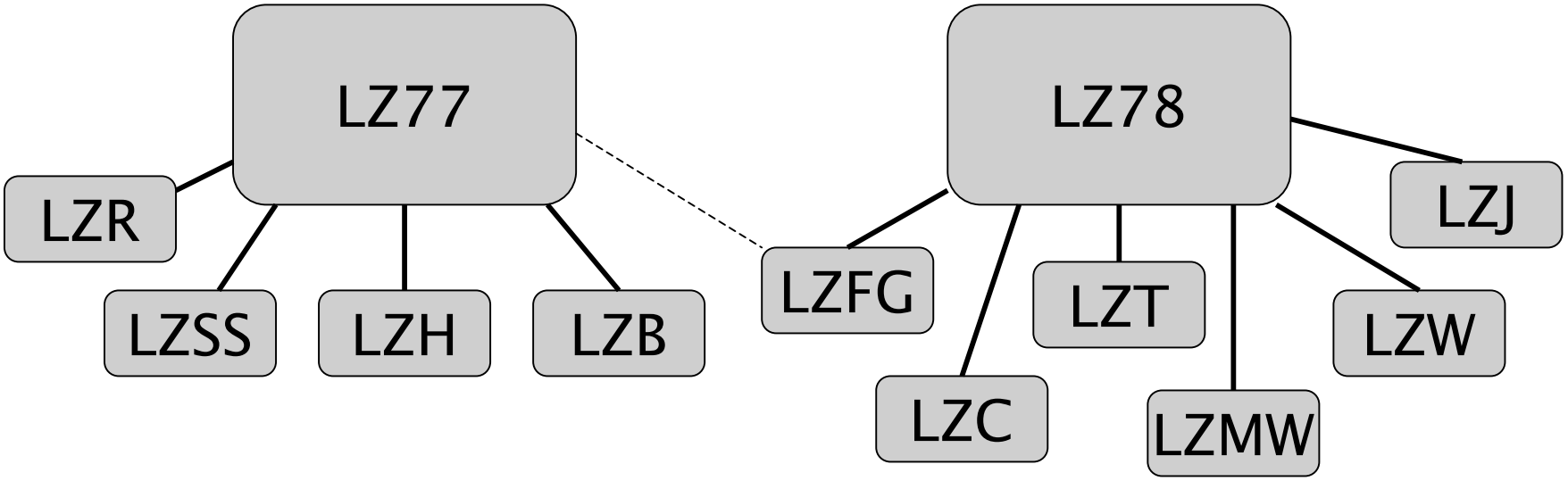
The Lempel Ziv Algorithm

Seminar “Famous Algorithms”
January 16, 2003

christina.zeeh@studi.informatik.uni-stuttgart.de

The (?) Lempel Ziv Algorithm



Applications:

- zip
- gzip
- Stacker
- ...

Applications:

- GIF
- V.42
- compress
- ...

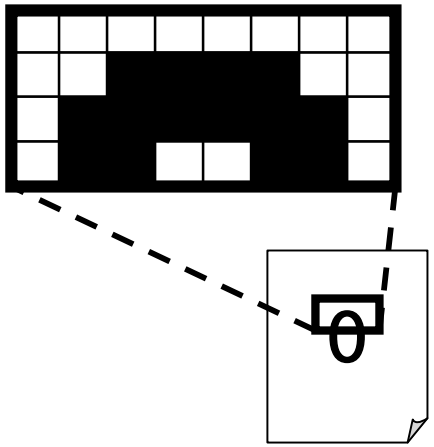
Overview

- Introduction
- Lossless Compression
- Dictionary Coding
- LZ77
  - Algorithm
  - Modifications
  - Comparison
- LZ78
  - Algorithm
  - Modifications
  - Comparison

Data Compression

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

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.



