CSEP 521
Applied Algorithms
Autumn 2009

Dictionary Coding for Lossless Compression

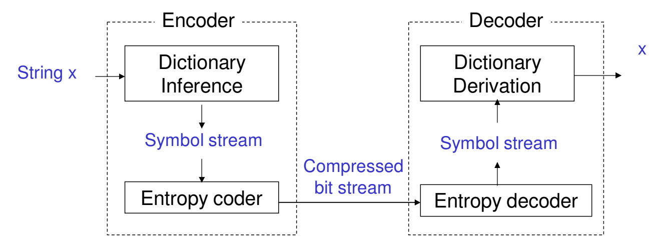
Plan for Tonight

- 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



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

0 a
1 b
2 ab
3 ba

a b a b a b a b a
0 1

LZW Encoding Example (4)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba

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

LZW Encoding Example (5)

Dictionary

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

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

LZW Encoding Example (6)

Dictionary

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

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

LZW 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

0 a
1 b
2 ab

0 1 2 4 3 6
a b

LZW Decoding Example (2b)

Dictionary

0 a
1 b
2 ab
3 b?

0 1 2 4 3 6
a b

LZW Decoding Example (3a)

Dictionary

0 a
1 b
2 ab
3 ba

0 1 2 4 3 6
a b a

LZW Decoding Example (3b)

Dictionary

0 a
1 b
2 ab
3 ba
4 ab?

0 1 2 4 3 6
a b ab

LZW Decoding Example (4a)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba

0 1 2 4 3 6
a b ab a

LZW Decoding Example (4b)

Dictionary

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

0 1 2 4 3 6
a b ab aba

LZW Decoding Example (5a)

Dictionary

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

0 1 2 4 3 6
a b ab aba b

LZW Decoding Example (5b)

Dictionary

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

0 1 2 4 3 6
a b ab aba ba

LZW Decoding Example (6a)

Dictionary

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

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

LZW Decoding Example (6b)

Dictionary

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

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

Decoding Exercise

Base Dictionary

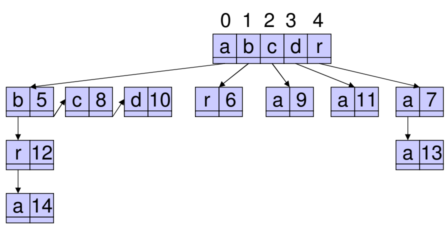
0 a
1 b
2 c
3 d
4 r

0 1 4 0 2 0 3 5 7

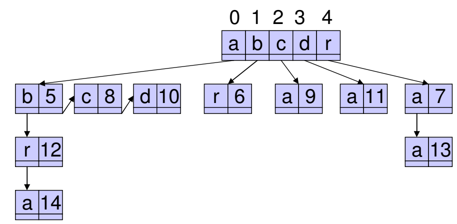
Trie Data Structure for Encoder's Dictionary

- Fredkin (1960)

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

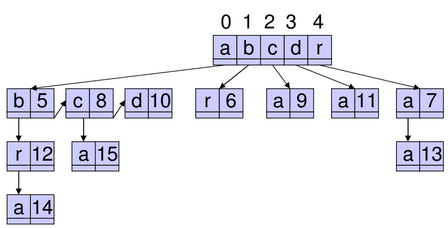


Encoder Uses a Trie (1)



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

Encoder Uses a Trie (2)



