Portfolio: PAQJP_3 Compression System

Project Name: PAQJP_3 Compression System

Author: Jurijus Pacalovas

Technologies: Python, Compression Algorithms (PAQ, zlib, Huffman), File I/O, Datetime Encoding, Logging, Object-Oriented Programming

Overview

The **PAQJP_3 Compression System** is a sophisticated, custom-built file compression and decompression tool that integrates multiple compression algorithms (PAQ, zlib, and Huffman coding) with advanced data transformation techniques. Designed to optimize file size while preserving data integrity, the system incorporates a unique datetime encoding mechanism to embed metadata and employs modular transformations for enhanced compression efficiency. The project demonstrates expertise in algorithm design, file handling, and robust error management.

Key Features

- **Hybrid Compression**: Combines PAQ, zlib, and Huffman coding, selecting the best method based on file size and content.

- **Data Transformations**: Applies reversible transformations (e.g., prime-based XOR, bit rotation, pattern chunking) to preprocess data for better compression ratios.

- **Datetime Encoding**: Embeds a 8-byte datetime stamp in compressed files, enabling metadata tracking (e.g., compression timestamp).

- **Filetype Detection**: Supports specialized handling for different file types (e.g., JPEG, EXE, text) to optimize compression.

- **Robust Error Handling**: Implements comprehensive logging and error checking to ensure reliability.

- **Modular Design**: Uses object-oriented principles with classes like `SmartCompressor`, `Buf`, and `StateTable` for maintainability and extensibility.

- **Cross-Platform Compatibility**: Written in Python, ensuring compatibility across Windows, macOS, and Linux.

Technical Highlights

- **Compression Algorithms**:

  - **PAQ**: A high-performance context-mixing compressor for maximum compression ratios.

  - **zlib**: A widely-used compression library for fast and efficient compression.

  - **Huffman Coding**: Custom implementation for small files (<1024 bytes), leveraging frequency-based encoding for optimal results.

- **Transformations**:

  - **Prime XOR (transform_01)**: Applies XOR operations with prime numbers every 3 bytes for data randomization.

  - **Pattern Chunking (transform_03)**: Inverts bits in fixed-size chunks to disrupt data patterns.

  - **Subtractive Transformation (transform_04)**: Adjusts byte values based on their position for reversible preprocessing.

  - **Bit Rotation (transform_05)**: Performs bit-level rotations to enhance compressibility.

  - **Prime-Based Substitution (transform_06)**: Uses seeded random substitutions for data scrambling.

- **Datetime Encoding**: Encodes date and time into a compact 8-byte format, capturing seconds, minutes, hours, day of week, day of year, month, day of month, and year (up to 4095).

- **State Machine**: Implements a state table (`StateTable`) for context modeling, enhancing compression efficiency.

- **Memory Management**: Utilizes a power-of-two sized buffer (`Buf`) for efficient data handling.

Code Structure

- **Main Components**:

  - `SmartCompressor`: Core class managing compression and decompression workflows, including algorithm selection and transformation application.

  - `Buf`: Circular buffer for efficient data access during compression.

  - `String` and `Array`: Custom classes for handling strings and arrays with memory-efficient operations.

  - `StateTable`: Defines state transitions for context modeling.

  - `Node`: Represents nodes in the Huffman tree for encoding/decoding.

- **Key Functions**:

  - `encode_datetime` / `decode_datetime`: Handle datetime metadata encoding and decoding.

  - `compress_with_best_method`: Selects the optimal compression method and transformation based on data size and content.

  - `decompress_with_best_method`: Reverses compression using embedded markers and datetime metadata.

  - Transformation functions (`transform_01` to `transform_06`): Apply reversible data preprocessing.

- **Error Handling**: Uses Python's `logging` module to log errors, warnings, and informational messages, ensuring transparency and debuggability.

Achievements

- **Compression Efficiency**: Achieved significant file size reductions, with compression ratios as low as 30–50% for text-heavy files (based on test data).

- **Robustness**: Successfully handles edge cases like empty files, invalid inputs, and corrupted compressed data.

- **Extensibility**: Modular design allows easy addition of new compression algorithms or transformations.

- **Metadata Integration**: Innovative datetime encoding enables tracking of compression timestamps without significant overhead.

Challenges Overcome

- **Algorithm Selection**: Designed a heuristic to dynamically choose between PAQ, zlib, and Huffman based on file size and content, balancing speed and compression ratio.

- **Reversible Transformations**: Ensured all transformations are mathematically reversible to guarantee lossless compression.

- **Datetime Encoding**: Developed a compact 8-byte format to encode comprehensive datetime information, handling edge cases like leap years and invalid dates.

- **Performance Optimization**: Optimized memory usage with power-of-two buffers and efficient array operations to handle large files.