Fonte: ash.atozviews.com

Compacto dos melhores momentos

AULA 17 e 18

Compressão de dados

Problema: representar um arquivo GRANDE por outro menor.

Exemplo:
arquivo GRANDE: `abababababababababababababab`
arquivo menor: `12ab`

Por que comprimir?
Menor espaço de armazenamento.
Menor tempo de transmissão.

Software: `gzip`, `bzip`, `7z`, etc.

Esquema

representar um dado fluxo de bits (*bitstream*) por outro mais curto.

Esquema básico de compressão de dados:

- um compressor transforma um fluxo de bits `B` em um fluxo `C(B)` e
- um expansor transforma `C(B)` de volta em `B`.

Basic model for data compression

Entradas e saídas binárias

Cadeia de bits (=*bitstring*) é uma sequência de bits:

```
1100010000011101010100111111011001110001010000
1100000000001111110000011101000111
```

fluxo de bits (=*bitstream*) é uma cadeia de bits na entrada ou na saída de um programa.

A classe `BinaryStdIn` lê um fluxo de bits a partir da entrada padrão.

A classe `BinaryStdOut` escreve um fluxo de bits na saída padrão.

BinaryDump, HexDump e PictureDump

```
% more abra.txt
ABRACADABRA!

% java BinaryDump 16 < abra.txt
0100000101000010
0101001001000001
0100001101000001
0100010001000001
0100001001010010
0100000100100001
96 bits
```

Genomas

Podemos usar apenas 2 bits por caractere:

```
% more genomeTiny.txt
ATAGATGCATAGCGCATAGCTAGATGTGCTAGC

% java Genome - < genomeTiny.txt | java
BinaryDump 32

00000000000000000000000000100001
00110010001110010011001001100100
11001001110010001110111001110010
01000000
104 bits
```

Codificação de comprimento de carreira

Em inglês: *run-length encoding* (=RLE).

Exemplo: cadeia de bits abaixo tem uma carreira de 15 0s, uma carreira de 7 1s, uma de 7 0s, e uma de 11 1s:

000000000000000111111100000011111111111

Pode ser representada pela sequência 15 7 7 11. Usando **8 bits** para cada um desses números, teremos uma cadeia de apenas 32 bits (ignore os espaços):

00001111000001110000011100001011

Algoritmo de Huffman

Ideia do algoritmo de Huffman: usar códigos **curtos** para os caracteres que ocorrem com **frequência** e deixar os códigos mais **longos** para os caracteres mais **raros**.

Algoritmo de Huffman

Uma tabela de códigos leva cada caractere de 8 bits no seu código.
Exemplo de códigos de comprimento fixo:

caractere	código
!	001
A	010
B	011
C	100
D	101
R	110

A tabela de códigos é uma ST em que as chaves são os caracteres e os valores são os códigos.

Algoritmo de Huffman

Usando a tabela de códigos anterior gastaríamos

$$100000 \times 3 = 300000 \quad \text{bits}$$

para codificar o arquivo.

Examinando o arquivo observamos a seguinte frequência de símbolos.

caractere	frequência
!	5000
A	45000
B	13000
C	9000
D	16000
R	12000

Algoritmo de Huffman

Exemplo de códigos de comprimento variável:

caractere	código
!	1010
A	0
B	111
C	1011
D	100
R	110

Com essa nova tabela de códigos gastaríamos

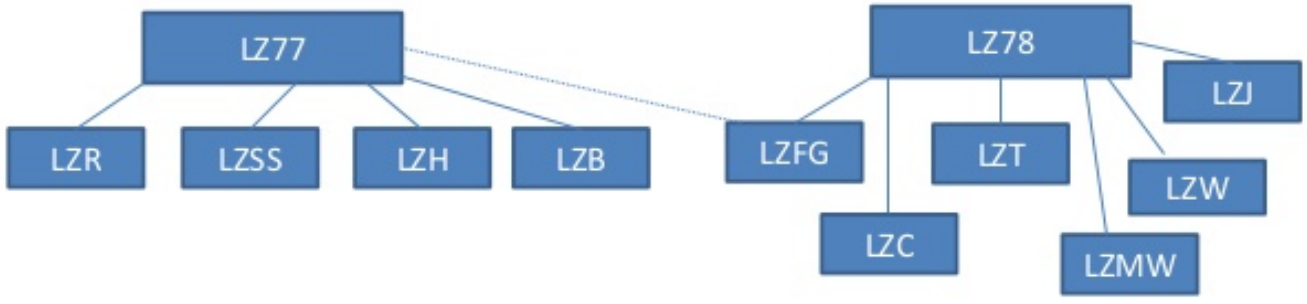
$$5\mathtt{M} \times 4 + 45\mathtt{M} \times 1 + 13\mathtt{M} \times 3 + 9\mathtt{M} \times 4 + 16\mathtt{M} \times 3 + 12\mathtt{M} \times 3$$

