Quantum PAQJP_6 Compression System Overview

Already working and lossless

The **PAQJP_6 Compression System**, created by Jurijus Pacalovas, is an advanced, lossless file compression and decompression tool that integrates multiple transformation techniques, base-10 pi digits, and a combination of compression methods (Huffman coding, zlib, and PAQ). It is designed to optimize compression ratios by preprocessing data with transformations, incorporating file creation time, and supporting various file types (JPEG, TEXT, DEFAULT). Below is a detailed description of the project, its transformations, compression methods, and a conclusion on its effectiveness and potential applications.

Transformations (1-11)

PAQJP_6 employs 11 unique transformations to preprocess data, aiming to enhance compressibility. Each transformation is reversible to ensure lossless operation. Here's a summary of each:

1. **Transform_01**: XORs every third byte with prime numbers from a predefined list (`PRIMES`), repeated 100 times by default.
   - **Reverse**: Identical to the forward transform due to XOR's involutive property.

2. **Transform_03**: Inverts bits in chunks (default size 4) by XORing with 0xFF.
   - **Reverse**: Same as forward, as XOR with 0xFF is self-inverse.

3. **Transform_04**: Subtracts the byte index modulo 256 from each byte, repeated multiple times.
   - **Reverse**: Adds the byte index modulo 256, repeated the same number of times.

4. **Transform_05**: Applies a left circular shift by 3 bits (default) to each byte.

   - **Reverse**: Applies a right circular shift by the same number of bits.

5. **Transform_06**: Uses a substitution cipher with a shuffled table based on a seed (default 42).

   - **Reverse**: Applies the inverse substitution table.

6. **Transform_07**: XORs each byte with a size-dependent byte and base-256 pi digits, with cycles based on data size.

   - **Reverse**: Reverses XOR operations and adjusts pi digit alignment.

7. **Transform_08**: Similar to Transform_07, but uses a size-dependent prime number for XOR.

   - **Reverse**: Reverses XOR with the same prime and pi digits.

8. **Transform_09**: Extends Transform_08 by adding XOR with a seed table value for increased randomization.

   - **Reverse**: Reverses XOR with seed table values and pi digits.

9. **Transform_10**: Computes a value `n` based on the count of “X1” sequences (0x58, 0x31), XORs each byte with `n`, and prepends `n` to the output.

   - **Reverse**: Extracts `n` from the first byte and reverses the XOR.

10. **Transform_11**: Tests values of `y` (1 to 255), adding `(y+1)` modulo 256 to each byte, selecting the `y` yielding the smallest PAQ-compressed output. Prepends `y` to the output.

    - **Reverse**: Extracts `y` and subtracts `(y+1)` modulo 256.

Compression Methods

The system uses three compression methods to achieve optimal results:

1. **Huffman Coding**:

   - Applied to files smaller than `HUFFMAN_THRESHOLD` (1024 bytes).

   - Converts data to a binary string, builds a Huffman tree based on bit frequencies, and encodes using variable-length codes.

   - Decompression decodes the binary string back to the original data.

   - Marker: `4`.

2. **zlib Compression**:

   - Uses Python's `zlib` library for DEFLATE compression.

   - Applied after each transformation to evaluate compression efficiency.

   - Decompression uses `zlib.decompress`.

3. **PAQ Compression**:

   - Employs the `paq` module, likely a custom or optimized implementation for high-efficiency compression.

   - Often yields better results for larger files or specific transformations.

   - Decompression uses `paq.decompress`.

Best Method Selection

- **Compression**: Tests each transformation with PAQ and zlib, and Huffman for small files. Selects the combination with the smallest output size, prepending a marker byte (1-11) and 6 bytes of encoded file creation time.

- **Decompression**: Reads the marker to identify the transformation, decodes the datetime, decompresses using the appropriate method (PAQ, zlib, or Huffman), and applies the reverse transformation.

Base-10 Pi Digits

- Uses 3 base-10 digits of pi (3.14), mapped to 0-255, in transformations 7, 8, and 9.

- Digits are loaded from `pi_digits.txt` or generated using `mpmath` and saved for reuse.

- Adds pseudo-randomness to transformations, potentially improving compression for certain data patterns.

Filetype Handling

- Detects file types by extension: `.jpg`/`.jpeg` → JPEG, `.txt` → TEXT, others → DEFAULT.

- Prioritizes transformations 7-11 for JPEG and TEXT files, as they may be more effective for these formats.

Datetime Integration

- Encodes the input file's creation time into 6 bytes, included in the compressed output.

- During decompression, restores the datetime and sets it as the output file's creation/modification time.

Usage

- **Compress (Option 1)**: Reads input file, applies transformations, compresses with the best method, and saves output with marker and datetime.

- **Decompress (Option 2)**: Reads compressed file, identifies transformation via marker, decompresses, reverses transformation, and restores file with original datetime.

Logging and Error Handling

- Logs transformation cycles, compression sizes, zero-byte counts, and errors.

- Handles edge cases (e.g., empty files, invalid datetime, compression failures) robustly.

Conclusion

The **PAQJP_6 Compression System** is a highly innovative and versatile compression tool that leverages a combination of mathematical transformations, pi-based randomization, and multiple compression algorithms to achieve efficient, lossless compression. Its use of base-10 pi digits in transformations 7-9 adds a unique layer of pseudo-randomness, potentially enhancing compressibility for specific data patterns. The system's ability to test multiple transformations and compression methods (Huffman, zlib, PAQ) ensures optimal performance across different file types and sizes. The prioritization of certain transformations for JPEG and TEXT files demonstrates its adaptability to specific data characteristics.

Strengths:

- **Flexibility**: Supports multiple file types with tailored transformations.

- **Robustness**: Comprehensive error handling and logging ensure reliability.

- **Innovation**: Integration of pi digits and prime numbers adds a novel approach to data transformation.

- **Efficiency**: Selects the best compression method, often achieving competitive compression ratios.

Limitations:

- **Complexity**: Multiple transformations and compression methods increase computational overhead, especially for Transform_11, which tests 255 values of `y`.

- **Dependency**: Relies on the `paq` module, which may not be standard or widely available.

- **Small File Bias**: Huffman coding is limited to files under 1024 bytes, which may not always be optimal for slightly larger files.

Potential Applications:

- Archival of multimedia files (e.g., JPEG, TEXT) where lossless compression is critical.

- Data transfer in bandwidth-constrained environments, leveraging its high compression ratios.

- Research into novel compression techniques, particularly those involving mathematical constants like pi.