

15-499: Algorithms and Applications

Data Compression III

- Lempel-Ziv algorithms
- Burrows-Wheeler

15-499

Page 1

Compression Outline

Introduction: Lossy vs. Lossless, Benchmarks, ...

Information Theory: Entropy, etc.

Probability Coding: Huffman + Arithmetic Coding

Applications of Probability Coding: PPM + others

➔ **Lempel-Ziv Algorithms:**

- LZ77, gzip,
- LZ78, compress

Other Lossless Algorithms: Burrows-Wheeler

Lossy algorithms for images: JPEG, MPEG, ...

Compressing graphs and meshes: BBK

15-499

Page 2

Lempel-Ziv Algorithms

LZ77 (Sliding Window)

Variants: LZSS (Lempel-Ziv-Storer-Szymanski)

Applications: gzip, Squeeze, LHA, PKZIP, ZOO

LZ78 (Dictionary Based)

Variants: LZW (Lempel-Ziv-Welch), LZC

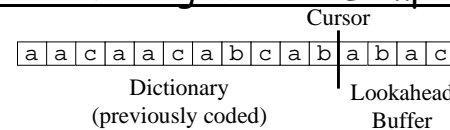
Applications: compress, GIF, CCITT (modems), ARC, PAK

Traditionally LZ77 was better but slower, but the gzip version is almost as fast as any LZ78.

15-499

Page 3

LZ77: Sliding Window Lempel-Ziv



Dictionary and **buffer** "windows" are fixed length and slide with the **cursor**

Repeat:

Output (p, l, c) where

p = position of the longest match in the dictionary (relative to the cursor)

l = length of longest match

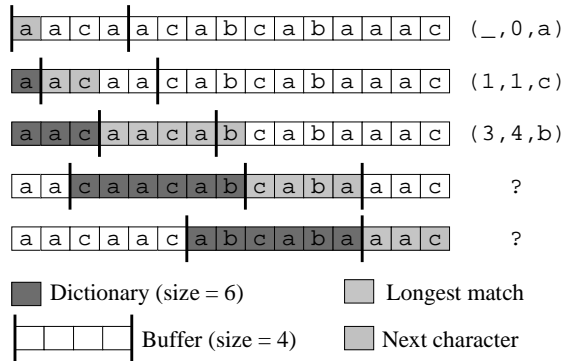
c = next char in buffer beyond longest match

Advance window by $l + 1$

15-499

Page 4

LZ77: Example



15-499

Page 5

LZ77 Decoding

Decoder keeps same dictionary window as encoder.
For each message it looks it up in the dictionary and inserts a copy

What if $l > p$? (only part of the message is in the dictionary.)

E.g. dict = abcd, codeword = (2, 9, e)

What does this give?

15-499

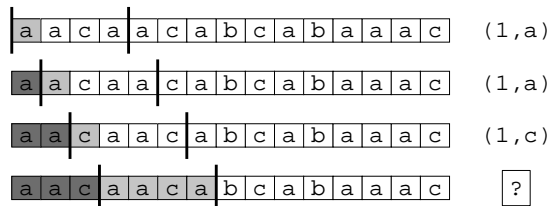
Page 6

LZ77 Optimizations used by gzip

LZSS: Output one of the following two formats

(0, position, length) or (1, char)

Uses the second format if length < 3.



15-499

Page 7

Optimizations used by gzip (cont.)

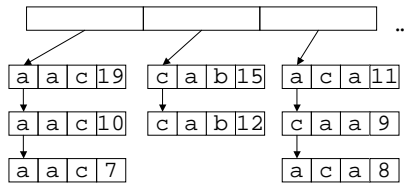
1. Huffman code the positions, lengths and chars
2. Non greedy: possibly use shorter match so that next match is better
3. Use a hash table to store the dictionary.
 - Hash keys are all strings of length 3 in the dictionary window.
 - Find the longest match within the correct hash bucket.
 - Puts a limit on the length of the search within a bucket.
 - Within each bucket store in order of position

15-499

Page 8

The Hash Table

... [7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21] ...
 ... [a | a | c | a | a | c | a | b | c | a | b | a | a | a | c] ...



15-499

Page 9

Theory behind LZ77

Sliding Window LZ is Asymptotically Optimal
 [Wyner-Ziv,94]

Will compress long enough strings to the source entropy as the window size goes to infinity.

$$H_n = \sum_{X \in A^n} p(X) \log \frac{1}{p(X)}$$

$$H = \lim_{n \rightarrow \infty} H_n$$

Uses logarithmic code (e.g. gamma) for the position.
 Problem: "long enough" is really **really** long.

