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
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

LZW Encoding Example (2)

Dictionary
0 a
1 b
2 ab

a b a b a b a

LZW Encoding Example (3)

Dictionary
0 a
1 b
2 ab
3 ba

a b a b a b a
LZW Encoding Example (4)

Dictionary
0  a
1  b
2  ab
3  ba
4  aba

LZW Encoding Example (5)

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

LZW Encoding Example (6)

Dictionary
0  a
1  b
2  ab
3  ba
4  aba
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; 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?

\[ \begin{array}{c|ccccc}
0 & 1 & 2 & 3 & 4 & 6 \\
\hline
a & \end{array} \]

LZW Decoding Example (2a)

Dictionary

0  a
1  b
2  ab

\[ \begin{array}{c|ccccc}
0 & 1 & 2 & 4 & 3 & 6 \\
\hline
a & b & \end{array} \]

LZW Decoding Example (2b)

Dictionary

0  a
1  b
2  ab
3  b?

\[ \begin{array}{c|ccccc}
0 & 1 & 2 & 4 & 3 & 6 \\
\hline
a & b \end{array} \]

LZW Decoding Example (3a)

Dictionary

0  a
1  b
2  ab
3  ba

\[ \begin{array}{c|ccccc}
0 & 1 & 2 & 4 & 3 & 6 \\
\hline
a & b & a \end{array} \]
LZW Decoding Example (5b)

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>ba?</td>
</tr>
</tbody>
</table>

0 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 a b ab aba ba b

LZW Decoding Example (6b)

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 a b ab aba ba bab

Decoding Exercise

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

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

- Fredkin (1960)

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:
    - \( S \rightarrow SA, S \rightarrow A, A \rightarrow bSe, A \rightarrow be \)
    - \( S \) 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

**Arithmetic Expressions**

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

**Overview of Grammar Compression**

**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)

\[ bbebeebebebee \]

\[ S \rightarrow b \]

Sequitur Example (2)

\[ bbebeebebebee \]

\[ S \rightarrow bb \]

Sequitur Example (3)

\[ bbebeebebebee \]

\[ S \rightarrow bbe \]

Sequitur Example (4)

\[ bbebeebebebee \]

\[ S \rightarrow bbeb \]
Sequitur Example (5)

bbebebebebebebebebe

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

Sequitur Example (6)

bbebebebebebebebebe

$S \rightarrow bAA$

$A \rightarrow be$

Sequitur Example (7)

bbebebebebebebebebe

$S \rightarrow bAAe$

$A \rightarrow be$

Sequitur Example (8)

bbebebebebebebebebe

$S \rightarrow bAAb$

$A \rightarrow be$
Dictionary Coding 45

Sequitur Example (9)

bbebeebebbeebee

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

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

Dictionary Coding 46

Sequitur Example (10)

bbebeebebbeebee

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

Dictionary Coding 47

Sequitur Example (11)

bbebeebebbeebee

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

Dictionary Coding 48

Sequitur Example (12)

bbebeebebbeebee

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

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

bbebeebebe

S → bAAeAA
A → be

Enforce digram uniqueness
AA occurs twice.
Create new rule B → AA.

Sequitur Example (14)

bbebeebebe

S → bBeB
A → be
B → AA

Sequitur Example (15)

bbebeebebe

S → bBeBb
A → be
B → AA

Sequitur Example (16)

bbebeebebe

S → bBeBbb
A → be
B → AA
Sequitur Example (17)

bbebeebebebbee

S → bBeBb
A → be
B → AA

Enforce digram uniqueness.
be occurs twice. Use existing rule A → be.

Sequitur Example (18)

bbebeebebebbee

S → bBeBbA
A → be
B → AA

Sequitur Example (19)

bbebeebebebbee

S → bBeBbAb
A → be
B → AA

Sequitur Example (20)

bbebeebebebbee

S → bBeBbA
A → be
B → AA

Enforce digram uniqueness.
be occurs twice. Use existing rule A → be.
Sequitur Example (21)

```
 bbebeebebebebee
```

- **S → bBeBbAA**  
  Enforce digram uniqueness.
- **A → be**  
  AA occurs twice.
- **B → AA**  
  Use existing rule B → AA.

Sequitur Example (22)

```
 bbebeebebebebee
```

- **S → bEeBbB**  
  Enforce digram uniqueness.
- **A → be**  
  bB occurs twice.
- **B → AA**  
  Create new rule C → bB.

Sequitur Example (23)

```
 bbebeebebebebee
```

- **S → CeBC**  
  Enforce digram uniqueness.
- **A → be**  
  CeB occurs twice.
- **B → AA**
- **C → bB**

Sequitur Example (24)

```
 bbebeebebebebee
```

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

bbebeebebebebee

S → DBD
A → be
B → AA
C → bB
D → Ce

Enforce rule utility. C occurs only once. Remove C → bB.

