CSEP 521
Applied Algorithms
Spring 2005

Dictionary Coding
Plan for Tonight

• Overview
• LZW
• Sequitur
• Move-to-front coding
• Burrows-Wheeler Transform
Dictionary Coding

• Does not use statistical knowledge of data.
• Encoder: As the input is processed develop a dictionary and transmit the index of strings found in the dictionary.
• Decoder: As the code is processed reconstruct the dictionary to invert the process of encoding.
• Examples: LZW, LZ77, Sequitur, Burrows-Wheeler
• Applications: Unix Compress, gzip, bzip, GIF
Overview of Dictionary Compression

Encoder

Dictionary Inference

Symbol stream

Entropy coder

Decoder

Dictionary Derivation

Symbol stream

Entropy decoder

String x

Compressed bit stream

x
LZW Dictionary Inference 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>
</tbody>
</table>

```
ab a b a b a b a a
0 1
```
LZW Encoding Example (4)

Dictionary

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>
</tbody>
</table>

ab ab ab a b a b a
0 1 2
LZW Encoding Example (5)

Dictionary

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

ab a b a b a b a
0 1 2 4
LZW Encoding Example (6)

Dictionary

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>

Encoded: a b a b a b a b a

0 1 2 4 3
LZW Dictionary Derivation 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

<table>
<thead>
<tr>
<th>Code</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>

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>
</tbody>
</table>

0 1 2 4 3 6
a b a
## LZW Decoding Example (3b)

**Dictionary**

<table>
<thead>
<tr>
<th>Code</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>ab?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Raw</th>
<th>Decoded</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>ab?</td>
</tr>
</tbody>
</table>
LZW Decoding Example (4a)

Dictionary

<table>
<thead>
<tr>
<th>Code</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>
</tbody>
</table>

012436 

a b ab a
LZW Decoding Example (4b)

Dictionary

<table>
<thead>
<tr>
<th>Code</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>aba?</td>
</tr>
</tbody>
</table>

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

Dictionary

<table>
<thead>
<tr>
<th>Code</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>

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

Dictionary

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>
<tr>
<td>6</td>
<td>ba?</td>
</tr>
</tbody>
</table>

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

Dictionary

<table>
<thead>
<tr>
<th>0</th>
<th>a</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>6</td>
<td>bab</td>
</tr>
</tbody>
</table>

0 1 2 4 3 6

a b ab aba ba b
LZW Decoding Example (6b)

Dictionary

<table>
<thead>
<tr>
<th>Code</th>
<th>String</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>
<tr>
<td>6</td>
<td>bab</td>
</tr>
<tr>
<td>7</td>
<td>bab?</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Base Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 a</td>
</tr>
<tr>
<td>1 b</td>
</tr>
<tr>
<td>2 c</td>
</tr>
<tr>
<td>3 d</td>
</tr>
<tr>
<td>4 r</td>
</tr>
</tbody>
</table>

0 1 4 0 2 0 3 5 7
Trie Data Structure for Encoder’s Dictionary

- Fredkin (1960)
Encoder Uses a Trie (1)
Encoder Uses a Trie (2)

```
abraca
```

```
d a b r a
```

```
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

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>9</th>
<th>ca</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>10</td>
<td>ad</td>
</tr>
<tr>
<td>1</td>
<td>c</td>
<td>11</td>
<td>da</td>
</tr>
<tr>
<td>2</td>
<td>d</td>
<td>12</td>
<td>abr</td>
</tr>
<tr>
<td>3</td>
<td>r</td>
<td>13</td>
<td>raa</td>
</tr>
<tr>
<td>4</td>
<td>ab</td>
<td>14</td>
<td>abr?</td>
</tr>
<tr>
<td>5</td>
<td>br</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ra</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>ac</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 1 4 0 2 0 3 5 7 12 8 ...
abracadabraabr
Bounded Size Dictionary

• Bounded Size Dictionary
  – $n$ bits of index allows a dictionary of size $2^n$
  – Doubtful that long entries in the dictionary will be useful.

• Strategies when the dictionary reaches its limit.
  1. Don’t add more, just use what is there.
  2. Throw it away and start a new dictionary.
  3. Double the dictionary, adding one more bit to indices.
  4. Throw out the least recently visited entry to make room for the new entry.
Notes on LZW

• Extremely effective when there are repeated patterns in the data that are widely spread.
• Negative: Creates entries in the dictionary that may never be used.
• Applications:
  – Unix compress, GIF, V.42 bis modem standard
**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 $bbebee$

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>bSe</td>
</tr>
<tr>
<td>bSAe</td>
<td>bA Ae</td>
</tr>
<tr>
<td>bbeAe</td>
<td>bbebee</td>
</tr>
</tbody>
</table>

hierarchical parse tree