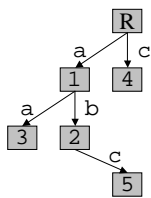
15-499

Page 10

LZ78: Dictionary Lempel-Ziv

Like LZ77, but "dictionary" is maintained differently

The **dictionary** maintains a mapping of **words** to **ids**
 Typically stored as a trie, e.g.:



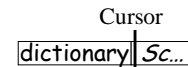
| Word | id |
|------|----|
| a | 1 |
| aa | 3 |
| ab | 2 |
| abc | 5 |
| c | 4 |

15-499

Page 11

LZ78: Encoding and Decoding

Keep a cursor as in LZ77



Repeat:

Output (*id*, *c*) where

id = The *id* of the longest match *S* found in the dictionary

c = next char in buffer beyond longest match

Add the string *Sc* to the dictionary with a new *id*

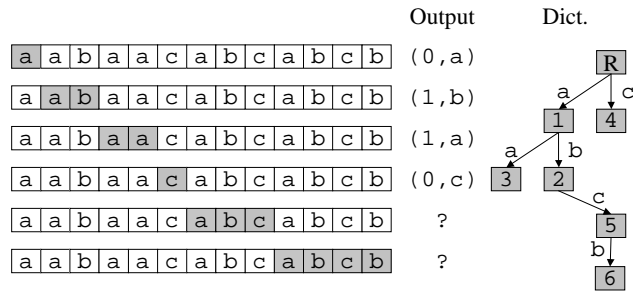
Move the cursor past *S* and *c*

Decoding keeps the same dictionary and looks up *ids*

15-499

Page 12

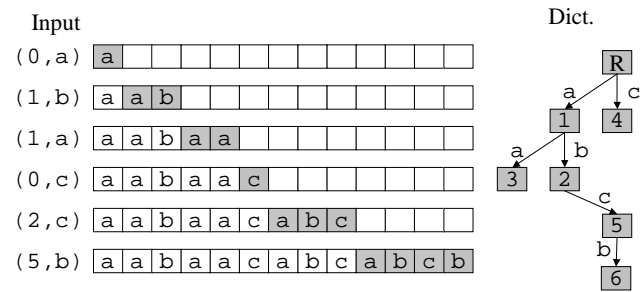
LZ78: Coding Example



15-499

Page 13

LZ78: Decoding Example



15-499

Page 14

LZW (Lempel-Ziv-Welch) [Welch84]

Don't send the extra character c , but still add Sc to the dictionary.

Initialize the dictionary with all possible character values. For bytes, the dictionary starts with 256 entries.

Decoder can only add Sc to the dictionary after decoding the next word since it does not know c .

15-499

Page 15

LZW : Possible Problem

The decoder is one step behind the encoder since it does not know c .

Consider a string $\dots Sc \dots$ where $S[0] = c$

The encoder:

- Outputs the id_1 for S and enters Sc into dictionary creating a new id_2 for Sc .
- Outputs the id_2 for Sc

The decoder:

- Inputs id_1 , looks it up and outputs S
- Inputs id_2 , looks it up, but not in the dictionary

What do we do?

15-499

Page 16

LZW: Encoding Example

| | Output | Dict. |
|---------------------------|--------|---------|
| a a b a a c a b c a b c b | 112 | 256=aa |
| a a b a a c a b c a b c b | 112 | 257=ab |
| a a b a a c a b c a b c b | 113 | 258=ba |
| a a b a a c a b c a b c b | 256 | 259=aac |
| a a b a a c a b c a b c b | 114 | 260=ca |
| a a b a a c a b c a b c b | 257 | 261=abc |
| a a b a a c a b c a b c b | 260 | 262=cab |

15-499

Page 17

LZ78 and LZW issues

What happens when the dictionary gets too large?

- Throw the dictionary away when it reaches a certain size (used in GIF)
- Throw the dictionary away when it is no longer effective at compressing (used in unix `compress`)
- Throw the least-recently-used (LRU) entry away when it reaches a certain size (used in BTLZ, the British Telecom standard)

15-499

Page 18

LZW: Decoding Example

| Input | | Dict |
|-------|---------------------------|---------|
| 112 | a a b a a c a b c a b c b | |
| 112 | a a b a a c a b c a b c b | 256=aa |
| 113 | a a b a a c a b c a b c b | 257=ab |
| 256 | a a b a a c a b c a b c b | 258=ba |
| 114 | a a b a a c a b c a b c b | 259=aac |
| 257 | a a b a a c a b c a b c b | 260=ca |
| 260 | a a b a a c a b c a b c b | 261=abc |

15-499

Page 19

Lempel-Ziv Algorithms Summary

Both LZ77 and LZ78 and their variants keep a "dictionary" of recent strings that have been seen.

The differences are:

- How the dictionary is stored
- How it is extended
- How it is indexed
- How elements are removed

15-499

Page 20

Lempel-Ziv Algorithms Summary (II)

Adapts well to changes in the file (e.g. a Tar file with many file types within it).

Initial algorithms did not use probability coding and performed poorly in terms of compression. More modern versions (e.g. gzip) do use probability coding as "second pass" and compress much better.

