

## Mini Project

### Implementation of Adaptive Huffman (Spring 2025)

### Project Objectives

1. Implement Adaptive Huffman Coding using java.
2. Develop an efficient system for **encoding and decoding** text dynamically.
3. Manage symbols effectively using a **binary tree structure** that adjusts in real time.
4. (Bonus) **Integrate a visualization module** to display tree updates during encoding/decoding.

### Submission Guidelines

- Each student does the project individually.
- The project will be graded out of 10 marks.
- Due date: Saturday, March 22 [11:59 PM].
- For every day of delay in delivery, 0.75 mark will be deducted.
- The project must be submitted through the course classroom before the due date. Then, it will be discussed and graded by your TA(s) in your Lab.
- The project submission must include the following:
  - '.java' file(s) that contains the code.
  - A report containing a copy of the code accompanied by two test cases.

---

### Evaluation Criteria (10 Points + 2 Bonus)

- **(8 points) Algorithm Correctness & Efficiency:** Ensures accurate and complete implementation of the encoding/decoding functions and proper tree updates. Five points for the encoding and three points for the decoding.
- **(1 point) Project Testing:** Ensures that the project is working correctly using test cases.
- **(1 point) Report:** Assesses the documentation and analysis.
- **(+2 bonus) Visualization Module:** Evaluates clarity and effectiveness of the tree update animations.

# Project Structure

## 1. Core Implementation

adaptive-huffman-coding/

— src/	
— AdaptiveHuffman.java	# Main driver class for encoding/decoding
— Node.java	# Data structure representing tree nodes
— HuffmanTree.java	# Handles tree construction and updates
— Encoder.java	# Implements the encoding process
— Decoder.java	# Implements the decoding process
— test/	
— AdaptiveHuffmanTest.java	# Unit tests for verification
— visualization/ (Bonus Part)	
— HuffmanTreeVisualizer.java	# JavaFX/Swing visualization module
— report.pdf	# Documentation and analysis

## Implementation Details

### 1. Node Representation (Node.java)

- Represents individual nodes in the Huffman tree.
- Contains:
  - Character data (for leaf nodes)
  - Frequency count
  - Left & right child references
  - Parent reference
  - Node number & order

### 2. Huffman Tree Operations (HuffmanTree.java)

- Manages the Huffman tree dynamically during encoding/decoding.
- Key operations:
  - **Inserting new symbols** and handling the NYT (Not Yet Transmitted) node.
  - **Rebalancing/adjusting the nodes counts** based on symbols frequency.
  - **Swapping nodes** to maintain Huffman tree properties.

### 3. Encoding Process (`Encoder.java`)

- Steps:
  - Retrieve binary codes for characters from the Huffman tree.
  - Append the NYT code followed by the ASCII representation for new characters.
  - Update the tree dynamically with frequency adjustments.

### 4. Decoding Process (`Decoder.java`)

- Steps:
    - Traverse the Huffman tree using the encoded binary stream.
    - Reconstruct and output the original text as the binary stream is processed.
- 

### Bonus: Visualization Feature (`HuffmanTreeVisualizer.java`)

- Uses **JavaFX/Swing** to render a real-time visualization of the Huffman tree.
- Key features:
  - Dynamic updates as encoding progresses.
  - Highlights node swaps and frequency adjustments.
  - Smooth animations for better conceptual understanding.