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

Decoder's Data Structure

- Simply an array of strings

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

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

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

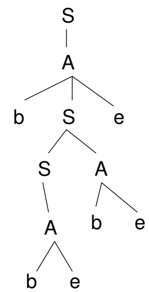
- S → SA
  S → A
  A → bSe
  A → be

Example: b and e matched as parentheses

derivation of bbebee

S
A
bSe
bSAe
bAAe
bbeAe
bbebee

hierarchical parse tree



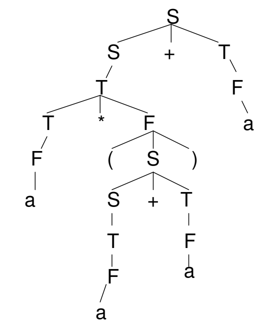
Arithmetic Expressions

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

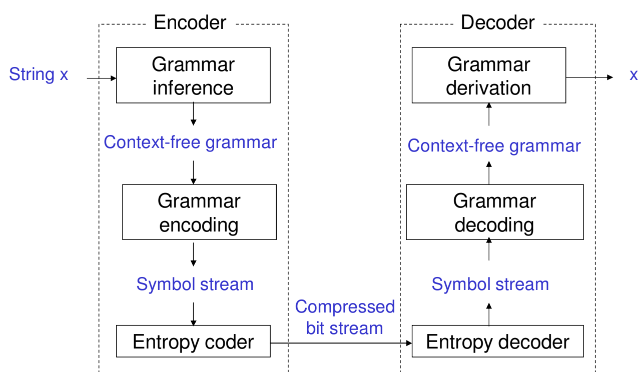
derivation of a * (a + a) + a

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

parse tree



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)

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

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

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

Sequitur Example (26)

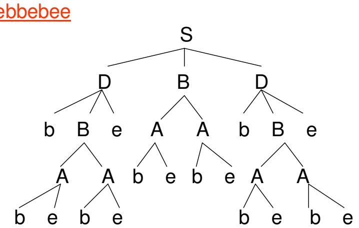
bbebeebebebbebee

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

The Hierarchy

bbebeebebebbebee

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



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:
  $A \rightarrow \ldots.XY\ldots$; $B \rightarrow XY$ $\longrightarrow$ $A \rightarrow \ldots.B\ldots.$; $B \rightarrow XY$
 create a new rule:
  $A \rightarrow \ldots.XY\ldots.$; $B \rightarrow \ldots.XY\ldots.$ $\longrightarrow$ $A \rightarrow \ldots.C\ldots.$; $B \rightarrow \ldots.C\ldots.$; $C \rightarrow XY$
 remove a rule:
  $A \rightarrow \ldots.B\ldots.$; $B \rightarrow X_1X_2\ldots X_k$ $\longrightarrow$ $A \rightarrow \ldots.\ X_1X_2\ldots X_k\ \ldots.$
 input a new symbol:
  $S \rightarrow X_1X_2\ldots X_k$ $\longrightarrow$ $S \rightarrow X_1X_2\ldots X_k s$
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 $m \le 2n$.
- 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.

$A \to \ldots.XY\ldots$ $\quad$ $A \to \ldots.B\ldots.$
$B \to XY$ $\longrightarrow$ $B \to XY$

s	r	2s - r
-1	0	-2

- Digram Uniqueness - create a new rule.

$A \to \ldots.XY\ldots.$ $\quad$ $A \to \ldots.C\ldots.$
$B \to \ldots.XY\ldots.$ $\longrightarrow$ $B \to \ldots.C\ldots.$
$C \to XY$

s	r	2s - r
0	1	-1

- Rule Utility - Remove a rule.

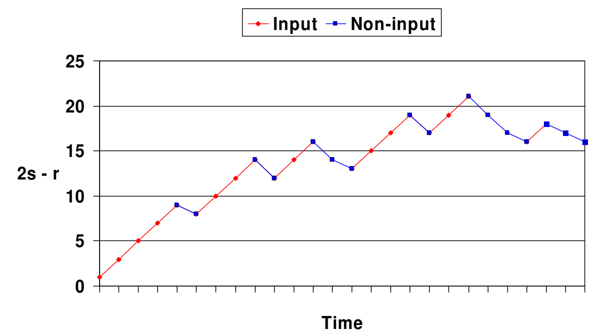
$A \to \ldots.B\ldots.$
$B \to X_1X_2\ldots X_k$ $\longrightarrow$ $A \to \ldots.\ X_1X_2\ldots X_k\ \ldots.$

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

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.

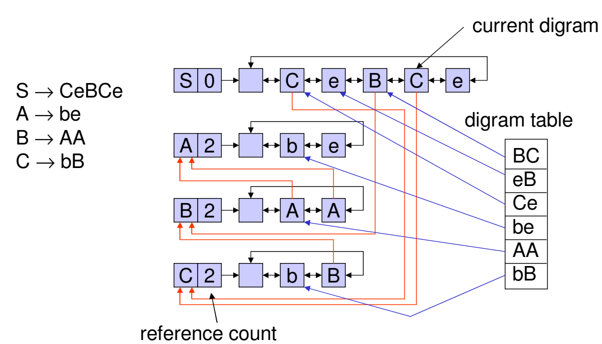
Amortized Complexity Argument



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

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

Symbol Code

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

No code for S needed

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

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.

