CSE 589
Applied Algorithms
Spring 1999

Dictionary Coding
Sequitur

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

a b a b a b a b a

LZW Encoding Example (2)

Dictionary

0 a
1 b
2 ab

a b a b a b a b a
0

LZW Encoding Example (3)

Dictionary

0 a
1 b
2 ab
3 ba

a b a b a b a b a
0 1

LZW Encoding Example (4)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba

a b a b a b a b a
0 1 2

LZW Encoding Example (5)

Dictionary

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

a b a b a b a b a
0 1 2 4

LZW Encoding Example (6)

Dictionary

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

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

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

0 a
1 b
2 a?

0 1 2 4 3 6
a

LZW Decoding Example (2a)

Dictionary

0 a
1 b
2 ab

0 1 2 4 3 6
a b

LZW Decoding Example (2b)

Dictionary

0 a
1 b
2 ab
3 b?

0 1 2 4 3 6
a b

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 ab 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 ab aba 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 (6a)

Dictionary

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

0 1 2 4 3 6
a b ab aba ba b

LZW Decoding Example (6b)

Dictionary

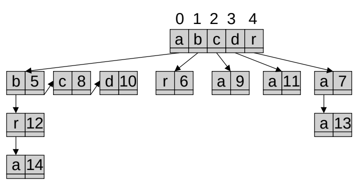
0	a
1	b
2	ab
3	ba
4	aba
5	abab
6	bab
7	bab?

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

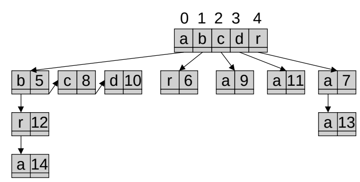
Trie Data Structure for Dictionary

- Fredkin (1960)

0	a	9	ca
1	b	10	ad
2	c	11	da
3	d	12	abr
4	r	13	raa
5	ab	14	abra
6	br		
7	ra		
8	ac		

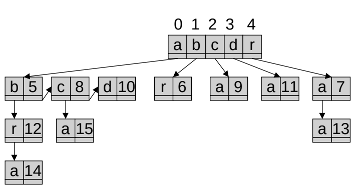


Encoder Uses a Trie (1)



a b r a c a d a b r a a b r a c a d a b r a
0 1 4 0 2 0 3 5 7 12

Encoder Uses a Trie (2)



a b r a c a d a b r a a b r a c a d a b r a
0 1 4 0 2 0 3 5 7 12 8

Decoder's Data Structure

- Simply an array of strings

0	a	9	ca
1	b	10	ad
2	c	11	da
3	d	12	abr
4	r	13	raa
5	ab	14	abr?
6	br		
7	ra		
8	ac		

0 1 4 0 2 0 3 5 7 12 8 ...
a b r a c a d ab ra abr

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

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 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

Context-Free Grammar Example

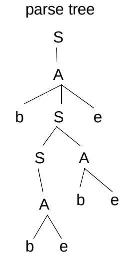
- S -> SA
  S -> A
  A -> bSe
  A -> be

Example: b and e matched as parentheses

derivation of bbebee

S
A
bSe
bSAe
bAAe
bbeAe
bbebee

hierarchical parse tree



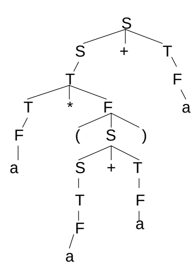
Arithmetic Expressions

- S -> S + T
  S -> T
  T -> T*F
  T -> F
  F -> a
  F -> (S)

derivation of a * (a + a) + a

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

parse tree



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)

bbebeebebebbebee

S -> b

Sequitur Example (2)

bbebeebebebbebee

S -> bb

Sequitur Example (3)

bbebeebebebbebee

S -> bbe

Sequitur Example (4)

bbebeebebebbebee

S -> bbeb

Sequitur Example (5)

bbebeebebebbebee

S -> bbebe

Enforce digram uniqueness.
be occurs twice.
Create new rule A ->be.

