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

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

Plan for Tonight

- Overview
- LZW
- Sequitur
- Move-to-front coding
- Burrows-Wheeler Transform

Overview of Dictionary Compression

LZW Encoding Example (1)

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>a b a b a b a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 a</td>
<td></td>
</tr>
<tr>
<td>1 b</td>
<td></td>
</tr>
</tbody>
</table>
LZW Encoding Example (2)

Dictionary
0 a
1 b
2 ab

LZW Encoding Example (3)

Dictionary
0 a
1 b
2 ab
3 ba

LZW Encoding Example (4)

Dictionary
0 a
1 b
2 ab
3 ba
4 abba

LZW Encoding Example (5)

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

LZW Encoding Example (6)

Dictionary
0 a
1 b
2 ab
3 ba
4 abba
5 abab

LZW Dictionary Derivation Algorithm

• Emulate the encoder in building the dictionary.
  Decoder is slightly behind the encoder.

initialize dictionary;
decode first index to w;
push 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;
push w? in the dictionary where w was just decoded;
LZW Decoding Example (1)

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>

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>

LZW Decoding Example (2b)

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>
</tbody>
</table>

LZW Decoding Example (3a)

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>aab</td>
</tr>
</tbody>
</table>

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>

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>
### 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>abab</td>
</tr>
</tbody>
</table>

**Decoding:**

012436  
01 01 01 01 01 01 01 01 01 01 01 01

### 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>

**Decoding:**

012436  
01 01 01 01 01 01 01 01 01 01 01 01

### LZW Decoding Example (5b)

**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>
<tr>
<td>6</td>
<td>bab</td>
</tr>
</tbody>
</table>

**Decoding:**

012436  
01 01 01 01 01 01 01 01 01 01 01 01

### LZW Decoding Example (6a)

**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>
<tr>
<td>6</td>
<td>bab</td>
</tr>
</tbody>
</table>

**Decoding:**

012436  
01 01 01 01 01 01 01 01 01 01 01 01

### LZW Decoding Example (6b)

**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>
<tr>
<td>6</td>
<td>bab</td>
</tr>
<tr>
<td>7</td>
<td>bab?</td>
</tr>
</tbody>
</table>

**Decoding:**

012436  
01 01 01 01 01 01 01 01 01 01 01 01

### Decoding Exercise

**Base 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>c</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
</tr>
<tr>
<td>4</td>
<td>r</td>
</tr>
</tbody>
</table>

**Decoding:**

014020357
Trie Data Structure for Encoder’s Dictionary

- Fredkin (1960)

```
Encoder Uses a Trie (1)
```

```
Encoder Uses a Trie (2)
```

Decoder’s Data Structure

- Simply an array of strings

```
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 → 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

Arithmetic Expressions

- S → S + T
  - S → T
  - T → T * F
  - T → F
  - F → a
  - F → ( S )
- Derivation of a * ( a + a ) + a

Overview of Grammar Compression

- 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)

$\texttt{bebebebebebee}$

$S \rightarrow b$

### Sequitur Example (2)

$\texttt{bebebebebebee}$

$S \rightarrow bb$

### Sequitur Example (3)

$\texttt{bebebebebebee}$

$S \rightarrow bbe$

### Sequitur Example (4)

$\texttt{bebebebebebee}$

$S \rightarrow bbbe$

### Sequitur Example (5)

$\texttt{bebebebebebee}$

$S \rightarrow bbe$

- Enforce digram uniqueness.
- A occurs twice.
- Create new rule $A \rightarrow be$.  

### Sequitur Example (6)

$\texttt{bebebebebebee}$

$S \rightarrow bAA$

$A \rightarrow be$
Sequitur Example (7)

\[ b\text{b}e\text{b}\text{b}e\text{b}e\text{b}\text{b}e\text{b}\text{b}e\text{b} \]

\[ S \rightarrow bAAe \]
\[ A \rightarrow be \]

Sequitur Example (8)

\[ b\text{b}eb\text{e}e\text{b}e\text{b}e\text{b}e\text{b} \]

\[ S \rightarrow bAAeb \]
\[ A \rightarrow be \]

Enforce diagram uniqueness.
be occurs twice.
Use existing rule \( A \rightarrow be \).

Sequitur Example (9)

\[ b\text{b}eb\text{e}e\text{b}e\text{b}e\text{b}e\text{b} \]

\[ S \rightarrow bAAebe \]
\[ A \rightarrow be \]

Enforce diagram uniqueness.
be occurs twice.
Use existing rule \( A \rightarrow be \).

Sequitur Example (10)

\[ b\text{b}eb\text{e}e\text{b}e\text{b}e\text{b}e\text{b} \]

\[ S \rightarrow bAAeA \]
\[ A \rightarrow be \]

Enforce diagram uniqueness.
be occurs twice.
Use existing rule \( A \rightarrow be \).

Sequitur Example (11)

\[ b\text{b}eb\text{e}e\text{b}e\text{b}e\text{b}e\text{b} \]

\[ S \rightarrow bAAeAb \]
\[ A \rightarrow be \]

Sequitur Example (12)

\[ b\text{b}eb\text{e}e\text{b}e\text{b}e\text{b}e\text{b} \]

\[ S \rightarrow bAAeAb \]
\[ A \rightarrow be \]
Sequitur Example (13)

