LZW Encoding Algorithm

Repeat
find the longest match w in the dictionary
output the index of w
put wa in the dictionary where a was the unmatched symbol

LZW Encoding Example (1)
Dictionary
0   a
1   b

LZW Encoding Example (2)
Dictionary
0   a
1   b
2   ab

LZW Encoding Example (3)
Dictionary
0   a
1   b
2   ab
3   ba

LZW Encoding Example (4)
Dictionary
0   a
1   b
2   ab
3   ba
4   aba
LZW Encoding Example (5)

Dictionary

<table>
<thead>
<tr>
<th>Index</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>ab</td>
</tr>
<tr>
<td>3</td>
<td>ba</td>
</tr>
<tr>
<td>4</td>
<td>aba</td>
</tr>
<tr>
<td>5</td>
<td>abab</td>
</tr>
</tbody>
</table>

LZW Encoding Example (6)

Dictionary

<table>
<thead>
<tr>
<th>Index</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>ab</td>
</tr>
<tr>
<td>3</td>
<td>ba</td>
</tr>
<tr>
<td>4</td>
<td>aba</td>
</tr>
<tr>
<td>5</td>
<td>abab</td>
</tr>
</tbody>
</table>

LZW Decoding Algorithm

• Emulate the encoder in building the dictionary. Decoder is slightly behind the encoder.

  initialize dictionary;
  decode first index to w;
  put w? in dictionary;
  repeat
    decode the first symbol s of the index;
    complete the previous dictionary entry with s;
    finish decoding the remainder of the index;
    put w? in the dictionary where w was just decoded;

LZW Decoding Example (1)

Dictionary

<table>
<thead>
<tr>
<th>Index</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>ab</td>
</tr>
<tr>
<td>3</td>
<td>b?</td>
</tr>
</tbody>
</table>

LZW Decoding Example (2a)

Dictionary

<table>
<thead>
<tr>
<th>Index</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>ab</td>
</tr>
</tbody>
</table>

LZW Decoding Example (2b)

Dictionary

<table>
<thead>
<tr>
<th>Index</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>ab</td>
</tr>
<tr>
<td>3</td>
<td>b?</td>
</tr>
</tbody>
</table>
LZW Decoding Example (3a)

Dictionary
0 a
1 b
2 ab
3 ba

0 1 2 4 3 6
a b a

LZW Decoding Example (3b)

Dictionary
0 a
1 b
2 ab
3 ba
4 ab?

0 1 2 4 3 6
a b ab

LZW Decoding Example (4a)

Dictionary
0 a
1 b
2 ab
3 ba
4 aba

0 1 2 4 3 6
a b a b a

LZW Decoding Example (4b)

Dictionary
0 a
1 b
2 ab
3 ba
4 aba
5 aba?

0 1 2 4 3 6
a b ab aba

LZW Decoding Example (5a)

Dictionary
0 a
1 b
2 ab
3 ba
4 aba
5 abab

0 1 2 4 3 6
a b a b a b

LZW Decoding Example (5b)

Dictionary
0 a
1 b
2 ab
3 ba
4 aba
5 abab
6 ba?

0 1 2 4 3 6
a b ab aba ba
LZW Decoding Example (5a)

Dictionary

| 0 | a |
| 1 | b |
| 2 | ab |
| 3 | ba |
| 4 | aba |
| 5 | abab |
| 6 | bab |

LZW Decoding Example (5b)

Dictionary

| 0 | a |
| 1 | b |
| 2 | ab |
| 3 | ba |
| 4 | aba |
| 5 | abab |
| 6 | bab |
| 7 | bab? |

Trie Data Structure for Dictionary

- Fredkin (1960)

Encoder Uses a Trie (1)

Decoder’s Data Structure

- Simply an array of strings
Notes on Dictionary Coding

• Extremely effective when there are repeated patterns in the data that are widely spread. Where local context is not as significant.
  – text
  – some graphics
  – program sources or binaries
• Variants of LZW are pervasive.
  – Unix compress
  – GIF

Context-Free Grammars

• Invented by Chomsky in 1959 to explain the grammar of natural languages.
• Also invented by Backus in 1959 to generate and parse Fortran.
• Example:
  – terminals: b, e
  – nonterminals: S, A
  – Production Rules: S → SA, S → A, A → bSe, A → be
  – S is the start symbol

Arithmetic Expressions

• S → S + T
  S → T
  T → T * F
  T → F
  F → a
  F → (S)

Sequitur

• Nevill-Manning and Witten, 1996.
• Uses a context-free grammar (without recursion) to represent a string.
• The grammar is inferred from the string.
• If there is structure and repetition in the string then the grammar may be very small compared to the original string.
• Clever encoding of the grammar yields impressive compression ratios.
• Compression plus structure!