```
S
  |
  A
  |
  bSe
  |
  bSAe
  |
  bAe
  |
  bbeAe
  |
  bbebee
```
Arithmetic Expressions

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

Parse tree for the derivation of \( a * (a + a) + a \):
Overview of Grammar Compression

Grammar inference → Context-free grammar → Grammar encoding → Symbol stream → Entropy coder

Grammar derivation → Context-free grammar → Grammar decoding → Symbol stream → Entropy decoder

String x → Encoder → Decoder → x

Lecture 6 - Dictionary Coding
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)

\[ \text{bbebeebebebebee} \]

\[ S \rightarrow b \]
Sequitur Example (2)

\[ \text{bbeebeebeebeebee} \]

\[ S \rightarrow \text{bb} \]
Sequitur Example (3)

$bbe$beebebebebee

$S \rightarrow bbe$
Sequitur Example (4)

\text{bbbeb}ebebebebseebeb

S \rightarrow \text{bbeb}
Sequitur Example (5)

\textcolor{red}{b}b\textcolor{red}{e}b\textcolor{red}{e}e\textcolor{red}{b}e\textcolor{red}{b}e\textcolor{red}{b}e\textcolor{red}{b}e\textcolor{red}{e}e

$S \rightarrow \textcolor{red}{b}b\textcolor{red}{e}b\textcolor{red}{e}$  

Enforce digram uniqueness.  
be occurs twice.  
Create new rule $A \rightarrow \text{be}$.  

Lecture 6 - Dictionary Coding
Sequitur Example (6)

\[ \text{bbebe} \text{ ebebebbeebee} \]

\[ S \rightarrow bAA \]
\[ A \rightarrow be \]
Sequitur Example (7)

bbebeebebebebee

S $\rightarrow$ bAAe
A $\rightarrow$ be
Sequitur Example (8)

\[
\text{bbebeeb} \rightarrow \text{ebebbbebee}\\
S \rightarrow bAAb\\
A \rightarrow be
\]
Sequitur Example (9)

bbebeebebebbeebee

S → bAAebe
A → be

Enforce digram uniqueness.
be occurs twice.
Use existing rule A → be.
Sequitur Example (10)

bbebeebe bebebee

S → bAAeA
A → be
Sequitur Example (11)

\[ bbebeebeebbebee \]

S \rightarrow bAAeAb
A \rightarrow be
Sequitur Example (12)

\textcolor{orange}{b}be\textcolor{orange}{bee}bebe

\[ S \rightarrow bA\textcolor{orange}{A}eA\textcolor{orange}{A}e \]
\[ A \rightarrow be \]

Enforce digram uniqueness.
be occurs twice.
Use existing rule \( A \rightarrow be \).
Sequitur Example (13)

\texttt{bbebeebebebbebee}

\begin{align*}
S & \rightarrow \text{bAAeAA} \\
A & \rightarrow \text{be}
\end{align*}

Enforce digram uniqueness
AA occurs twice.
Create new rule \( B \rightarrow \text{AA} \).
Sequitur Example (14)

bbebeebebe bbebee

S → bBeB
A → be
B → AA
Sequitur Example (15)

bbebeebebeb bebee

S → bBeBb
A → be
B → AA
Sequitur Example (16)

\[ bbebeebebebbbeee \]

\[ S \rightarrow bBeBbb \]
\[ A \rightarrow be \]
\[ B \rightarrow AA \]
Sequitur Example (17)

$bbebeebebebbe$bee

S → bBeBbbe
A → be
B → AA

Enforce digram uniqueness.
be occurs twice.
Use existing rule $A \rightarrow be$. 
Sequitur Example (18)

**bbebeeebeebbe**bee

S → bBeBbA
A → be
B → AA
Sequitur Example (19)

bbbebebebebebebe

S -> bBeBbAb
A -> be
B -> AA
Sequitur Example (20)

\texttt{bbebeebebebbebebe}

\begin{align*}
S & \rightarrow bBeBbAbe \\
A & \rightarrow \text{be} \\
B & \rightarrow \text{AA}
\end{align*}

Enforce digram uniqueness.
be occurs twice.
Use existing rule $A \rightarrow \text{be}$. 
Sequitur Example (21)

bbbebeebebebebebebebebe

S → bBeBbAA
A → be