The Hierarchy

bbebeebebebebee

S → DBD
A → be
B → AA
D → bBe

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

Sequitur Example (26)

bbebeebebebebee

S → DBD
A → be
B → AA
D → bBe

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....
remove a rule:
C → XY C → XY
input a symbol:
B → X1X2...Xk A → ....X1X2...Xk...
S → X1X2...Xk S → X1X2...Xk 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 ≤ 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.
  
  \[
  \begin{array}{ccc}
  A \rightarrow \ldots XY \ldots & \rightarrow & A \rightarrow \ldots B \ldots & s & r & 2s - r \\
  B \rightarrow XY & \rightarrow & B \rightarrow XY & -1 & 0 & -2 \\
  \end{array}
  \]

- Digram Uniqueness - create a new rule.
  
  \[
  \begin{array}{ccc}
  A \rightarrow \ldots XY \ldots & \rightarrow & A \rightarrow \ldots C \ldots & s & r & 2s - r \\
  B \rightarrow \ldots XY \ldots & \rightarrow & B \rightarrow \ldots C \ldots & 0 & 1 & -1 \\
  C \rightarrow XY & & & & & \\
  \end{array}
  \]

- Rule Utility - Remove a rule.
  
  \[
  \begin{array}{ccc}
  A \rightarrow \ldots B \ldots & \rightarrow & A \rightarrow \ldots X_1X_2\ldots X_k \ldots & s & r & 2s - r \\
  B \rightarrow X_1X_2\ldots X_k & \rightarrow & A \rightarrow \ldots X_1X_2\ldots X_k \ldots & -1 & -1 & -1 \\
  \end{array}
  \]
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.
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
39 bits

|Grammar Code| = (s + r − 1) [log₂(r + a)]

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.

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:

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

< 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 b c b c c b 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 b c b c c b c c b d b c c
  - 0
  - 0 1 2 3
  - a b c d
Example

- Example: \texttt{a b a b a b c b c b c c c b c d b c c}
  \[0 1 1 1 1 0\]
  \[
  \begin{array}{c}
  0 1 2 3 \\
  a b c d \\
  \end{array}
  \]

Example

- Example: \texttt{a b a b a b c b b c c c b c d b c c}
  \[0 1 1 1 1 0 1\]
  \[
  \begin{array}{c}
  0 1 2 3 \\
  a b c d \\
  \end{array}
  \]

Example

- Example: \texttt{a b a b a b c b c c b c d b c c}
  \[0 1 1 1 1 0 1 2\]
  \[
  \begin{array}{c}
  0 1 2 3 \\
  b a c d \\
  \end{array}
  \]

Example

- Example: \texttt{a b a b a b c b b c c c b c d b c c}
  \[0 1 1 1 1 0 1 2 0 1 0 1 0 0 1 3 1 2 0\]
  \[
  \begin{array}{c}
  0 1 2 3 \\
  c b d a \\
  \end{array}
  \]
Example

- Example: \( a b a b a b c b b c c c b c d b c c \)
  
  \[
  \begin{array}{c|c|c|c|c|c}
    \text{Frequency} & 0 & 1 & 1 & 1 & 0 \text{ 1 2 0 1 0 0 1 3 1 2 0} \\
  \hline
    \text{a} & 4 & 7 & 8 & 1 & \text{Entropy } = 1.74 \\
    \text{b} & 7 & 8 & 1 & 0 & \text{Entropy } = 1.6 \\
  \end{array}
  \]

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.

Extreme Example

Input:
\[
\text{aaaaaaaaaabbbbbbccccccccddddd}
\]

Output
\[
0000000001000000002000000030000000000000000000000000
\]

Frequencies of \(a b c d\)
\[
\begin{array}{c|c}
    0 & 1 2 3 \\
    4 & 1 0 1 0 1 0 0 1 3 1 2 0 \\
    \text{Entropy } = 2 \\
\end{array}
\]

Frequencies of \(0 1 2 3\)
\[
\begin{array}{c|c}
    0 & 1 2 3 \\
    8 & 9 2 1 \\
    \text{Entropy } = .5 \\
\end{array}
\]

Encoding Example

- abracadabra
  1. Create all cyclic shifts of the string.
  
  \[
  \begin{array}{c}
    0 \text{  abracadabra} \\
    1 \text{  bracadabra} \\
    2 \text{  racadabra} \\
    3 \text{  acadabra} \\
    4 \text{  cadabra} \\
    5 \text{  adabra} \\
    6 \text{  dabra} \\
    7 \text{  bra} \\
    8 \text{  ra} \\
    9 \text{  ra} \\
    10 \text{  a} \\
  \end{array}
  \]
Encoding Example
2. Sort the strings alphabetically into array A

