The Lempel Ziv Algorithm

Seminar “Famous Algorithms”
January 16, 2003

christina.zeeh@stud.i.informatik.uni-stuttgart.de
The (?) Lempel Ziv Algorithm

Applications:
- zip
- gzip
- Stacker
- ...

Applications:
- GIF
- V.42
- compress
- ...
Overview

- Introduction
- Lossless Compression
- Dictionary Coding
- LZ77
  - Algorithm
  - Modifications
  - Comparison
- LZ78
  - Algorithm
  - Modifications
  - Comparison
The eldest of these, and Bilbo’s favourite, was young Frodo Baggins. When Bilbo was ninety-nine he adopted Frodo as his heir, and brought him to live at Bag End; and the hopes of the Sackville-Bagginses were finally dashed. Bilbo and Frodo happened to have the same birthday, September 22nd. ‘You had better come and live here, Frodo my lad,’ said Bilbo one day; ‘and then we can celebrate our birthday-parties comfortably together.’ At that time Frodo was still in his tweens, as the hobbits called the irresponsible twenties between childhood and coming of age at thirty-three.

• Data shows patterns, constraints, ...
• Compression algorithms exploit those characteristics to reduce size
Lossless Compression

Lossless compression guarantees that the original information can be exactly reproduced from the compressed data.

- Run-length coding
- Statistical methods
  - Huffman coding
  - Arithmetic coding
  - PPM
- Dictionary methods
  - Lempel Ziv algorithms
<table>
<thead>
<tr>
<th>Dictionary Coding (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Observation: Correlations between parts of the data (patterns)</td>
</tr>
<tr>
<td>• Idea: Replace recurring patterns with references to a dictionary</td>
</tr>
<tr>
<td>• Static, semi-adaptive, adaptive</td>
</tr>
<tr>
<td>• LZ algorithms use adaptive approach</td>
</tr>
<tr>
<td>✓ coding scheme is universal</td>
</tr>
<tr>
<td>✓ no need to transmit/store dictionary</td>
</tr>
<tr>
<td>✓ single-pass (dictionary creation “on-the-fly”)</td>
</tr>
</tbody>
</table>
The eldest of these, and Bilbo’s favourite, was young Frodo Baggins. When Bilbo was ninety-nine he adopted Frodo as his heir, and brought him to live at Bag End; and the hopes of the Sackville-Bagginses were finally dashed. Bilbo and Frodo ...
Dictionary Coding (3)

- Use previously processed data as dictionary (LZ77 algorithm)

The eldest of these, and Bilbo’s favourite, was young Frodo Baggins. When Bilbo was ninety-nine he adopted Frodo as his heir, and brought him to live at Bag End; and the hopes of the Sackville-Bagginses were finally dashed. Bilbo and Frodo ...
LZ77 (1)

- Memory / speed constraints require restrictions
  ⇒ use a fixed-size window (“sliding window” principle)
while (lookAheadBuffer not empty) {
    get a reference (position, length) to longest match;
    if (length > 0) {
        output (position, length, next symbol);
        shift the window length+1 positions along;
    } else {
        output (0, 0, first symbol in lookahead buffer);
        shift the window 1 position along;
    }
}
LZ77 Example

\[ S = 0 0 1 0 1 0 2 1 0 2 1 0 2 1 2 0 2 1 0 2 1 2 0 0 \ldots \]

\[ a = 3 \quad (size \ of \ alphabet) \]

\[ L_s = 9 \quad (lookahead \ buffer \ size) \]

\[ n = 18 \quad (window \ size) \]

Codeword length:

\[ L_c = 1 + \log_a(n - L_s) + \log_a(L_s) \]

\[ = 1 + \log_3(9) + \log_3(9) \]

\[ = 5 \]
LZ77 Example – Encoder

1. 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 2 1 0 2 1 0 2 1 \ldots
   \quad C_1 = 22 02 1

2. 0 0 0 0 0 0 0 0 0 0 1 0 1 0 2 1 0 2 1 0 2 1 \ldots
   \quad C_2 = 21 11 2

3. 0 0 0 0 1 0 1 0 2 1 0 2 1 0 2 1 2 0 2 1 2 0 2 1 \ldots
   \quad C_3 = 20 21 2

4. 2 1 0 2 1 0 2 1 2 0 2 1 0 2 1 2 0 0 \ldots
   \quad C_4 = 02 22 0
## LZ77 Example – Decoder

1. \( C_1 = 22 \ 02 \ 1 \)
   
   \[
   0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1
   \]

2. \( C_2 = 21 \ 11 \ 2 \)
   
   \[
   0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 2
   \]

3. \( C_3 = 20 \ 21 \ 2 \)
   
   \[
   0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 2 \ 1 \ 0 \ 2 \ 1 \ 0 \ 2 \ 1 \ 2
   \]

4. \( C_4 = 02 \ 22 \ 0 \)
   
   \[
   2 \ 1 \ 0 \ 2 \ 1 \ 0 \ 2 \ 1 \ 2 \ 0 \ 2 \ 1 \ 0 \ 2 \ 1 \ 2 \ 0 \ 0
   \]
# LZ77 Improvements

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LZR</strong></td>
<td>references to any point in processed data, variable-length references</td>
</tr>
<tr>
<td><strong>LZSS</strong></td>
<td>codewords without symbol, output (offset, length) or symbol, flag to distinguish</td>
</tr>
<tr>
<td><strong>LZB</strong></td>
<td>increasing pointer size, variable-length matches (no lookahead buffer), min. match length</td>
</tr>
<tr>
<td><strong>LZH</strong></td>
<td>LZSS and Huffman coding (2 passes), Huffman table needs to be stored/transmitted</td>
</tr>
</tbody>
</table>
LZ77 Comparison

