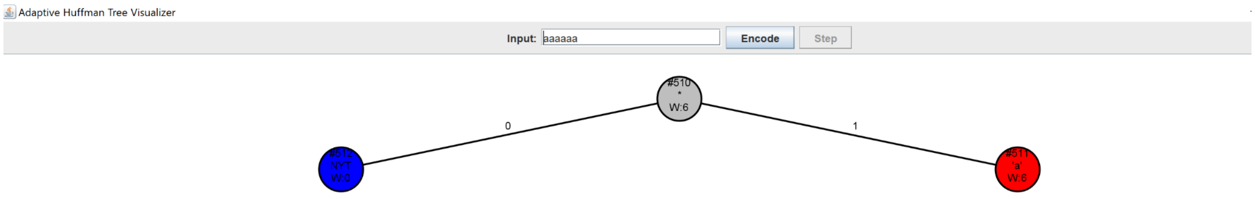
Final decoded message: aaaaaa

Verification: SUCCESS

Results:

- Original String: aaaaaa (6 identical characters)
- ASCII Representation: a (97) repeated 6 times
- Uncompressed Size: 6 chars × 8 bits = 48 bits
- Compression Ratio: 72.9% reduction
- Decoded Output: aaaaaa

Visualization:



Input: aaaaaa

4. Technical Analysis

4.1 Optimizations

1. **Node Swapping:**
   - Ensures sibling property via swapNodes()
   - Example: When 'a' reaches weight=3, it swaps with higher-numbered nodes.
2. **Efficient Encoding:**
   - First 'a': 9 bits (NYT+ASCII)
   - Subsequent 'a's: 1 bit
3. **Memory Management:**
   - Node numbers (nodeNumber) track creation order for swapping.

Appendix

A. Source Code

See attached .java files.

B. Sample Output

```
Enter a message: aacbdad
Encoded: 0110000110011000110001100010000011001001001 (43 bits)
Decoded: aacbdad
```

C. How to Run

```
javac --module-path [javafx-lib] --add-modules javafx.controls,javafx.fxml src/*.java
visualization/*.java
java --module-path [javafx-lib] --add-modules javafx.controls,javafx.fxml -cp out
src.AdaptiveHuffman
```