```
bbbeeebebebebebe
```

- **S → bBeA**
- **A → be**

- Enforce diagram uniqueness.
- AA occurs twice.
- Create new rule B → AA.

Sequitur Example (14)

```
bbbeeebebebebebe
```

- **S → bBeB**
- **A → be**
- **B → AA**

- Enforce diagram uniqueness.
- AA occurs twice.

Sequitur Example (15)

```
bbbeeebebebebebebe
```

- **S → bBeBb**
- **A → be**
- **B → AA**

- Enforce diagram uniqueness.
- Be occurs twice.

Sequitur Example (16)

```
bbbeeebebebebebebe
```

- **S → bBeBbb**
- **A → be**
- **B → AA**

- Enforce diagram uniqueness.
- Be occurs twice.

Sequitur Example (17)

```
bbbeeebebebebebebe
```

- **S → bBeBbe**
- **A → be**
- **B → AA**

- Enforce diagram uniqueness.
- Be occurs twice.

Sequitur Example (18)

```
bbbeeebebebebebebe
```

- **S → bBeBbA**
- **A → be**
- **B → AA**

- Use existing rule A → be.
Sequitur Example (19)

```
S → bBbBbAb
A → be
B → AA
```

Enforce diagram uniqueness.

Sequitur Example (20)

```
S → bBbBbAb
A → be
B → AA
```

be occurs twice.

Use existing rule A → be.

Sequitur Example (21)

```
S → bBbBbAA
A → be
B → AA
```

AA occurs twice.

Use existing rule B → AA.

Sequitur Example (22)

```
S → bBbBbBb
A → be
B → AA
```

bB occurs twice.

Create new rule C → bB.

Sequitur Example (23)

```
S → CeBC
A → be
B → AA
C → bB
```

Sequitur Example (24)