Lossless Compression

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

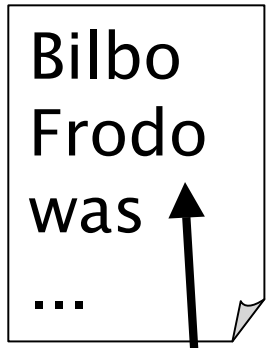
- Run-length coding
- Statistical methods
  - Huffman coding
  - Arithmetic coding
  - PPM
- Dictionary methods
  - Lempel Ziv algorithms

Dictionary Coding (1)

- Observation: Correlations between parts of the data (patterns)
- Idea: Replace recurring patterns with references to a dictionary
- Static, semi-adaptive, adaptive
- LZ algorithms use adaptive approach
  - ✓ coding scheme is universal
  - ✓ no need to transmit/store dictionary
  - ✓ single-pass (dictionary creation “on-the-fly”)

Dictionary Coding (2)

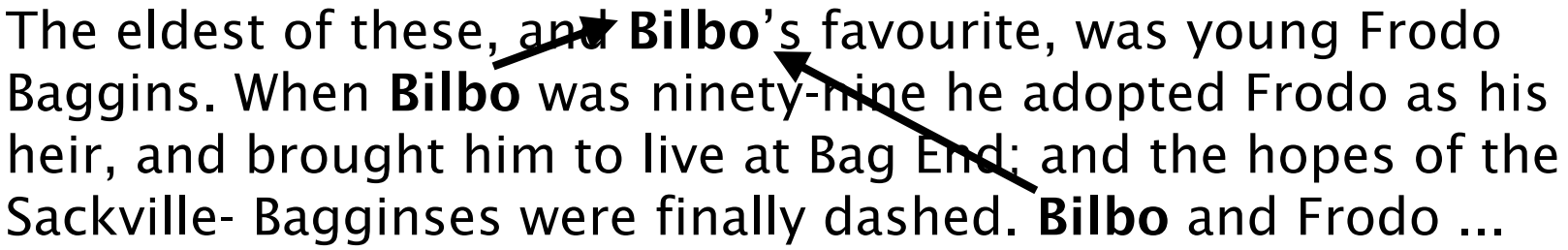
- Keep explicit dictionary (LZ78 algorithm)



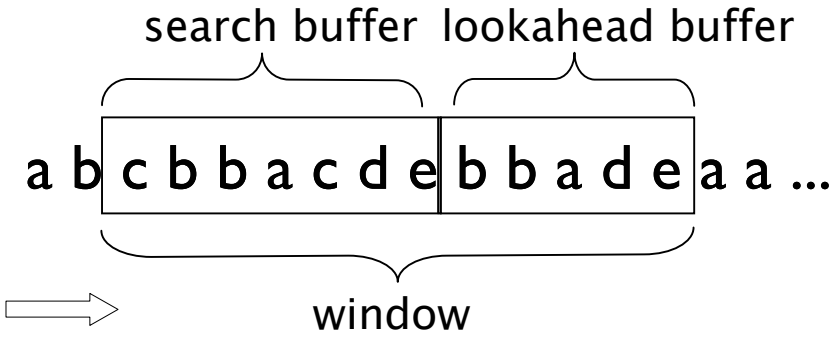
The eldest of these, and **Bilbo**'s favourite, **was** young **Frodo** Baggins. When Bilbo was ninety-nine he adopted Frodo as his heir, and brought him to live at Bag End; and the hopes of the Sackville- Bagginses were finally dashed. Bilbo and Frodo ...

Dictionary Coding (3)

- Use previously processed data as dictionary (LZ77 algorithm)



LZ77 (1)



Match: “bba”
Position: 3
Length: 3
Next symbol: ‘d’
Output: (3, 3, ‘d’)

- Memory / speed constraints require restrictions
  ⇨ use a fixed-size window (“sliding window” principle)

