LZ78 Compression Algorithm

Mohamed darwish

202201273

Abstract

This report describes the implementation of the LZ78 compression algorithm, a dictionary-based technique that incrementally builds a database of previously encountered patterns to reduce redundancy in textual data. The implementation emphasizes modularity, maintainability, and efficient encoding/decoding of input files.

Keywords: LZ78, data compression, dictionary-based encoding, lossless compression, modular software design

---

I. Overview

LZ78 is a lossless data compression algorithm that encodes repeating patterns using a dynamically constructed dictionary. Each new pattern is stored and referenced using integer indices, significantly reducing file size when patterns reoccur. This report details a clean and extensible implementation of LZ78 for text file processing.

II. Project Structure

The project follows a modular architecture with separation of concerns between components:

```
LZ78Main
├── FileHandler
├── LZ78Encoder
├── LZ78Decoder
└── CompressedFile
```

III. Core Components

A. LZ78Encoder (Compression)

Encodes the input text into a stream of LZ78 tags by:

- Scanning the input text one character at a time.
- Searching for the longest match in the dictionary.
- Emitting a tag `(index, next_char)` and updating the dictionary with the new pattern.

B. LZ78Decoder (Decompression)

Reverses the compression process by:

- Reading each tag `(index, next_char)` from the compressed data.
- Using the dictionary to reconstruct previously encoded segments.
- Appending new patterns as needed to ensure consistent dictionary growth.

IV. Compression and Decompression Workflow

A. Compression Workflow

1. Read input from the source file.
2. Initialize an empty dictionary.
3. While input remains:
    - Find the longest prefix match.
    - Output tag `(position, next_char)`.
    - Update dictionary with the new entry.
4. Write the output tags to the compressed file.

B. Decompression Workflow

1. Read the tag sequence from the compressed file.
2. Initialize an empty dictionary.
3. For each tag:
    - Lookup prefix using the position.
    - Append the next character.
    - Add the new string to the dictionary.
4. Write the decompressed content to the output file.

C. Tag Format

LZ78 tags are structured as:

```
(position: int, next_char: char)
```

This allows references to earlier entries without explicit substring offsets, enabling concise storage.

V. Conclusion

The implemented LZ78 algorithm offers a compact and extensible approach to text compression using dictionary-based references. The clean module boundaries and efficient tag generation enable it to scale for real-world datasets. Further improvements could include trie-based dictionary management and parallel encoding techniques for high-throughput applications.