### Project created by Jurijus Pacalovas: Advanced Compression System with Smart and PAQJP_6 Compressors

#### Project Overview

This project develops a hybrid compression system combining two distinct algorithms: the **Smart Compressor** (marked with `[x00]`) and the **PAQJP_6 Compressor** (marked with `[x01]`). The system leverages reversible transformations, including a custom `transform_09` algorithm influenced by pi digits, and integrates Huffman coding for small data sets. A translation dictionary maps transformation markers (1-255) to specific algorithms, ensuring flexibility and reversibility. The goal is to achieve lossless compression, preserving all original data during compression and decompression.

#### 1. Compression Algorithms

- **Smart Compressor ([x00])**:
- Uses SHA-256 hashing for integrity verification.
- Applies a reversible XOR transform with `0xAA`.
- Compresses data using PAQ9a if the compressed size is smaller than the original.
- Ideal for general-purpose files with dictionary-based optimization.

- **PAQJP_6 Compressor ([x01])**:
- Implements multiple transformation methods (1-255), with `transform_09` as a key feature.
- Incorporates pi digits (10,000 generated), last 3 digits for "circle size" influence, and a subtraction operation \(0 - (2^0 - 2^{27} \cdot 1024 \cdot 1024)\) adjusted modulo 256. - Uses PAQ9a compression and Huffman coding for small data (<1024 bytes).
- Optimized for JPEG and text files with prioritized transforms.

- **Marker System**:
- `[x00]` indicates Smart Compressor output.
- `[x01]` indicates PAQJP_6 Compressor output.
- The combined compressor selects the best method based on compressed size.

#### 2. Translation Dictionary (Markers 1-255)

The PAQJP_6 Compressor uses a dynamic set of transformation methods, each assigned a unique marker from 1 to 255. Below is a translation dictionary explaining these transformations:

Marker	Transformation Description	Reverse Method
1	Subtract index modulo 256	Add index modulo 256
2	XOR every 3rd byte with prime-derived value	Same as forward
3	XOR each 4-byte chunk with 0xFF	Same as forward
4	Huffman coding for small data	Huffman decoding
5	Rotate bytes left by 3 bits	Rotate bytes right by 3 bits
6	Random substitution table	Reverse substitution table
7	XOR with pi digits and size byte	Reverse XOR with pi digits
8	XOR with nearest prime and pi digits	Reverse XOR with pi digits
9	XOR with prime, seed, pi digits, add 4 bytes, subtract adjusted value	Reverse all steps
10	XOR with value from 'X1' sequence count	Reverse XOR
11	Test multiple y values for best compression	Reverse with best y
12	XOR with Fibonacci sequence	Reverse XOR
13-255	Dynamic XOR with size modulus and index	Reverse dynamic XOR

- **Purpose**: This dictionary allows the system to select and reverse the appropriate transformation based on the marker, ensuring compatibility and reversibility.

- **Implementation**: The `compress_with_best_method` and `decompress_with_best_method` functions use this mapping to apply and undo transformations.

#### 3. Huffman Coding Explanation

- **Concept**: Huffman coding is a variable-length prefix code that assigns shorter codes to frequently occurring symbols and longer codes to less frequent ones, optimizing compression for small data sets (<1024 bytes).

- **Implementation**:

- `calculate_frequencies`: Computes the frequency of each bit in the binary string.

- `build_huffman_tree`: Constructs a binary tree based on frequencies using a min-heap.

- `generate_huffman_codes`: Assigns binary codes (0s and 1s) to symbols, traversing the tree.

- `compress_data_huffman` and `decompress_data_huffman`: Encode and decode the data, respectively.

- **Lossless Nature**: Huffman coding is inherently lossless because the decoding process reconstructs the original data by following the tree structure, preserving all information.

#### 4. Lossless Compression Explanation

- **Definition**: Lossless compression reduces data size without losing any information, allowing perfect reconstruction of the original data.

- **Mechanisms in This Project**:

- **Reversible Transformations**: All transformations (e.g., XOR, subtraction modulo 256, rotation) have corresponding reverse operations that undo the changes exactly.

- **PAQ9a Compression**: A context-mixing algorithm known for lossless compression, preserving data integrity.

- **Huffman Coding**: Ensures lossless encoding for small data by maintaining a one-toone mapping between codes and symbols.

- **Integrity Check**: The Smart Compressor uses SHA-256 to verify that decompressed data matches the original.

- **Marker System**: Tracks the compression method, ensuring the correct reverse transform is applied.

- **Example - transform_09**:

- Forward: Applies XOR with pi digits, primes, and seeds, adds 4 bytes, and subtracts an adjustment (e.g., 127) modulo 256.

- Reverse: Undoes the subtraction, removes 4 bytes, and reverses XOR operations in the opposite order.

- The process is reversible because each step has an inverse, and the modulo 256 operation ensures byte values remain within 0-255.

#### 5. Conclusion

- **Lossless Verification**: The compression system is lossless because all transformations and compression algorithms (PAQ9a and Huffman) are designed to be reversible. The use of markers `[x00]` and `[x01]`, along with the 1-255 dictionary, ensures the correct decompression path, preventing data loss. Testing with sample files (e.g., text or JPEG) confirms that decompressed output matches the original byte-for-byte.

- **Pi Digit Influence**: While `transform_09` uses pi digits to modulate the data (simulating "circle size" change), this does not introduce loss, as the operation is reversible. However, if a visual circle size change is desired, it would require additional image processing outside the current data compression scope.

- **Base 256 Compliance**: All data is handled as bytes (0-255), naturally aligning with base 256, and saved in binary format, meeting the requirement.

- **Future Improvements**: Adding visual circle size adjustment (e.g., via a canvas or image library) could enhance the project's utility for image processing, while maintaining losslessness.

This project successfully integrates Smart and PAQJP_6 compression, a flexible transformation dictionary, and Huffman coding into a robust, lossless compression framework, suitable for a variety of file types.