LZ77 (2)

```
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 . . .

□ = 3 *(size of alphabet)*

L_s = 9 *(lookahead buffer size)*

n = 18 *(window size)*

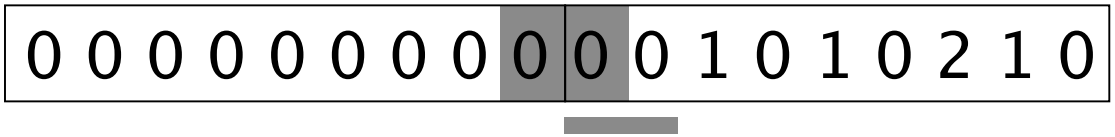
Codeword length:

$L_c = 1 + \log_{□}(n - L_s) + \log_{□}(L_s)$

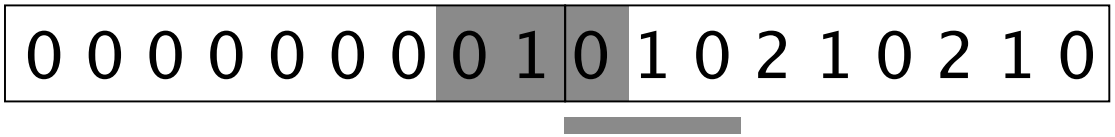
$= 1 + \log_3(9) + \log_3(9)$

$= 5$

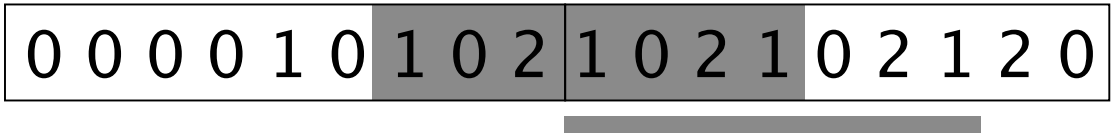
LZ77 Example – Encoder

1. 

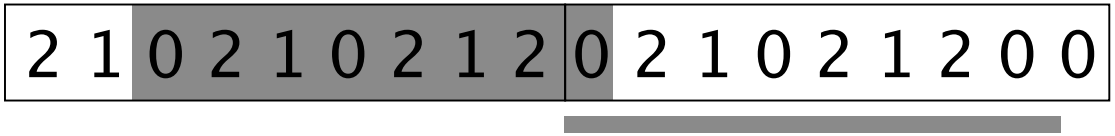
C_1=22 02 1

2. 

C_2=21 11 2

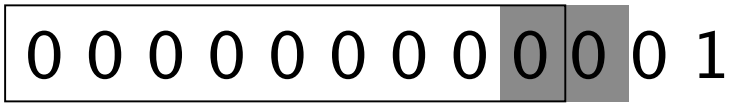
3. 