Sequitur Example (6)

bbebeebebebbebee

S -> bAA
A -> be

Sequitur Example (7)

bbebeebebebbebee

S -> bAAe
A -> be

Sequitur Example (8)

bbebeebebebbebee

S -> bAAeb
A -> be

Sequitur Example (9)

bbebeebebebbebee

S -> bAAebe
A -> be

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

Sequitur Example (10)

bbebeebebebbebee

S -> bAAeA
A -> be

Sequitur Example (11)

bbebeebebebbebee

S -> bAAeAb
A -> be

Sequitur Example (12)

bbebeebebebbebee

S -> bAAeAbe
A -> be

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

Sequitur Example (13)

bbebeebebebbebee

S -> bAAeAA
A -> be

Enforce digram uniqueness
AA occurs twice.
Create new rule B -> AA.

Sequitur Example (14)

bbebeebebebbebee

S -> bBeB
A -> be
B -> AA

Sequitur Example (15)

bbebeebebebbebee

S -> bBeBb
A -> be
B -> AA

Sequitur Example (16)

bbebeebebebbebee

S -> bBeBbb
A -> be
B -> AA

Sequitur Example (17)

bbebeebebebbebee

S -> bBeBbbe
A -> be
B -> AA

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

Sequitur Example (18)

bbebeebebebbebee

S -> bBeBbA
A -> be
B -> AA

Sequitur Example (19)

bbebeebebebbebee

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

Sequitur Example (20)

bbebeebebebbebee

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

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

Sequitur Example (21)

bbebeebebebbebee

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

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

Sequitur Example (22)

bbebeebebebbebee

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

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

Sequitur Example (23)

bbebeebebebbebee

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

Sequitur Example (24)

bbebeebebebbebee

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

Enforce digram uniqueness.
Ce occurs twice.
Create new rule D -> Ce.

Sequitur Example (25)

bbebeebebebbebee

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

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

Sequitur Example (26)

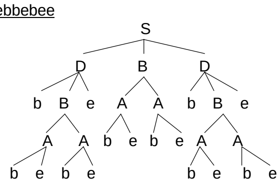
bbebeebebebbebee

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

The Hierarchy

bbebeebebebbebee

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



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

Sequitur Algorithm

Input the first symbol s to create the production S -> s;
repeat
match an existing rule:
A -> ….XY… → A -> ….B….
B -> XY → B -> XY
create a new rule:
A -> ….XY…. → A -> ….C....
B -> ....XY.... → B -> ....C....
C -> XY
remove a rule:
A -> ….B….
B -> $X_1X_2…X_k$ → A -> …. $X_1X_2…X_k$ ….
input a new symbol:
S -> $X_1X_2…X_k$ → S -> $X_1X_2…X_k$s
until no symbols left

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.

Sequitur Rule Complexity

- digram Uniqueness - match an existing rule.

A -> ….XY… → A -> ….B….
B -> XY → B -> XY

s	r	s - r/2
-1	0	-1

- digram Uniqueness - create a new rule.

A -> ….XY…. → A -> ….C....
B -> ....XY.... → B -> ....C....
C -> XY

s	r	s - r/2
0	1	-1/2

- Rule Utility - Remove a rule.

A -> ….B….
B -> $X_1X_2…X_k$ → A -> …. $X_1X_2…X_k$ ….

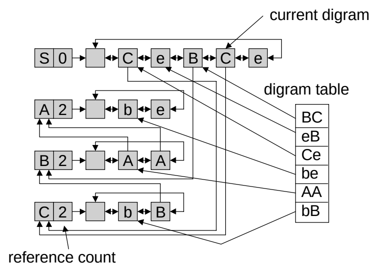
s	r	s - r/2
-1	-1	-1/2

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

Symbol Code

b	000
e	001
S	010
A	011
B	100
D	101
#	110

Grammar Code

D	B	D	#	b	e	#	A	A	#	b	B	e	
101	100	101	110	000	001	110	011	011	110	000	100	001	39 bits

|Grammar Code| = $(s+r-1)\lceil \log_2(r+a+1) \rceil$

r = number of rules
s = sum of right hand sides
a = number in original symbol alphabet

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

	size	compress	gzip	sequitur	PPMC
bib	111261	3.35	2.51	2.48	2.12
book	768771	3.46	3.35	2.82	2.52
geo	102400	6.08	5.34	4.74	5.01
obj2	246814	4.17	2.63	2.68	2.77
pic	513216	0.97	0.82	0.90	0.98
progc	38611	3.87	2.68	2.83	2.49

Files from the Calgary Corpus
Units in bits per character (8 bits)
compress - based on LZW
gzip - based on LZ77
PPMC - adaptive arithmetic coding with context

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.