The algorithms are becoming outdated, but ideas are used in many of the newer algorithms.

15-499

Page 21

Compression Outline

Introduction: Lossy vs. Lossless, Benchmarks, ...

Information Theory: Entropy, etc.

Probability Coding: Huffman + Arithmetic Coding

Applications of Probability Coding: PPM + others

Lempel-Ziv Algorithms: LZ77, gzip, compress, ...

➡ **Other Lossless Algorithms:**

- Burrows-Wheeler

- ACB

Lossy algorithms for images: JPEG, MPEG, ...

Compressing graphs and meshes: BBK

15-499

Page 22

Burrows -Wheeler

Currently best "balanced" algorithm for text

Breaks file into fixed-size blocks and encodes each block separately.

For each block:

- Sort each character by its full context. This is called the **block sorting transform**.
- Use **move-to-front transform** to encode the sorted characters.

The ingenious observation is that the decoder only needs the sorted characters and a pointer to the first character of the original sequence.

15-499


Page 23

Burrows Wheeler: Example

Let's encode: $d_1e_2c_3o_4d_5e_6$

We've numbered the characters to distinguish them.

Context "wraps" around.

| <u>Context</u> | <u>Char</u> | | <u>Context</u> | <u>Output</u> |
|--------------------|----------------|--|--------------------|------------------|
| ecode ₆ | d ₁ | | dedec ₃ | o ₄ |
| coded ₁ | e ₂ | Sort Context  | coded ₁ | e ₂ |
| odede ₂ | c ₃ | | decod ₅ | e ₆ |
| dedec ₃ | o ₄ | | odede ₂ | c ₃ |
| edeco ₄ | d ₅ | | ecode ₆ | d ₁ ← |
| decod ₅ | e ₆ | | edeco ₄ | d ₅ |

15-499

Page 24

Burrows-Wheeler (Continued)

Theorem: After sorting, equal valued characters appear in the same order in the output as in the most significant position of the context.

Proof sketch: Since the chars have equal value in the most-significant-position of the context, they will be ordered by the rest of the context, i.e. the previous chars. This is also the order of the output since it is sorted by the previous characters.

| Context | Output |
|----------------------------|----------------|
| dedec ₃ | o ₄ |
| <u>code</u> d ₁ | e ₂ |
| <u>deco</u> d ₅ | e ₆ |
| ode ₂ de | c ₃ |
| <u>ecode</u> ₆ | d ₁ |
| <u>edeco</u> ₄ | d ₅ |

15-499

Page 25

Burrows-Wheeler: Decoding

Consider dropping all but the last **Context** **Output** character of the context.

- What follows the a with an arrow?
- What follows the first b?
- What is the whole string?

| | |
|---|---|
| a | c |
| a | b |
| a | b |
| b | a |
| b | a |
| c | a |

15-499

Page 26

Burrows-Wheeler: Decoding

What about now?

Can also use the "rank".
The "rank" is the position of a character if it were sorted using a stable sort.

| Output | Rank |
|--------|------|
| c | 6 |
| a | 1 |
| b | 4 |
| b | 5 |
| a | 2 |
| a | 3 |

15-499

Page 27

Burrows-Wheeler Decode

Function BW_Decode(In, Start, n)

S = MoveToFrontDecode(In, n)

R = Rank(S)

j = Start

for i=1 to n do

Out[i] = S[j]

j = R[j]

Rank gives position of each char in sorted order.

15-499

Page 28

Decode Example

| <u>S</u> | <u>Sort(S)</u> | <u>Rank(S)</u> | <u>Out</u> |
|----------------|----------------|----------------|---------------------------------|
| o ₄ | c ₃ | 6 | e ₆ d ₁ ← |
| e ₂ | d ₁ | 4 | d ₁ e ₂ |
| e ₆ | d ₅ | 5 | e ₂ c ₃ |
| c ₃ | e ₂ | 1 | c ₃ o ₄ |
| d ₁ | e ₆ | 2 ← | o ₄ d ₅ |
| d ₅ | o ₄ | 3 | d ₅ e ₆ |

Overview of Text Compression

PPM and **Burrows-Wheeler** both encode a single character based on the immediately preceding context.

LZ77 and **LZ78** encode multiple characters based on matches found in a block of preceding text

Can you **mix these ideas**, *i.e.*, code multiple characters based on immediately preceding context?

- **BZ** does this, but they don't give details on how it works - current best compressor
- **ACB** also does this - close to best

ACB (Associate Coder of Buyanovsky)

Keep dictionary sorted by context
(the last character is the most significant)

- Find longest match for context
- Find longest match for contents
- Code
 - Distance between matches in the sorted order
 - Length of contents match

| <u>Context</u> | <u>Contents</u> |
|----------------|-----------------|
| | decode |
| | dec ode |
| | d ecode |
| | decod e |
| | de code |
| | deco de |

Has aspects of Burrows-Wheeler, and LZ77