B → AA

Enforce digram uniqueness.
AA occurs twice.
Use existing rule B → AA.
Sequitur Example (22)

bbebeebebebebebebebe

\[ S \rightarrow bBeBbB \]
\[ A \rightarrow be \]
\[ B \rightarrow AA \]

Enforce digram uniqueness.
bB occurs twice.
Create new rule C $\rightarrow$ bB.
Sequitur Example (23)

bbebeebebebebebebebe

S → CeBC
A → be
B → AA
C → bB
Sequitur Example (24)

bbebeebebebebebee

S → CeBCe  Enforce digram uniqueness.
A → be  Ce occurs twice.
B → AA  Create new rule D → Ce.
C → bB
Sequitur Example (25)

bbebeebebebebeebee

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 $S \rightarrow s$;
repeat
match an existing rule:
   $A \rightarrow \ldots XY\ldots$  $A \rightarrow \ldots B\ldots$
   $B \rightarrow XY$  $B \rightarrow XY$
create a new rule:
   $A \rightarrow \ldots XY\ldots$  $A \rightarrow \ldots C\ldots$
   $B \rightarrow \ldots XY\ldots$  $B \rightarrow \ldots C\ldots$
remove a rule:
   $A \rightarrow \ldots B\ldots$
   $B \rightarrow X_1X_2\ldots X_k$  $A \rightarrow \ldots X_1X_2\ldots X_k\ldots$
input a new symbol:
   $S \rightarrow X_1X_2\ldots X_k$  $S \rightarrow X_1X_2\ldots X_k s$
until no symbols left
Exercise

Use Sequitur to construct a grammar for aaaaaaaaaaa = a^{10}
Complexity

- The number of non-input sequitur operations applied $< 2n$ where $n$ is the input length.
- Since each operation takes constant time, sequitur is a linear time algorithm
Amortized Complexity Argument

– Let \( m \) = \# of non-input sequitur operations.
  Let \( n \) = input length. Show \( m \leq 2n \).
– 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 \( r = 1 \).
– \( 2s - r > 0 \) at all times because each rule has at least 1 symbol on the right hand side.
Sequitur Rule Complexity

- **Digram Uniqueness** - match an existing rule.

  \[
  \begin{align*}
  A & \rightarrow \ldots XY \ldots & \quad & A & \rightarrow \ldots B \ldots & \quad & s & \quad & r & \quad & 2s - r \\
  B & \rightarrow XY & \quad & B & \rightarrow XY & \quad & -1 & \quad & 0 & \quad & -2
  \end{align*}
  \]

- **Digram Uniqueness** - create a new rule.

  \[
  \begin{align*}
  A & \rightarrow \ldots XY \ldots & \quad & A & \rightarrow \ldots C \ldots & \quad & s & \quad & r & \quad & 2s - r \\
  B & \rightarrow \ldots XY \ldots & \quad & B & \rightarrow \ldots C \ldots & \quad & 0 & \quad & 1 & \quad & -1 \\
  C & \rightarrow XY
  \end{align*}
  \]

- **Rule Utility** - Remove a rule.

  \[
  \begin{align*}
  A & \rightarrow \ldots B \ldots & \quad & A & \rightarrow \ldots X_1X_2\ldots X_k \ldots & \quad & s & \quad & r & \quad & 2s - r \\
  B & \rightarrow X_1X_2\ldots X_k & \quad & A & \rightarrow \ldots X_1X_2\ldots X_k \ldots & \quad & -1 & \quad & -1 & \quad & -1
  \end{align*}
  \]
Amortized Complexity Argument

- $2s - r \geq 0$ 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.
- $n = \text{number of input symbols}$
  
  $m = \text{number of non-input operations}$
- $2n - m \geq 0. \ m \leq 2n.$
Amortized Complexity Argument

![Graph showing the amortized complexity argument with two lines representing 'Input' and 'Non-input' over time. The y-axis is labeled '2s - r' and the x-axis is labeled 'Time'.]
