# CSE 490 GZ
# Introduction to Data Compression
Winter 2002

Sequitur

## 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
  - non-terminals: S, A
  - Production Rules:
    $S \rightarrow SA$, $S \rightarrow A$, $A \rightarrow bSe$, $A \rightarrow be$
  - S is the start symbol



## Context-Free Grammar Example

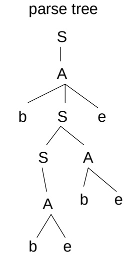
- $S \rightarrow SA$
  $S \rightarrow A$
  $A \rightarrow bSe$
  $A \rightarrow be$

Example: b and e matched as parentheses

derivation of bbebee

S
A
bSe
bSAe
bAAe
bbeAe
bbebee

hierarchical parse tree





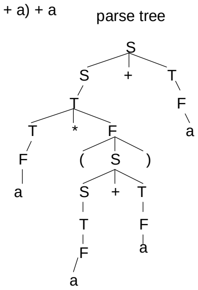

## Arithmetic Expressions

- $S \rightarrow S + T$
  $S \rightarrow T$
  $T \rightarrow T*F$
  $T \rightarrow F$
  $F \rightarrow a$
  $F \rightarrow (S)$

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

$S \to DBD$
$A \to be$
$B \to AA$
$C \to bB$
$D \to Ce$

Enforce rule utility.
C occurs only once.
Remove $C \to bB$.



## Sequitur Example (26)

bbebeebebebbebee

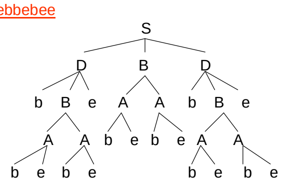
$S \to DBD$
$A \to be$
$B \to AA$
$D \to bBe$



## The Hierarchy

bbebeebebebbebee

$S \to DBD$
$A \to be$
$B \to AA$
$D \to bBe$




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



## Sequitur Algorithm

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



## Complexity

- The number of non-input sequitur operations applied $\leq 2n$ 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 2s - r.
  - Initially 2s - r = 1 because s = 1 and r = 1.
  - 2s - r $\geq$ 0 at all times because each rule has at least 1 symbol on the right hand side.
  - 2s - r increases by 2 for every input operation.
  - 2s - r decreases by at least 1 for each non-input sequitur rule applied.



## Sequitur Rule Complexity

- Digram Uniqueness - match an existing rule.

| | | | s | r | 2s - r |
|---|---|---|---|---|---|
| $A \to \ldots.XY\ldots$<br>$B \to XY$ | ⟶ | $A \to \ldots.B\ldots.$<br>$B \to XY$ | -1 | 0 | -2 |

- Digram Uniqueness - create a new rule.

| | | | s | r | 2s - r |
|---|---|---|---|---|---|
| $A \to \ldots.XY\ldots.$<br>$B \to \ldots.XY\ldots.$ | ⟶ | $A \to \ldots.C\ldots.$<br>$B \to \ldots.C\ldots.$<br>$C \to XY$ | 0 | 1 | -1 |

- Rule Utility - Remove a rule.

| | | | s | r | 2s - r |
|---|---|---|---|---|---|
| $A \to \ldots.B\ldots.$<br>$B \to X_1X_2\ldots X_k$ | ⟶ | $A \to \ldots.\ X_1X_2\ldots X_k\ \ldots.$ | -1 | -1 | -1 |

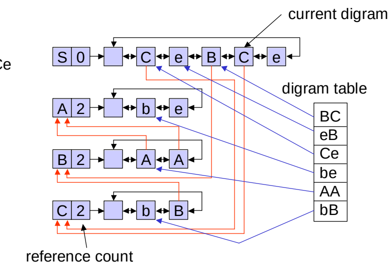
## Linear Time Algorithm

- 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 non-terminal 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 | | Symbol Code | | |
|---|---|---|---|---|
| | S → DBD<br>A → be<br>B → AA<br>D → bBe | | b 000<br>e 001<br>A 010<br>B 011<br>D 100<br># 101 | No code for S needed |

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

$r$ = 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 uses LZ77 ideas.
  - Send the right hand side of the S production.
  - The first time a non-terminal is sent, its right hand side is transmitted instead.
  - The second time a non-terminal is sent as a tuple <i,j,k> which says the right hand side starts occurs in production i, at position j and is k long. A new production rule is then added to a dictionary.
  - Subsequently, the non-terminal is represented by the index of the production rule.



## Compression Quality

- Neville-Manning and Witten 1997

| | size | comp | gzip | sequitur | PPMC | bzip2 |
|---|---|---|---|---|---|---|
| bib | 111261 | 3.35 | 2.51 | 2.48 | 2.12 | 1.98 |
| book | 768771 | 3.46 | 3.35 | 2.82 | 2.52 | 2.42 |
| geo | 102400 | 6.08 | 5.34 | 4.74 | 5.01 | 4.45 |
| obj2 | 246814 | 4.17 | 2.63 | 2.68 | 2.77 | 2.48 |
| pic | 513216 | 0.97 | 0.82 | 0.90 | 0.98 | 0.78 |
| progc | 38611 | 3.87 | 2.68 | 2.83 | 2.49 | 2.53 |

First
Second
Third

Files from the Calgary Corpus
Units in bits per character (8 bits)
compress - based on LZ78 and LZW
gzip - based on LZ77
PPMC - adaptive arithmetic coding with context
bzip2 - Burrows Wheeler transform



## Notes on Sequitur

- Very new and different from the standards.
- Yields compression and hierarchical structure simultaneously.
- With clever encoding is competitive with the best of the standards.
- Practical linear time encoding and decoding.
- Alternatives
  - Off-line algorithms – (i) find the most frequent digram, (ii) find the longest repeated substring

## Other Grammar Based Methods

- YK Algorithm
  - Kieffer, Yang 2000
  - Like Sequitur, but does not allow different non-terminals to generate the same string
  - Slower, but has some better theoretical properties
- Longest Match
- Most frequent digram
- Match producing the best compression