bits $=$ 224000 bits para representar o arquivo.

AULA 19

Algoritmo LZW

Limpel Ziv Algorithm Family



APPLICATIONS:
- ZIP
- GZIP
- STACKER

APPLICATIONS:
- GIF
- V.42
- COMPRESS

Fig 1: Limpel Ziv Algorithm Family

Referências: Data Compression (SW), slides (SW), LZW compression, video (SW)

Algoritmo LZW

Desenvolvido por Abraham Lempel e Jakob Ziv em 1977 (LZ77).

Refinamento de LZ78 por Terry Welch em 1978.

Huffman é um método estatística de compressão de dados.

LZW é um método baseado em dicionários (*dictionary based compression systems*).

Diferentemente do algoritmo de Huffman, o algoritmo LZW transforma strings de tamanho variado em códigos de tamanho fixo de W bits (usualmente de 8 a 12).

Métodos baseados em dicionários

Não usam conhecimento estatístico dos dados.

`compress()`: a medida que a entrada é processada constrói um dicionário e utiliza os índices das strings no dicionário como código.

`expand()`: a medida que a entrada é processada reconstrói o dicionário para reverter o trabalho de `compress()`.
Exemplos: `LZW`, `LZ77`, Sequitur

Aplicações: Unix compress, `gzip`, GIF

LZW compress()

input = string de entrada

```
crie dicionário com símbolos do alfabeto
repita
   encontre o maior prefixo s de input
             que está no dicionário
   transmita para saída o índice de s
   ponha s+c no dicionário onde c é
         o símbolo que segue s em input
         (unmatched symbol).
   apague do input o prefixo s
até que input é vazio
```

LZW exemplo

Decisões de projeto:

- Alfabeto $= \{\texttt{a}, \texttt{b}\} \Rightarrow \texttt{R} = 2$
- $\texttt{W} = 3$ bits $\Rightarrow$ índices entre 0 e 7
- tamanho $\texttt{L}$ do dicionário é $2^{\texttt{W}} = 8$
- índice $\texttt{R}$ codificará $\texttt{EOF}$ (end of file)

LZW compress(): exemplo

input	a	b	a	b	a	b	a	b	a	b	
codificação											

dicionário

string	código
a	0
b	1
EOF	2

LZW compress(): exemplo

input	a	b	a	b	a	b	a	b	a	b	
codificação	0										

dicionário

string	código
a	0
b	1
EOF	2
ab	3

LZW compress(): exemplo

input	a	b	a	b	a	b	a	b	a	b	
codificação	0	1									

dicionário

string	código
a	0
b	1
EOF	2
ab	3
ba	4

LZW compress(): exemplo

input	a	b	a	b	a	b	a	b	a	b	
codificação	0	1	3								

dicionário

string	código
a	0
b	1
EOF	2
ab	3
ba	4
aba	5

LZW compress(): exemplo

input	a	b	a	b	a	b	a	b	a	b	
codificação	0	1	3		5						

dicionário

string	código
a	0
b	1
EOF	2
ab	3
ba	4
aba	5
abab	6

LZW compress(): exemplo

input	a	b	a	b	a	b	a	b	a	b	
codificação	0	1	3		5			4			

dicionário

string	código
a	0
b	1
EOF	2
ab	3
ba	4
aba	5
abab	6
bab	7

LZW compress(): exemplo

input	a	b	a	b	a	b	a	b	a	b	
codificação	0	1	3		5			4		1	

dicionário

string	código
a	0
b	1
EOF	2
ab	3
ba	4
aba	5
abab	6
bab	7

LZW compress(): exemplo

input	a	b	a	b	a	b	a	b	a	b	
codificação	0	1	3		5			4		1	2

dicionário

string	código
a	0
b	1
EOF	2
ab	3
ba	4
aba	5
abab	6
bab	7

LZW compress(): exemplo

Resumo

Fomos de $8 \times 10 = 80$ bits para representar

a b a b a b a b a b

para W $\times 7 = 3 \times 7 = 21$ bits:

000 001 011 101 100 001 010

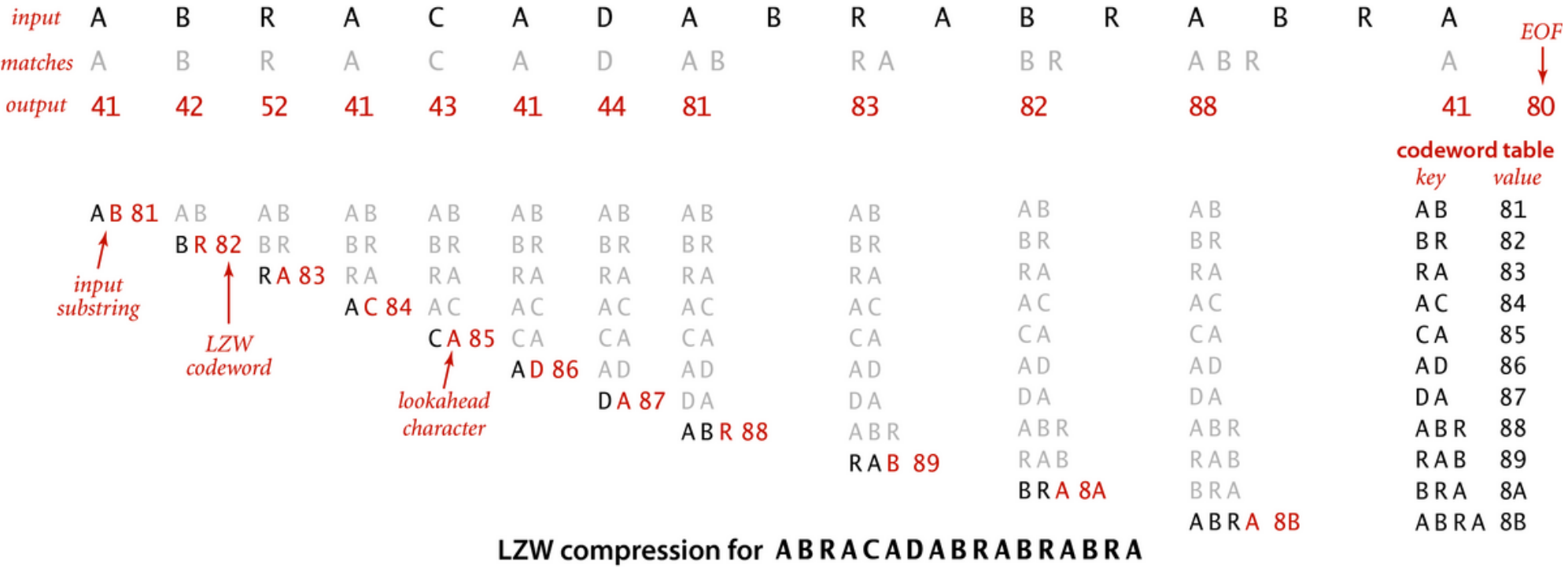
Taxa de compressão $= 21/80 = 0.2625 \sim 26\%$

LZW compress(): mais um exemplo

Decisões de projeto:

- Alfabeto = ASCII $\Rightarrow$ R = 128
- W = 8 bits $\Rightarrow$ índices entre 0 e 255
- tamanho L do dicionário é $2^W = 256$
- índice R = 128 (= 80 hexa) codificará EOF