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

current digram

digram table

reference count
Basic Encoding a Grammar

Grammar

S → DBD
A → be
B → AA
D → bBe

Symbol Code

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

Grammar Code

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

|Grammar Code| = (s + r − 1)\left\lceil \log_2 (r + a) \right\rceil

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

Lecture 6 - Dictionary Coding
Better Encoding of the Grammar

• Nevill-Manning and Witten suggest a more efficient encoding of the grammar that uses LZ77 ideas.
Kieffer-Yang Improvement

• Kieffer and Yang
  – Eliminate rules that are redundant
  – KY is universal; it achieves entropy in the limit

• Add to sequitur Reduction Rule 5:

\[
\begin{align*}
S \to AB \\
A \to CD \\
B \to aE \\
C \to ab \\
D \to cd \\
E \to bD
\end{align*}
\]

\[
\begin{align*}
S \to AA \\
A \to CD \\
B \to aE \\
C \to ab \\
D \to cd \\
E \to bD
\end{align*}
\]

Adding this constraint makes sequitur universal.

\[<A> = <B> = abcd\]
Other Grammar Based Methods

• Longest Match
• Most frequent digram
• Match producing the best compression
Notes on Sequitur

- Yields compression and hierarchical structure simultaneously.
- With clever encoding is competitive with the best of the standards.
Move-to-Front Coding

- Non-numerical data
- The data have a relatively small working set that changes over the sequence.
- Example: a b a b a a b c c b b c c c c b d b c c
- Move-to-front coding allows data with a small working set to be transformed to data with better statistics for entropy coding.
Move-to-Front Algorithm

• Move-to-Front
  – Symbols are kept in a list indexed 0 to m-1
  – To code a symbol output its index and move the symbol to the front of the list
  – The index stream is entropy coded using arithmetic coding or some other statistical technique
Example

Example: $a\ b\ a\ b\ a\ a\ b\ c\ c\ b\ b\ c\ c\ c\ c\ b\ d\ b\ c\ c$

0

0 1 2 3
a b c d
Example

- Example: \textcolor{red}{a b} a b a b c c b b c c c c b d b c c

\begin{align*}
0 & \quad 1 & \quad 2 & \quad 3 \\
\text{a} & \quad \text{b} & \quad \text{c} & \quad \text{d} \\
\downarrow \\
0 & \quad 1 & \quad 2 & \quad 3 \\
\text{b} & \quad \text{a} & \quad \text{c} & \quad \text{d}
\end{align*}
Example

- Example: \textcolor{red}{a b a} b a a b c c b b c c c c b d b c c
  
  0 1 1

\begin{align*}
  &0 1 2 3 \\
  &b \ a \ c \ d \\
  &\downarrow \\
  &0 1 2 3 \\
  &a \ b \ c \ d
\end{align*}
Example

- Example: \( \text{a b a b a a b c c b b c c c c b d b c c} \)
  
  \[
  \begin{array}{cccc}
  0 & 1 & 1 & 1 \\
  \text{a} & \text{b} & \text{c} & \text{d} \\
  \end{array}
  \]

  \[
  \begin{array}{cccc}
  0 & 1 & 2 & 3 \\
  \text{b} & \text{a} & \text{c} & \text{d} \\
  \end{array}
  \]
Example

Example: \textbf{a b a b a} a b c c b b c c c c b d b c c

\begin{tabular}{cccc}
0 & 1 & 1 & 1 \\
\end{tabular}

\begin{tabular}{cccc}
0 & 1 & 2 & 3 \\
b & a & c & d \\
\end{tabular}

\begin{tabular}{cccc}
0 & 1 & 2 & 3 \\
a & b & c & d \\
\end{tabular}
Example