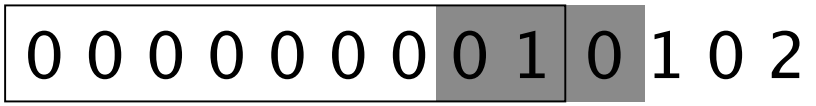
C_3=20 21 2

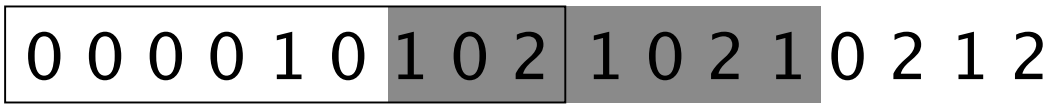
4. 

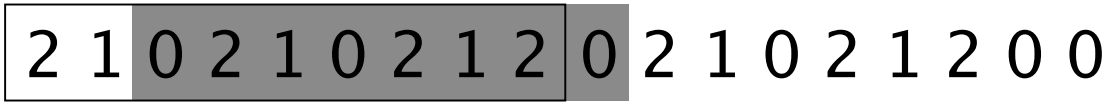
C_4=02 22 0

LZ77 Example – Decoder

1. C_1=22 02 1 

2. C_2=21 11 2 

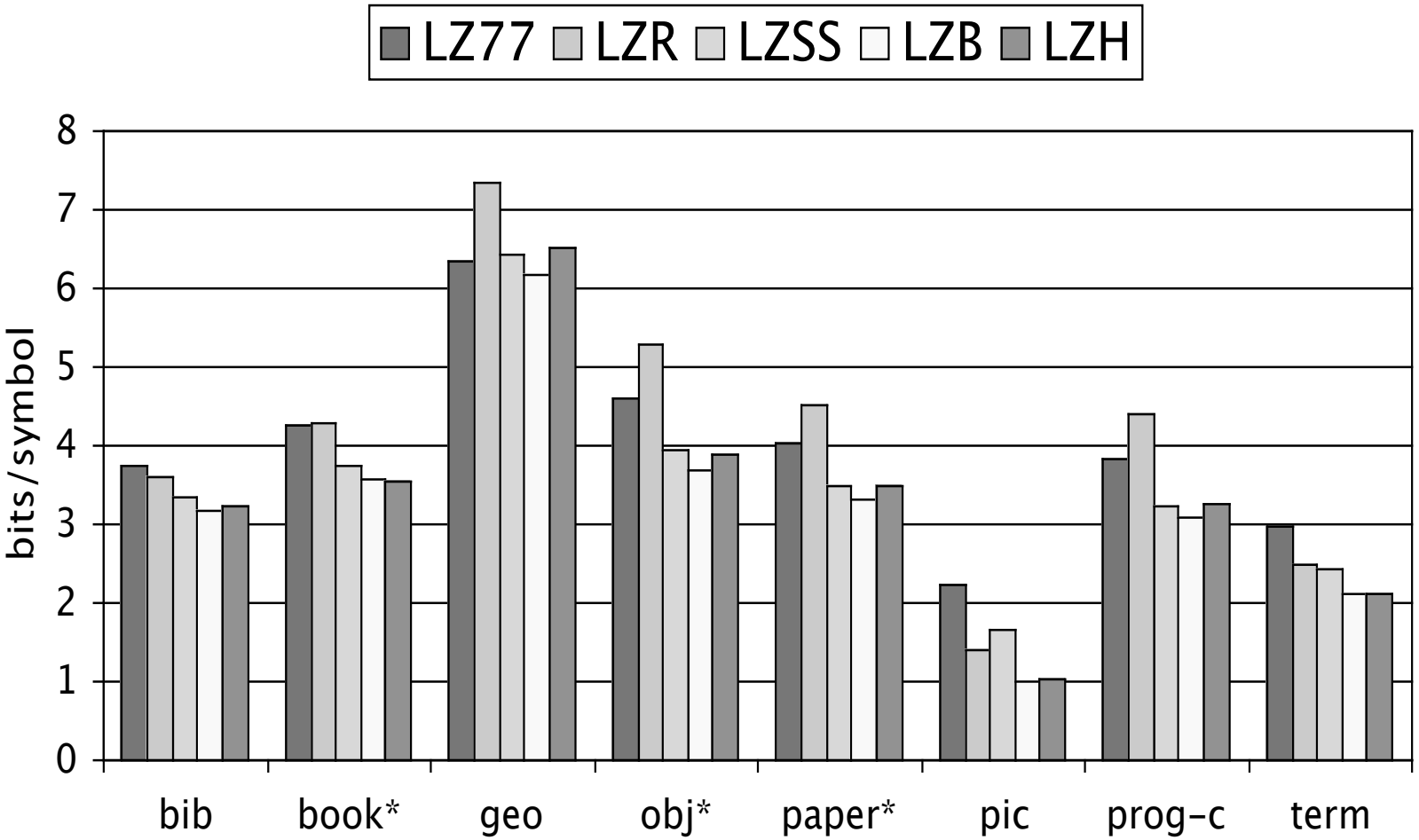
3. C_3=20 21 2 

4. C_4=02 22 0 

LZ77 Improvements

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

LZ77 Comparison



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

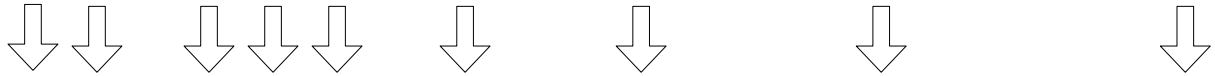
LZ78 (1)

- 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)