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:

$S \rightarrow AB$
$A \rightarrow CD$
$B \rightarrow aE$
$C \rightarrow ab$
$D \rightarrow cd$
$E \rightarrow bD$

$\Rightarrow$

$S \rightarrow AA$
$A \rightarrow CD$
$\cancel{B \rightarrow aE}$
$C \rightarrow ab$
$D \rightarrow cd$
$\cancel{E \rightarrow bD}$

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 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: a b a b a a b c c b b c c c c b d b c c
  0 1

0	1	2	3
a	b	c	d

↓

0	1	2	3
b	a	c	d

Example

- Example: a b a b a a b c c b b c c c c b d b c c
  0 1 1

0	1	2	3
b	a	c	d

↓

0	1	2	3
a	b	c	d

Example

- Example: a b a b a a b c c b b c c c c b d b c c
  0 1 1 1

0	1	2	3
a	b	c	d

↓

0	1	2	3
b	a	c	d

Example

- Example: 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	2	3
b	a	c	d

↓

0	1	2	3
a	b	c	d

Example

- Example: 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

0	1	2	3
a	b	c	d

Example

- Example: 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

0	1	2	3
a	b	c	d

↓

0	1	2	3
b	a	c	d

Example

- Example: 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 2

0	1	2	3
b	a	c	d

↓

0	1	2	3
c	b	a	d

Example

- Example: 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 2 0 1 0 1 0 00 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 b d b c c
  0 1 1 1 1 0 1 2 0 1 0 1 0 00 1 3 1 2 0

Frequencies of {a, b, c, d}

a	b	c	d	
4	7	8	1	Entropy = 1.74

Frequencies of {0, 1, 2, 3}

0	1	2	3	
8	9	2	1	Entropy = 1.6

Extreme Example

Input:

aaaaaaaaaabbbbbbbbbbccccccccccdddddddddd

Output

0000000000100000000020000000003000000000

Frequencies of a b c d

a	b	c	d	
10	10	10	10	Entropy = 2

Frequencies of 0 1 2 3

0	1	2	3	
37	1	1	1	Entropy = .5

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.

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

Encoding Example

2. Sort the strings alphabetically in to array A

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

Encoding Example

3. L = the last column

A

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

L = rdarcaaaabb

Encoding Example

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

A

0	aabracadabr
1	abraabracad
$\boxed{2}$	abracadabra
3	acadabraabr
4	adabraabrac
5	braabracada
6	bracadabraa
7	cadabraabra
8	dabraabraca
9	raabracadab
10	racadabraab

L = 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

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

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]

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

Decoding Example

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

```
F = 0 1 2 3 4 5 6 7 8 9 10
    a a a a a b b c 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
```

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

Decoding Example

```
F = 0 1 2 3 4 5 6 7 8 9 10
    a a a a a b b c 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

      a
```

Decoding Example

```
F = 0 1 2 3 4 5 6 7 8 9 10
    a a a a a b b c 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
```

ab

Decoding Example

```
F = 0 1 2 3 4 5 6 7 8 9 10
    a a a a a b b c 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
```

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].

A		T	A^s	
0	aabracadabr	2	0	raabracadab
1	abraabracad	5	1	dabraabraca
2	abracadabra	6	2	aabracadabr
3	acadabraabr	7	3	racadabraab
4	adabraabrac	8	4	cadabraabra
5	braabracada	9	5	abraabracad
6	bracadabraa	10	6	abracadabra
7	cadabraabra	4	7	acadabraabr
8	dabraabraca	1	8	adabraabrac
9	raabracadab	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

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

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

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


Decoding Example

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


Decoding Example

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


Decoding Example

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

Decoding Example

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

BWT Encoding Exercise

Encode the string abababababababab = $(ab)^8$
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

	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 LZW
gzip - based on LZ77
PPMC - adaptive arithmetic coding with context
bzip2 – Burrows-Wheeler block sorting