<table>
<thead>
<tr>
<th></th>
<th>aabracadabr</th>
<th>abracadabra</th>
<th>abraabracad</th>
<th>abracadabra</th>
<th>abracadabra</th>
<th>abracadabra</th>
<th>abracadabra</th>
<th>abracadabra</th>
<th>abracadabra</th>
<th>abracadabra</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>aabracadabr</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>1</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>2</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>3</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>4</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>5</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>6</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>7</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>8</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>9</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>10</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
</tbody>
</table>

Dictionary Coding 93

Encoding Example
3. L = the last column

<table>
<thead>
<tr>
<th></th>
<th>aabracadabr</th>
<th>abracadabra</th>
<th>abraabracad</th>
<th>abracadabra</th>
<th>abracadabra</th>
<th>abracadabra</th>
<th>abracadabra</th>
<th>abracadabra</th>
<th>abracadabra</th>
<th>abracadabra</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>aabracadabr</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>1</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>2</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>3</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>4</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>5</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>6</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>7</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>8</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>9</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>10</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abraabracad</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
</tbody>
</table>

Dictionary Coding 94

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

```
A
0  aabracadabr
1  abraabracad
2  abracadabra
3  acadabraabr
4  cadabraabra
5  adabraabrac
6  dabraabraca
7  abraabracad
8  braabracada
9  raabracadab
10 aabracadabr
```

```
L = rdarcaaaabb
```

```
X = 2
```

Dictionary Coding 95

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.

Dictionary Coding 96
Decoding Example

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

<table>
<thead>
<tr>
<th>A</th>
<th>$A'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>aabracadab</td>
<td>raabracadab</td>
</tr>
<tr>
<td>abraabracab</td>
<td>dababraabra</td>
</tr>
<tr>
<td>abracadabra</td>
<td>aabracadabra</td>
</tr>
<tr>
<td>acadabrabr</td>
<td>cadabraabr</td>
</tr>
<tr>
<td>adabraabrac</td>
<td>dabraabrac</td>
</tr>
<tr>
<td>braabracada</td>
<td>raabracadab</td>
</tr>
<tr>
<td>bracadabraa</td>
<td>dabraabrac</td>
</tr>
<tr>
<td>cadabraabra</td>
<td>raabracadab</td>
</tr>
<tr>
<td>dabraabraca</td>
<td>aabracadabra</td>
</tr>
<tr>
<td>raabracadab</td>
<td>dababraabra</td>
</tr>
</tbody>
</table>

Decoding Example

- Assume we know the mapping $T[i]$ is the index in $A'$ 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>$A'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>aabracadab</td>
<td>raabracadab</td>
</tr>
<tr>
<td>abraabracab</td>
<td>dababraabra</td>
</tr>
<tr>
<td>abracadabra</td>
<td>aabracadabra</td>
</tr>
<tr>
<td>acadabrabr</td>
<td>cadabraabr</td>
</tr>
<tr>
<td>adabraabrac</td>
<td>dabraabrac</td>
</tr>
<tr>
<td>braabracada</td>
<td>raabracadab</td>
</tr>
<tr>
<td>bracadabraa</td>
<td>dabraabrac</td>
</tr>
<tr>
<td>cadabraabra</td>
<td>raabracadab</td>
</tr>
<tr>
<td>dabraabraca</td>
<td>aabracadabra</td>
</tr>
<tr>
<td>raabracadab</td>
<td>dababraabra</td>
</tr>
</tbody>
</table>

Decoding Example

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

<table>
<thead>
<tr>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
<tr>
<td>a a a a a b b c d r r</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
<tr>
<td>2 5 6 7 8 9 10 4 1 0 3</td>
</tr>
</tbody>
</table>

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

Decoding Example

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

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

\text{ab}

Why does this work?

• The first symbol of \( A[T[i]] \) is the second symbol of \( A[i] \)
  because \( A[T[i]] = A[i] \).

\begin{array}{cccccccccc}
A & T & A^s \\
0 & aabracadbr & 2 & \text{raabracadab} \\
1 & abraabracad & 5 & \text{daabraabra} \\
2 & abracabraab & 6 & \text{ababracad} \\
3 & acadabraabra & 7 & \text{acedabraab} \\
4 & adabraabra & 8 & \text{adabraabra} \\
5 & braabraabra & 9 & \text{abraabraa} \\
6 & bracadabraa & 10 & \text{abracadabra} \\
7 & cadabraabra & 4 & \text{acadabraa} \\
8 & dabraabra & 1 & \text{dabraabra} \\
9 & raabracadab & 0 & \text{braabraabra} \\
10 & racadabraa & 3 & \text{raabracadab}
\end{array}

Decoding Example

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

\begin{array}{cccccccccccc}
F & 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} \\

\begin{array}{cccccccccccc}
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}

\text{abr}

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

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

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

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

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

BWT Encoding Exercise

Encode the string \(\text{ababababababab} = (ab)^8\)
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>3.63</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>3861</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;  
- Second;  
- 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