LZ77 Compression Algorithm

Mohamed darwish

202201273

Abstract

This report presents an improved implementation of the LZ77 compression algorithm, a sliding-window, dictionary-based technique for lossless data compression. By identifying and encoding repeated sequences using backward references, LZ77 achieves significant file size reduction. The implementation focuses on modular design, clarity, and performance in both compression and decompression tasks.

Keywords: LZ77, lossless compression, dictionary-based algorithm, sliding window, text file optimization

---

I. Overview

LZ77 is a foundational compression algorithm that substitutes recurring data with references to prior occurrences in a sliding window buffer. This report details a modular and scalable implementation of LZ77 tailored for compressing and decompressing textual data efficiently.

II. Project Structure

The project is organized into several independent modules, facilitating maintainability and testing:

```
LZ77Main
├── FileHandler
├── LZ77Encoder
├── LZ77Decoder
└── CompressedFile
```

III. Core Components

A. LZ77Encoder (Compression)

Responsible for transforming raw input text into LZ77-encoded triplets. The encoder follows these steps:

- Iterates through the input stream.
- Searches the sliding window for the longest match.
- Emits encoded tokens in the form `(offset, length, next_char)`.

B. LZ77Decoder (Decompression)

Restores the original text from the compressed form by:

- Reading each `(offset, length, next_char)` triplet.
- Reconstructing the referenced data segment from previously decoded output.
- Appending the result to the decompressed stream.

IV. Compression and Decompression Workflow

A. Compression Workflow

1. Load raw text from input.
2. Initialize the sliding window.
3. For each index:
    - Identify the longest backward match.
    - Emit corresponding token.
4. Output the token stream to a file.

B. Decompression Workflow

1. Read compressed tokens from file.
2. Reconstruct text using references and characters.
3. Write decompressed data to the output file.

C. Token Format

Each LZ77 token is structured as:

```
(offset: int, length: int, next_char: char)
```

These triplets allow precise reconstruction of the original input using window-based referencing.

V. Conclusion

This LZ77 implementation provides an efficient, modular solution for compressing and decompressing text files. The design balances simplicity and performance, with clear separation between encoding, decoding, and file handling. Future optimizations may explore advanced matching algorithms and dynamic window sizing for large-scale data processing.