- Example: \textcolor{red}{a} \textcolor{red}{b} \textcolor{red}{a} \textcolor{red}{b} \textcolor{red}{a} \textcolor{red}{a} \textcolor{red}{b} \textcolor{red}{c} \textcolor{red}{c} \textcolor{red}{b} \textcolor{red}{b} \textcolor{red}{c} \textcolor{red}{c} \textcolor{red}{c} \textcolor{red}{c} \textcolor{red}{b} \textcolor{red}{d} \textcolor{red}{b} \textcolor{red}{c} \textcolor{red}{c}

0 1 1 1 1 0

0 1 2 3
\textcolor{red}{a} \textcolor{red}{b} \textcolor{red}{c} \textcolor{red}{d}
Example

- Example: \texttt{a b a b a a b c c b b c c c c b d b c c 0 1 1 1 1 0 1}

\begin{center}
\begin{tabular}{cccc}
0 & 1 & 2 & 3 \\
a & b & c & d \\
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{cccc}
0 & 1 & 2 & 3 \\
b & a & c & d \\
\end{tabular}
\end{center}
Example

• Example: a b a b a a b c c b c c c c b d b c c

  0 1 1 1 1 0 1 2

  0 1 2 3
  b a c d

  0 1 2 3
  c b a d
Example

- Example: \texttt{a b a b a a b c c b b c c c c c b d b c c}
  
  0 1 1 1 1 0 1 2 0 1 0 1 0 0 0 1 3 1 2 0

  0 1 2 3
  c b d a
Example

• Example: $a b a b a a b c c b b c c c c c b d b c c$

$0 1 1 1 1 0 1 2 0 1 0 1 0 0 0 1 3 1 2 0$

Frequencies of $\{a, b, c, d\}$

\[
\begin{array}{llll}
a & b & c & d \\
4 & 7 & 8 & 1 \\
\end{array}
\]

Entropy $= 1.74$

Frequencies of $\{0, 1, 2, 3\}$

\[
\begin{array}{llll}
0 & 1 & 2 & 3 \\
8 & 9 & 2 & 1 \\
\end{array}
\]

Entropy $= 1.6$
Extreme Example

Input:
aaaaaaaaaaaaaaaaabbbbbbbbbccccccccccccccdddddttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttt
Burrows-Wheeler Transform

- Burrows-Wheeler, 1994
- BW Transform creates a representation of the data which has a small working set.
- The transformed data is compressed with move to front compression.
- The decoder is quite different from the encoder.
- The algorithm requires processing the entire string at once (it is not on-line).
- It is a remarkably good compression method.
Encoding Example

• abracadabra
  1. Create all cyclic shifts of the string.

<table>
<thead>
<tr>
<th>Shift</th>
<th>Shifted String</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>abracadabra</td>
</tr>
<tr>
<td>1</td>
<td>bracadabraa</td>
</tr>
<tr>
<td>2</td>
<td>racadabraab</td>
</tr>
<tr>
<td>3</td>
<td>acadabraabr</td>
</tr>
<tr>
<td>4</td>
<td>cadabraabra</td>
</tr>
<tr>
<td>5</td>
<td>adabraabrac</td>
</tr>
<tr>
<td>6</td>
<td>dabraabraca</td>
</tr>
<tr>
<td>7</td>
<td>abraabracad</td>
</tr>
<tr>
<td>8</td>
<td>braabracada</td>
</tr>
<tr>
<td>9</td>
<td>raabracadab</td>
</tr>
<tr>
<td>10</td>
<td>aabracadabr</td>
</tr>
</tbody>
</table>
Encoding Example

2. Sort the strings alphabetically into array A

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>abracadabra</td>
<td>A</td>
<td>aabracadabr</td>
</tr>
<tr>
<td>1</td>
<td>bracadabraaa</td>
<td>1</td>
<td>abraabracad</td>
</tr>
<tr>
<td>2</td>
<td>racadabraab</td>
<td>2</td>
<td>abracadabra</td>
</tr>
<tr>
<td>3</td>
<td>acadabraabr</td>
<td>3</td>
<td>acadabraabr</td>
</tr>
<tr>
<td>4</td>
<td>cadabraabra</td>
<td>4</td>
<td>dabababraabr</td>
</tr>
<tr>
<td>5</td>
<td>dababraabrac</td>
<td>5</td>
<td>braabracada</td>
</tr>
<tr>
<td>6</td>