```
S → CeBCe
A → be
B → AA
C → bB
```

Ce occurs twice.

Create new rule D → Ce.
Sequitur Example (25)

\[
\begin{align*}
S &\rightarrow DBD \\
A &\rightarrow be \\
B &\rightarrow AA \\
C &\rightarrow bB \\
D &\rightarrow bbe \\
\end{align*}
\]

Enforce rule utility.

C occurs only once.

Remove C \rightarrow bB.

The Hierarchy

\[
\begin{align*}
S &\rightarrow DBD \\
A &\rightarrow be \\
B &\rightarrow AA \\
D &\rightarrow bbe \\
\end{align*}
\]

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

Sequitur Algorithm

Input the first symbol 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:
  \[ C \rightarrow XY \] \[ B \rightarrow \ldots C \ldots \]

- Input a new symbol:
  \[ A \rightarrow \ldots B \ldots \] \[ A \rightarrow \ldots X_i \ldots X_s \ldots \]
  \[ B \rightarrow X_i X_j \ldots X_s \] \[ S \rightarrow X_i X_j \ldots X_s \]

Until no symbols left.

Exercise

Use Sequitur to construct a grammar for aaaaaaaaaa = 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.
  \[
  A \rightarrow \ldots XY \ldots \quad B \rightarrow XY
  \]
  \[
  s \quad r \quad 2s - r
  \]
- Digram Uniqueness - create a new rule.
  \[
  A \rightarrow \ldots XY \ldots \quad B \rightarrow \ldots XY \ldots
  \]
  \[
  s \quad r \quad 2s - r
  \]
- Rule Utility - Remove a rule.
  \[
  A \rightarrow \ldots \ B \ldots \quad B \rightarrow X_1X_2\ldots X_k
  \]
  \[
  s \quad r \quad 2s - r
  \]

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.
- \( n \) = number of input symbols
- \( m \) = number of non-input operations
- \( 2n - m \geq 0 \), \( m \leq 2n \).

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
Basic Encoding a Grammar

Grammar

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>000</td>
</tr>
<tr>
<td>A</td>
<td>001</td>
</tr>
<tr>
<td>B</td>
<td>010</td>
</tr>
<tr>
<td>D</td>
<td>011</td>
</tr>
<tr>
<td>#</td>
<td>100</td>
</tr>
</tbody>
</table>

Symbol Code

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>000</td>
</tr>
<tr>
<td>e</td>
<td>001</td>
</tr>
<tr>
<td>A</td>
<td>010</td>
</tr>
<tr>
<td>B</td>
<td>011</td>
</tr>
<tr>
<td>D</td>
<td>100</td>
</tr>
<tr>
<td>#</td>
<td>101</td>
</tr>
</tbody>
</table>

Grammar Code

D B D # b e # A A # b B e 100 011 100 101 000 001 101 010 101 000 011 001 39 bits

\[ |\text{Grammar Code}| = \left( \frac{s + r - 1}{r} \right) \log_2 (r + a) \]

\[ r = \text{number of rules} \]

\[ s = \text{sum of right hand sides} \]

\[ a = \text{number in original symbol alphabet} \]

Better Encoding of the Grammar

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

Kieffer-Yang Improvement

• Kieffer and Yang
  – Eliminate rules that are redundant
  – KY is universal; it achieves entropy in the limit
• Add to seqitur Reduction Rule 5:

  - S → AB
  - A → CD
  - B → aE
  - C → ab
  - D → cd
  - E → cd

  <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 b b 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: \( a \ b \ a \ a \ b \ c \ b \ c \ c \ c \ b \ c \ b \ c \ c \)
  - Index: 0 1 2 3
  - Symbols: a b c d

Example

- Example: \( a \ b \ a \ a \ b \ c \ b \ c \ c \ c \ b \ c \ b \ c \ c \)
  - Index: 0 1 2 3
  - Symbols: a b c d

Example

- Example: \( a \ b \ a \ a \ b \ c \ b \ c \ c \ c \ b \ c \ b \ c \ c \)
  - Index: 0 1 2 3
  - Symbols: a b c d

Example

- Example: \( a \ b \ a \ a \ b \ c \ b \ c \ c \ c \ b \ c \ b \ c \ c \)
  - Index: 0 1 2 3
  - Symbols: a b c d
Example

• Example: \texttt{ababa bccbccbbccbbcbc}

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

Example

• Example: \texttt{ababa abcbbccbbbcc}

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

Example

• Example: \texttt{ababa bccbccbbccbbcbc}

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

Example

• Example: \texttt{ababa abccbcbbc}

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

Example

• Example: \texttt{ababa bccbccbbccbbccc}

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

Example

• Example: \texttt{ababa abccbcbbc}

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

Example

• Example: \texttt{ababa abccbcbbc}

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

Example

• Example: \texttt{ababa abccbcbbc}

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

Example

• Example: \texttt{ababa abccbcbbc}

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

Example

• Example: \texttt{ababa abccbcbbc}