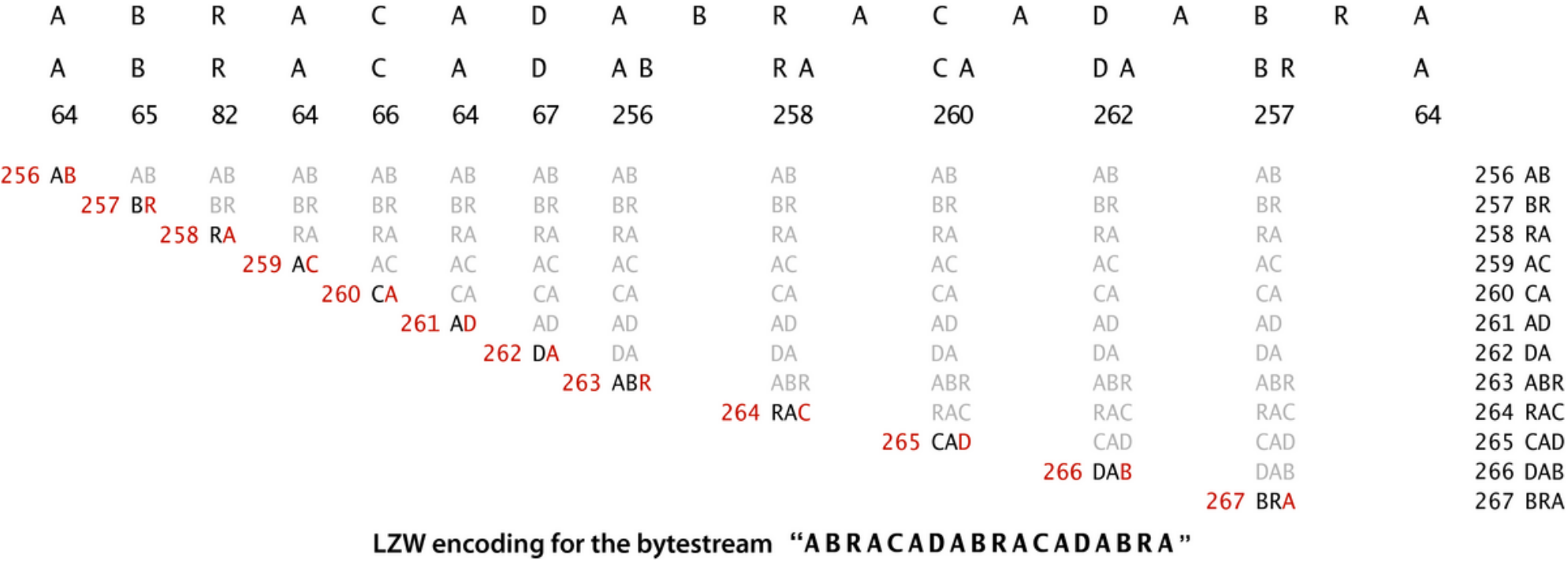
LZW compress(): mais um exemplo



LZW compression for A B R A C A D A B R A B R A B R A

Fonte: algs4

LZW compress(): mais um exemplo



LZW encoding for the bytestream "A B R A C A D A B R A C A D A B R A"