<td>dababraacraca</td>
<td>6</td>
<td>bracadabraa</td>
</tr>
<tr>
<td>7</td>
<td>abraabracad</td>
<td>7</td>
<td>cadabraabra</td>
</tr>
<tr>
<td>8</td>
<td>braabracada</td>
<td>8</td>
<td>dababraabraca</td>
</tr>
<tr>
<td>9</td>
<td>raabracadab</td>
<td>9</td>
<td>raabracadab</td>
</tr>
<tr>
<td>10</td>
<td>aabracadab</td>
<td>10</td>
<td>racadabraab</td>
</tr>
</tbody>
</table>
Encoding Example

3. $L = \text{the last column}$

$$A$$

|   | aabracadabr | 1   | abraabracad | 2   | abracadabra | 3   | acadabraabr | 4   | adabraabrac | 5   | braabraabra  | 6   | bracradabraa  | 7   | cadabraabra  | 8   | dabrabraabra | 9   | raabracadab | 10  | racadabraab |
|---|-------------|-----|-------------|-----|-------------|-----|-------------|-----|-------------|-----|-------------|-----|--------------|-----|--------------|-----|-------------|-----|-------------|
|   |             | 0   |             | 1   |             | 2   |             | 3   |             | 4   |             | 5   |             | 6   |             | 7   |             | 8   |             | 9   |             |

$L = \text{rdarcaaaabb}$
1. Encoding Example

4. Transmit $X$ the index of the input in $A$ and $L$ (using a predictive coding scheme).

<table>
<thead>
<tr>
<th>0</th>
<th>aabracadabr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abraabracad</td>
</tr>
<tr>
<td>2</td>
<td><strong>abracadabra</strong></td>
</tr>
<tr>
<td>3</td>
<td>acadabraabr</td>
</tr>
<tr>
<td>4</td>
<td>adabraabrac</td>
</tr>
<tr>
<td>5</td>
<td>braabraabrac</td>
</tr>
<tr>
<td>6</td>
<td>bracadabraaa</td>
</tr>
<tr>
<td>7</td>
<td>cadabraabra</td>
</tr>
<tr>
<td>8</td>
<td>dabraabracad</td>
</tr>
<tr>
<td>9</td>
<td>raabracadab</td>
</tr>
<tr>
<td>10</td>
<td>racadabraab</td>
</tr>
</tbody>
</table>

$A$

$L = \text{rdarcaaaabb}$

$X = 2$
Why BW Works

• Ignore decoding for the moment.

• The prefix of each shifted string is a context for the last symbol.
  – The last symbol appears just before the prefix in the original.

• By sorting similar contexts are adjacent.
  – This means that the predicted last symbols are similar.
Decoding Example

- We first decode assuming some information. We then show how to compute the information.
- Let $A^s$ be $A$ shifted by 1

<table>
<thead>
<tr>
<th>A</th>
<th>$A^s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>aabracadab</td>
</tr>
<tr>
<td>1</td>
<td>abraabracad</td>
</tr>
<tr>
<td>2</td>
<td>abracadabra</td>
</tr>
<tr>
<td>3</td>
<td>acadabraabr</td>
</tr>
<tr>
<td>4</td>
<td>adabraabraacr</td>
</tr>
<tr>
<td>5</td>
<td>braabraacadada</td>
</tr>
<tr>
<td>6</td>
<td>bracadabraa</td>
</tr>
<tr>
<td>7</td>
<td>cadabraabraa</td>
</tr>
<tr>
<td>8</td>
<td>dabraabraacad</td>
</tr>
<tr>
<td>9</td>
<td>raabracadabab</td>
</tr>
<tr>
<td>10</td>
<td>raadabraabraab</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A^s$</th>
<th>$A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>raabracadab</td>
</tr>
<tr>
<td>1</td>
<td>dababraabraac</td>
</tr>
<tr>
<td>2</td>
<td>aabracadabra</td>
</tr>
<tr>
<td>3</td>
<td>racadabraaab</td>
</tr>
<tr>
<td>4</td>
<td>cadabraabraab</td>
</tr>
<tr>