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

Example

• Example: \texttt{ababa abccbcbbc}

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

Example

• Example: \texttt{ababa abccbcbbc}

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

1. Create all cyclic shifts of the string.

   \[ 0 \quad \text{abracadabra} \quad 1 \quad \text{bracadabra} \quad 2 \quad \text{racadabraab} \quad 3 \quad \text{cadabraabr} \quad 4 \quad \text{nadabraabr} \quad 5 \quad \text{dabraabrac} \quad 6 \quad \text{abraabraac} \quad 7 \quad \text{abraabraac} \quad 8 \quad \text{braabraacda} \quad 9 \quad \text{raabraacab} \quad 10 \quad \text{aabraacabra} \]

2. Sort the strings alphabetically into array A

   \[
   \begin{array}{ll}
   0 & \text{abraabraac} \\
   1 & \text{abracadabra} \\
   2 & \text{cadabraabr} \\
   3 & \text{acadabraabr} \\
   4 & \text{dabraabrac} \\
   5 & \text{abraabraac} \\
   6 & \text{braabraacda} \\
   7 & \text{abraabraac} \\
   8 & \text{braabraacda} \\
   9 & \text{raabraacab} \\
   10 & \text{aabraacabra} \\
   \end{array}
   \]

3. \( L = \) the last column

   \[ A = \begin{array}{ll}
   0 & \text{abraabraac} \\
   1 & \text{abracadabra} \\
   2 & \text{cadabraabr} \\
   3 & \text{acadabraabr} \\
   4 & \text{dabraabrac} \\
   5 & \text{abraabraac} \\
   6 & \text{braabraacda} \\
   7 & \text{abraabraac} \\
   8 & \text{braabraacda} \\
   9 & \text{raabraacab} \\
   10 & \text{aabraacabra} \\
   \end{array}
   \]

   \[ L = \text{rdarcaaaabb} \]

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

   \[ A = \begin{array}{ll}
   0 & \text{abraabraac} \\
   1 & \text{abracadabra} \\
   2 & \text{cadabraabr} \\
   3 & \text{acadabraabr} \\
   4 & \text{dabraabrac} \\
   5 & \text{abraabraac} \\
   6 & \text{braabraacda} \\
   7 & \text{abraabraac} \\
   8 & \text{braabraacda} \\
   9 & \text{raabraacab} \\
   10 & \text{aabraacabra} \\
   \end{array}
   \]

   \[ L = \text{rdarcaaaabb} \]

   \[ X = 2 \]

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 compute the information.
- Let $A^s$ be $A$ shifted by 1

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

Decoding Example

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

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

Decoding Example

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

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

- Follow the pointers in $T$ in $F$ to recover the input starting with $X$. 

Decoding Example

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

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

A
0  abracadabra 2 0  raabracadab
1  abraabra 5 1  dabrabracas
2  abracadabra 6 2  abracadabr
3  acadabraabr 7 3  racadabraab
4  adabraabra 8 4  cadabraabra
5  braabracadas 9 5  abraabraac
6  bracadabraas 10 6  abracadabra
7  cadabraabra 4 7  acadabraabr
8  dababrabras 1 8  adabraabra
9  raabraabracad 0 9  braabracada
10  racadabraab 3 10  bracadabraa

Decoding Example

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

F = 0 1 2 3 4 5 6 7 8 9 10
L = rdarcaaaaaabb

Note that L is the first column of A^t and A^t 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

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

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

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

BWT Encoding Exercise

Encode the string \(abababababababab\) = (ab)

1. Find \(L\) and \(X\)

BWT Decoding Exercise

Decode \(L = baaaaaba, X = 6\)

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>file</th>
<th>size</th>
<th>comp</th>
<th>gap</th>
<th>sequenc</th>
<th>PPMC</th>
<th>bzip2</th>
</tr>
</thead>
<tbody>
<tr>
<td>bb</td>
<td>111281</td>
<td>3.35</td>
<td>2.51</td>
<td>2.48</td>
<td>2.12</td>
<td>1.98</td>
</tr>
<tr>
<td>block</td>
<td>76771</td>
<td>3.46</td>
<td>3.35</td>
<td>2.82</td>
<td>2.52</td>
<td>2.42</td>
</tr>
<tr>
<td>pcd</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>db2</td>
<td>246814</td>
<td>4.17</td>
<td>2.62</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>98611</td>
<td>3.97</td>
<td>2.68</td>
<td>2.93</td>
<td>2.40</td>
<td>2.63</td>
</tr>
</tbody>
</table>

Files from the Calgary Corpus
Units in bits per character (8 bits)
comp = based on LZW
gap = based on LZ77
PPMC = adaptive arithmetic coding with context
bzip2 = Burrows-Wheeler block sorting