Context-Free Grammar Example

Sequitur Principles

• Digram Uniqueness:
  – no pair of adjacent symbols (digram) appears more than once in the grammar.
• Rule Utility:
  – Every production rule is used more than once.
• These two principles are maintained as an invariant while inferring a grammar for the input string.
Sequitur Example (1)

bbebebebebebee

S → b

Sequitur Example (2)

bbebebebebebee

S → bb

Sequitur Example (3)

bbebebebebebee

S → bbe

Sequitur Example (4)

bbebebebebebee

S → bbeb

Sequitur Example (5)

bbebebebebebee

S → bbebe
Enforce digram uniqueness.
be occurs twice.
Create new rule A → be.

Sequitur Example (6)

bbebebebebebee

S → bAA
A → be
Sequitur Example (7)

\[ bbeebeebbee \]

\[ S \rightarrow bAAe \]
\[ A \rightarrow be \]

Sequitur Example (8)

\[ bbeebeebbee \]

\[ S \rightarrow bAAeb \]
\[ A \rightarrow be \]

Sequitur Example (9)

\[ bbeebeebbee \]

\[ S \rightarrow bAAebe \]
\[ A \rightarrow be \]

Enforce digram uniqueness.

be occurs twice.

Use existing rule \( A \rightarrow be \).
Sequitur Example (13)

S \rightarrow b\text{AA}e\text{AA}
A \rightarrow be

Enforce digram uniqueness
AA occurs twice.
Create new rule B \rightarrow AA.

Sequitur Example (14)

S \rightarrow b\text{BeB}
A \rightarrow be
B \rightarrow AA

Sequitur Example (15)

S \rightarrow b\text{BeB}b
A \rightarrow be
B \rightarrow AA

Sequitur Example (16)

S \rightarrow b\text{BeB}bb
A \rightarrow be
B \rightarrow AA

Sequitur Example (17)

S \rightarrow b\text{BeB}bbe
A \rightarrow be
B \rightarrow AA

Enforce digram uniqueness.
be occurs twice.
Use existing rule A \rightarrow be.

Sequitur Example (18)

S \rightarrow b\text{BeB}ba
A \rightarrow be
B \rightarrow AA
Sequitur Example (19)

S -> bBeBbAb
A -> be
B -> AA

Sequitur Example (20)

S -> bBeBbAbe
A -> be
B -> AA

Enforce digram uniqueness.
be occurs twice.
Use existing rule A -> be.

Sequitur Example (21)

S -> bBeBbAA
A -> be
B -> AA

Enforce digram uniqueness.
AA occurs twice.
Use existing rule B -> AA.

Sequitur Example (22)

S -> bBeBbB
A -> be
B -> AA

Enforce digram uniqueness.
bB occurs twice.
Create new rule C -> bB.

Sequitur Example (23)

S -> CeBC
A -> be
B -> AA
C -> bB

Sequitur Example (24)

S -> CeBCe
A -> be
B -> AA
C -> bB

Enforce digram uniqueness.
Ce occurs twice.
Create new rule D -> Ce.
Sequitur Example (25)

bbebeebebebee

S -> DBD
A -> be
B -> AA
C -> bB
D -> be

Enforce rule utility.
C occurs only once.
Remove C -> bB.

Sequitur Example (26)

bbebeebebebee

S -> DBD
A -> be
B -> AA
D -> bBe

The Hierarchy

bbebeebebebee

S -> DBD
A -> be
B -> AA
D -> bBe

Is there compression? In this small example, probably not.