<td>5</td>
<td>ababraabraacd</td>
</tr>
<tr>
<td>6</td>
<td>abracadabraa</td>
</tr>
<tr>
<td>7</td>
<td>cadabraabraa</td>
</tr>
<tr>
<td>8</td>
<td>dababraabraac</td>
</tr>
<tr>
<td>9</td>
<td>braabraacadada</td>
</tr>
<tr>
<td>10</td>
<td>bracadabraa</td>
</tr>
</tbody>
</table>
Decoding Example

• Assume we know the mapping $T[i]$ is the index in $A^s$ of the string $i$ in $A$.
• $T = [2 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 4 \ 1 \ 0 \ 3]$

<table>
<thead>
<tr>
<th>A</th>
<th>$A^s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 aabracadabr</td>
<td>0 raabracadab</td>
</tr>
<tr>
<td>1 abraabracad</td>
<td>1 dabraabracad</td>
</tr>
<tr>
<td>2 abracadabra</td>
<td>2 aabracadabr</td>
</tr>
<tr>
<td>3 acadabraabr</td>
<td>3 racadabraabr</td>
</tr>
<tr>
<td>4 adabraabrac</td>
<td>4 cadabraabra</td>
</tr>
<tr>
<td>5 braabracada</td>
<td>5 abraabracad</td>
</tr>
<tr>
<td>6 bracadabraa</td>
<td>6 abracadabra</td>
</tr>
<tr>
<td>7 cadabraabra</td>
<td>7 acadabraabr</td>
</tr>
<tr>
<td>8 dabraabrac</td>
<td>8 adabraabrac</td>
</tr>
<tr>
<td>9 raabracadab</td>
<td>9 braabracada</td>
</tr>
<tr>
<td>10 racadabraab</td>
<td>10 bracadabraa</td>
</tr>
</tbody>
</table>
Decoding Example

• Let F be the first column of A, it is just L, sorted.

\[
F = \begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{b} & \text{b} & \text{c} & \text{d} & \text{r} & \text{r}
\end{array}
\]

\[
T = \begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
2 & 5 & 6 & 7 & 8 & 9 & 10 & 4 & 1 & 0 & 3
\end{array}
\]

• Follow the pointers in T in F to recover the input starting with X.
Decoding Example

\[ F = \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{b} & \text{b} & \text{c} & \text{d} & \text{r} & \text{r} \\
\end{array} \]

\[ T = \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
2 & 5 & 6 & 7 & 8 & 9 & 10 & 4 & 1 & 0 & 3 \\
\end{array} \]

\text{a}

Lecture 6 - Dictionary Coding
Decoding Example

\[
\begin{align*}
F &= \begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{b} & \text{b} & \text{c} & \text{d} & \text{r} & \text{r}
\end{array} \\
T &= \begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
2 & 5 & 6 & 7 & 8 & 9 & 10 & 4 & 1 & 0 & 3
\end{array}
\end{align*}
\]

ab
Decoding Example

\[
F = \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{b} & \text{b} & \text{c} & \text{d} & \text{r} & \text{r}
\end{array}
\]

\[
T = \begin{array}{cccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
2 & 5 & 6 & 7 & 8 & 9 & 10 & 4 & 1 & 0 & 3
\end{array}
\]

abr
Decoding Example

• Why does this work?
• The first symbol of $A[T[i]]$ is the second symbol of $A[i]$ because $A_s[T[i]] = A[i]$.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$T$</th>
<th>$A_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>aabracadabr</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>abraabracad</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td><strong>abracadaabra</strong></td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>acadabraabracadab</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>adabraabra</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>braabra</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>bracadaabraa</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>cadabraabra</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>dabraabra</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>raabracadab</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>racadabraa</td>