All values taken from Bell/Cleary/Witten: Text Compression
* combined result for two test files
LZ78 (1)

- Maintain explicit dictionary
- Gradually build dictionary during encoding
- Codeword consists of 2 elements:
  - index (reference to longest match in dictionary)
  - first non-matching symbol
- Every codeword also becomes new dictionary entry
LZ78 (2)

\[
\begin{align*}
  &w := \text{NIL}; \\
  &\text{while (there is input) } \{ \\
  &\quad K := \text{next symbol from input}; \\
  &\quad \text{if (wK exists in the dictionary) } \{ \\
  &\quad \quad w := wK; \\
  &\quad \} \text{ else } \{ \\
  &\quad \quad \text{output (index}(w), K); \\
  &\quad \quad \text{add wK to the dictionary}; \\
  &\quad \quad w := \text{NIL}; \\
  &\quad \} \\
  &\}
\end{align*}
\]
# LZ78 Example – Encoder

<table>
<thead>
<tr>
<th>#</th>
<th>entry</th>
<th>phrase</th>
<th>Output:</th>
<th>(ternary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0 0</td>
<td>(0 0)</td>
</tr>
<tr>
<td>2</td>
<td>1+1</td>
<td>01</td>
<td>1 1</td>
<td>(1 1)</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0 2</td>
<td>(0 2)</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0 1</td>
<td>(00 1)</td>
</tr>
<tr>
<td>5</td>
<td>3+1</td>
<td>21</td>
<td>3 1</td>
<td>(10 1)</td>
</tr>
<tr>
<td>6</td>
<td>5+0</td>
<td>210</td>
<td>5 0</td>
<td>(12 0)</td>
</tr>
<tr>
<td>7</td>
<td>6+1</td>
<td>2101</td>
<td>6 1</td>
<td>(20 1)</td>
</tr>
<tr>
<td>8</td>
<td>7+2</td>
<td>21012</td>
<td>7 2</td>
<td>(21 2)</td>
</tr>
<tr>
<td>9</td>
<td>7+1</td>
<td>21011</td>
<td>7 1</td>
<td>(21 1)</td>
</tr>
</tbody>
</table>
### LZ78 Example – Decoder

<table>
<thead>
<tr>
<th>Input:</th>
<th>#</th>
<th>entry</th>
<th>phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 ✓</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1 ✓</td>
<td>2</td>
<td>1+1</td>
<td>01</td>
</tr>
<tr>
<td>0 2 ✓</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0 1 ✓</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3 1 ✓</td>
<td>5</td>
<td>3+1</td>
<td>21</td>
</tr>
<tr>
<td>5 0 ✓</td>
<td>6</td>
<td>5+0</td>
<td>210</td>
</tr>
<tr>
<td>6 1 ✓</td>
<td>7</td>
<td>6+1</td>
<td>2101</td>
</tr>
<tr>
<td>7 2 ✓</td>
<td>8</td>
<td>7+2</td>
<td>21012</td>
</tr>
<tr>
<td>7 1 ✓</td>
<td>9</td>
<td>7+1</td>
<td>21011</td>
</tr>
</tbody>
</table>

0 0 1 2 1 2 1 2 1 0 2 1 0 1 2 1 0 1 2 2 1 0 1 1
LZ78 Weaknesses

- Dictionary grows without bound
- Long phrases appear late
- Inclusion of first non-matching symbol may prevent a good match
- Few substrings of the processed input are entered into the dictionary
LZW (1)

- Most popular modification to LZ78
- Algorithm used to compress GIF images
- LZW is patented (like many other LZ algorithms)

- Next symbol no longer included in codeword
  (⇒ dictionary pre-filled with input alphabet)
- More substrings entered into dictionary
- Fixed-length references (12 bit, 4096 entries)
- Static after max. entries reached
LZW (2)

\[ w := \text{NIL}; \]
\[ \text{while (there is input)\{} \]
\[ \quad K := \text{next symbol from input}; \]
\[ \quad \text{if (} wK \text{ exists in the dictionary) \{} \]
\[ \quad \quad w := wK; \]
\[ \quad \} \ \text{else} \ \{ \]
\[ \quad \quad \text{output (index}(w)); \]
\[ \quad \quad \text{add } wK \text{ to the dictionary}; \]
\[ \quad \quad w := K; \]
\[ \} \]
\[ \} \]
## LZ78 Other Improvements

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LZC</td>
<td>variable-length pointers, increasing pointer size, monitor compression ratio</td>
</tr>
<tr>
<td>LZT</td>
<td>LZW + removal of least recently used entries</td>
</tr>
<tr>
<td>LZMW</td>
<td>new entries created by concatenating two last encoded phrases</td>
</tr>
<tr>
<td>LZJ</td>
<td>dictionary contains every unique string of the data up to certain length, delete entries used only once</td>
</tr>
<tr>
<td>LZFG</td>
<td>LZ78 with dictionary storage in a trie and sliding-window principle (remove oldest entries)</td>
</tr>
</tbody>
</table>
LZ78 Comparison

All values taken from Bell/Cleary/Witten: Text Compression

* combined result for two test files
Comparison LZ and Statistical Coding

All values taken from Bell/Cleary/Witten: Text Compression
* combined result for two test files
Questions?