The Sequitur Algorithm

Input the first symbol s to create the production S -> s;
repeat
match an existing rule:
create a new rule:
remove a rule:
input a new symbol:
until no symbols left

A -> ….XY….
B -> XY
A -> ….B….
B -> XY
A -> ….B….
B -> X
1
X
2
…X
k
A -> ….X….
1
X
2
…X
k
….
S -> X
1
X
2
…X
k
S -> X
1
X
2
…X
k
s

Sequitur Rule Complexity

• digram Uniqueness - match an existing rule.
A -> ….XY...
B -> XY
A -> ….B...
B -> XY
s r s r/2
-1 0 -1

• digram Uniqueness - create a new rule.
A -> ….XY...
B -> ….XY...
A -> ….C...
B -> ….C...
s r s r/2
0 1 -1/2

• Rule Utility - Remove a rule.
A -> ….B...
B -> X
1
X
2
…X
k
A -> ….X
1
X
2
…X
k
….
A -> ….X
1
X
2
…X
k
….
C -> XY
s r s r/2
-1 -1 -1/2

Complexity

• The number of non-input sequitur operations applied ≤ 2n where n is the input length.
• Amortized Complexity Argument
  – Let s = the sum of the right hand sides of all the production rules. Let r = the number of rules.
  – We evaluate s - r/2.
  – Initially s - r/2 = 1/2 because s = 1 and r = 1.
  – s - r/2 ≥ 0 at all times because each rule has at least 1 symbol on the right hand side.
  – s - r/2 increases by 1 for every input operation.
  – s - r/2 decreases by at least 1/2 for each non-input sequitur rule applied.
Linear Time Algorithm

- There is a data structure to implement all the sequitur operations in constant time.
  - Production rules in an array of doubly linked lists.
  - Each production rule has reference count of the number of times used.
  - Each nonterminal points to its production rule.
  - digrams stored in a hash table for quick lookup.

Data Structure Example

```
S -> CeBCe
A -> be
B -> AA
C -> bB
```

Basic Encoding a Grammar

Grammar

```
S -> DBD
A -> be
B -> AA
D -> bBe
```

Grammar Code

```
<table>
<thead>
<tr>
<th>Grammar Code</th>
<th>Øø</th>
<th>ÿø</th>
</tr>
</thead>
<tbody>
<tr>
<td>Øø</td>
<td>101 100 110 000 001 110 011 011 110 000 001 110 001 101 100 101 100 101 100 101 100 101 100 101</td>
<td></td>
</tr>
</tbody>
</table>
```

Better Encoding of the Grammar

- Nevill-Manning and Witten suggest a more efficient encoding of the grammar that resembles LZ77.
  - The first time a nonterminal is sent, its right hand side is transmitted instead.
  - The second time a nonterminal is sent the new production rule is established with a pointer to the previous occurrence sent along with the length of the rule.
  - Subsequently, the nonterminal is represented by the index of the production rule.

Compression Quality

- Neville-Manning and Witten 1997

<table>
<thead>
<tr>
<th>File</th>
<th>size</th>
<th>compress</th>
<th>gzip</th>
<th>sequitur</th>
<th>PPMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>bib</td>
<td>11126</td>
<td>3.35</td>
<td>2.51</td>
<td>2.48</td>
<td>2.12</td>
</tr>
<tr>
<td>book</td>
<td>7687</td>
<td>3.46</td>
<td>3.35</td>
<td>2.82</td>
<td>2.52</td>
</tr>
<tr>
<td>geo</td>
<td>10040</td>
<td>6.08</td>
<td>5.34</td>
<td>4.74</td>
<td>5.01</td>
</tr>
<tr>
<td>obj2</td>
<td>24861</td>
<td>4.17</td>
<td>2.63</td>
<td>2.66</td>
<td>2.77</td>
</tr>
<tr>
<td>pic</td>
<td>51321</td>
<td>0.97</td>
<td>0.82</td>
<td>0.90</td>
<td>0.98</td>
</tr>
<tr>
<td>progc</td>
<td>38611</td>
<td>3.87</td>
<td>2.88</td>
<td>2.83</td>
<td>2.49</td>
</tr>
</tbody>
</table>

Notes on Sequitur

- Very new and different from the standards.
- Yields compression and structure simultaneously.
- With clever encoding is competitive with the best of the standards.
- Practical linear time encoding and decoding.