```
w := NIL;
while (there is input) {
  K := next symbol from input;
  if (wK exists in the dictionary) {
     w := wK;
  } else {
     output (index(w), K);
     add wK to the dictionary;
     w := NIL;
  }
}
```

LZ78 Example – Encoder



#	entry	phrase	Output:	(ternary)
1	0	0	0 0	(0 0)
2	1+1	01	1 1	(1 1)
3	2	2	0 2	(0 2)
4	1	1	0 1	(00 1)
5	3+1	21	3 1	(10 1)
6	5+0	210	5 0	(12 0)
7	6+1	2101	6 1	(20 1)
8	7+2	21012	7 2	(21 2)
9	7+1	21011	7 1	(21 1)

LZ78 Example – Decoder

Input:

0 0 ✓
1 1 ✓
0 2 ✓
0 1 ✓
3 1 ✓
5 0 ✓
6 1 ✓
7 2 ✓
7 1 ✓

#	entry	phrase
1	0	0
2	1+1	01
3	2	2
4	1	1
5	3+1	21
6	5+0	210
7	6+1	2101
8	7+2	21012
9	7+1	21011

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

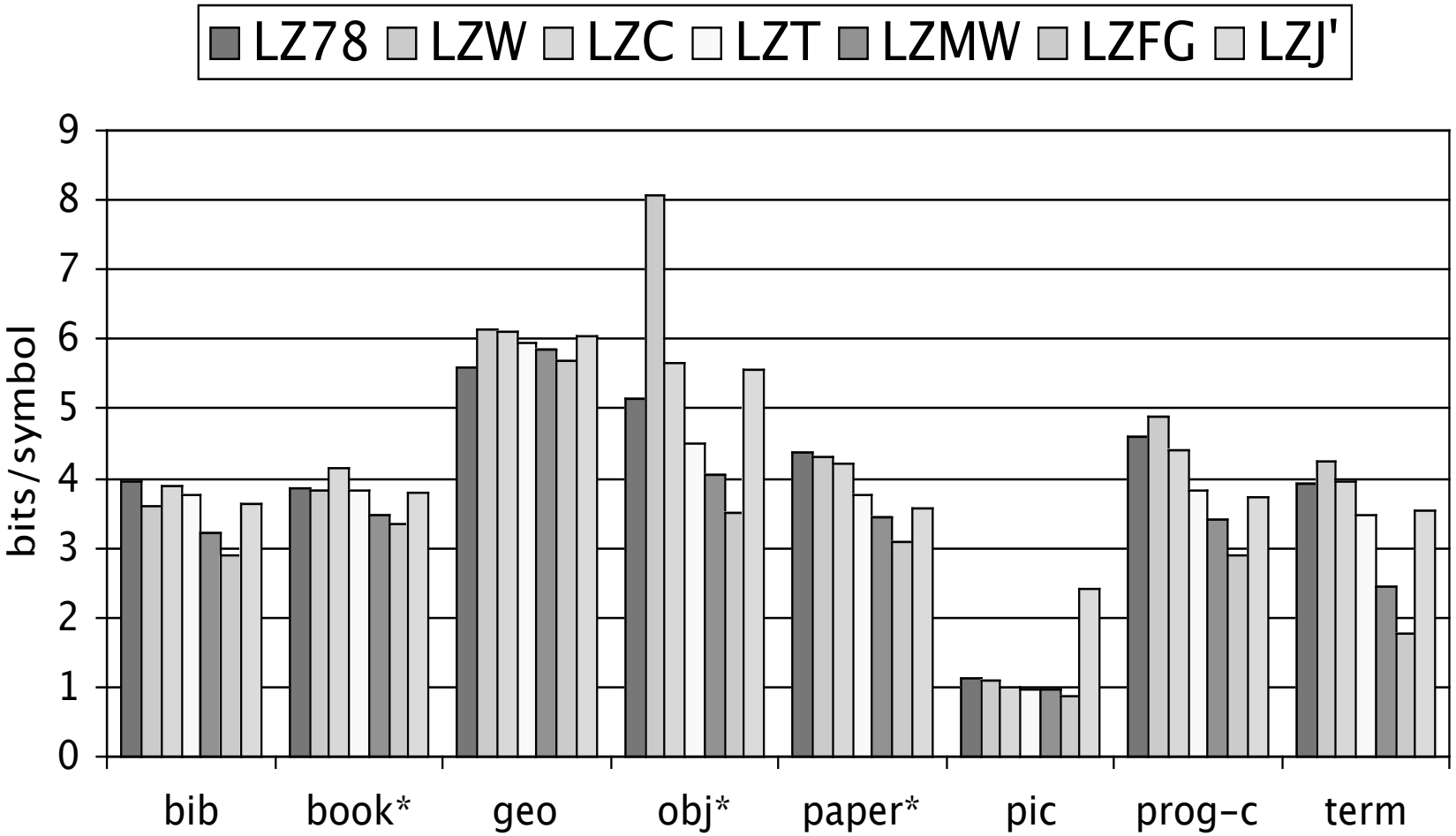
LZW (2)