<td>3</td>
</tr>
</tbody>
</table>
Decoding Example

• How do we compute F and T from L and X?

  F is just L sorted

  \[
  \begin{align*}
  &0 1 2 3 4 5 6 7 8 9 10 \\
  F &= a a a a a b b b c d r r \\
  L &= r d a r c a a a a b b 
  \end{align*}
  \]

  Note that L is the first column of \( A^s \) and \( A^s \) is in the same order as A.

  If i is the k-th x in F then T[i] is the k-th x in L.
Decoding Example

\[
\begin{align*}
F &= 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \\
   &\quad \ a \ a \ a \ a \ a \ a \ b \ b \ c \ d \ r \ r \\
L &= r \ d \ a \ r \ c \ a \ a \ a \ a \ b \ b \\
T &= 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \\
   &\quad \ 2 \ 5 \ 6 \ 7 \ 8
\end{align*}
\]
Decoding Example

F = a a a a a b b c d r r
L = r d a r c a a a a a b b
T = 0 1 2 3 4 5 6 7 8 9 10
    2 5 6 7 8 9 10
Decoding Example

\[ F = \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array} \]
\[ L = \begin{array}{cccccccccccc}
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{b} & \text{b} & \text{c} & \text{d} & \text{r} & \text{r} \\
\end{array} \]
\[ T = \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
2 & 5 & 6 & 7 & 8 & 9 & 10 & 4 \\
\end{array} \]
Decoding Example

\[ F = \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{b} & \text{b} & \text{c} & \text{d} & \text{r} & \text{r} \\
\end{array} \]

\[ L = \begin{array}{cccccccccccc}
\text{r} & \text{d} & \text{a} & \text{r} & \text{c} & \text{a} & \text{a} & \text{a} & \text{a} & \text{b} & \text{b} \\
\end{array} \]

\[ T = \begin{array}{ccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
2 & 5 & 6 & 7 & 8 & 9 & 10 & 4 & 1 \\
\end{array} \]
Decoding Example

\[
\begin{align*}
F &= \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
 a & a & a & a & a & b & b & b & c & d & r & r \\
\end{array} \\
L &= \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
 r & d & a & r & c & a & a & a & a & b & b \\
\end{array} \\
T &= \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
 2 & 5 & 6 & 7 & 8 & 9 & 10 & 4 & 1 & 0 & 3 \\
\end{array}
\]
BWT Encoding Exercise

Encode the string ababababababababababab = (ab)^8
1. Find L and X
BWT Decoding Exercise

Decode $L = \text{baaaaaba}$, $X = 6$

1. First Compute $F$ and $T$
2. Use those to decode.
Notes on BW

• Alphabetic sorting does not need the entire cyclic shifted inputs.
  – Sort the indices of the string
  – Most significant symbols first radix sort works

• There are high quality practical implementations
  – Bzip
  – Bzip2
## Compression Quality

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

- **First** = First;
- **Second** = Second;
- **Third** = Third.

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  
bzip2 – Burrows-Wheeler block sorting