```
w := NIL;
while (there is input){
  K := next symbol from input;
  if (wK exists in the dictionary) {
      w := wK;
  } else {
      output (index(w));
      add wK to the dictionary;
      w := K;
  }
}
```

LZ78 Other Improvements

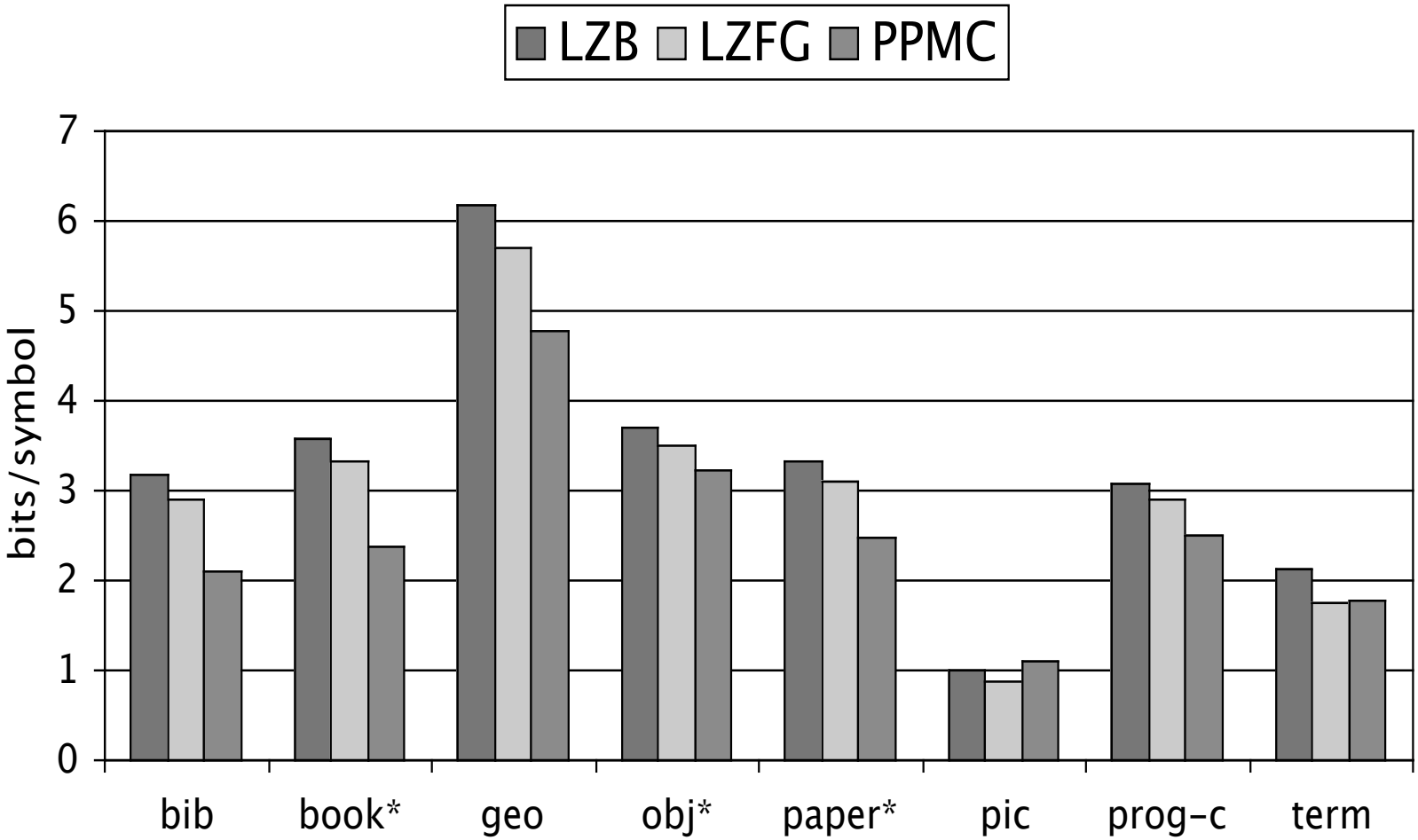
LZC	variable-length pointers, increasing pointer size, monitor compression ratio
LZT	LZW + removal of least recently used entries
LZMW	new entries created by concatenating two last encoded phrases
LZJ	dictionary contains every unique string of the data up to certain length, delete entries used only once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

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?