Fonte: algs4

Sobre o dicionário

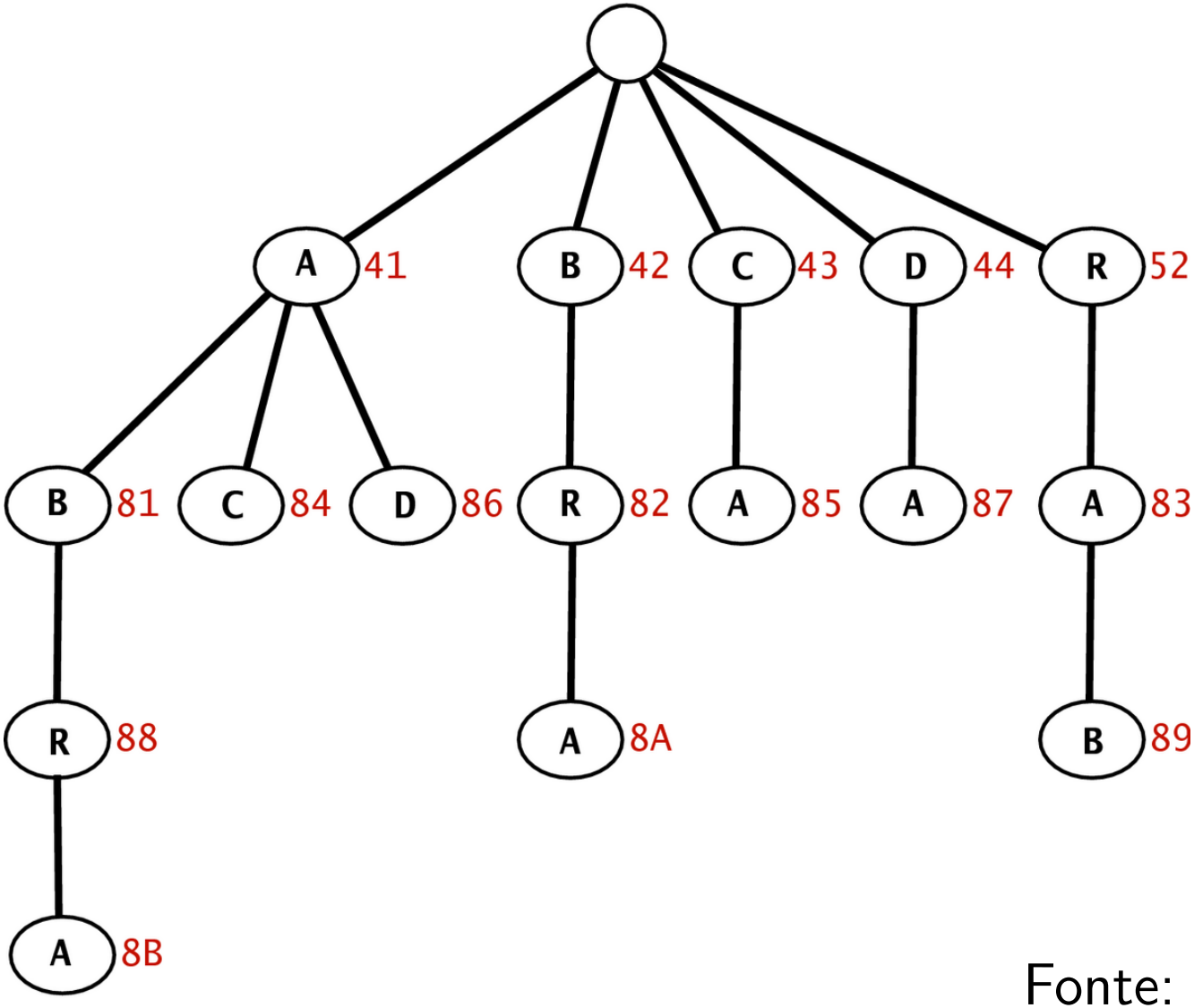
`Dicionário` de tamanho fixo:

- `W` bits de índices permite um dicionário de tamanho $2^{\texttt{W}}$;
- dificilmente todas as entradas no dicionário serão úteis.

Estratégias quando o dicionário atinge seu limite:

- não ponha mais strings, somente use o que já estão no dicionário;
- jogue tudo fora e comece um novo dicionário;
- dobre o dicionário, acrescentando mais um bit ao índices ($\texttt{W} \leftarrow \texttt{W} + 1$);
- jogue fora as entradas mais recentes para fazer espaço para mais entrada;
- ideias?

Dicionário (Trie) de LZW



Fonte: algs4

Trie representation of LZW code table

compress()

Estatística dos dados não precisa ser passada por compress() para expand().

```
public static void compress() {
   String input=BinaryStdIn.readString();
   TST<Integer> st = new TST<Integer>();
   for (int i = 0; i < R; i++)
       st.put("" + (char) i, i);

   // R é o código para EOF
   int code = R+1;
```

compress()

```
 while (input.length() > 0) {
    String s = st.longestPrefixOf(input);
    BinaryStdOut.write(st.get(s), W);
    int t = s.length();
    if (t < input.length() && code < L)
        st.put(input.substring(0,t+1),code++);
    input = input.substring(t);
 }
 BinaryStdOut.write(R, W); // EOF
 BinaryStdOut.close();
}
```

abra.txt

```
% more abra.txt | java B...Dump 0
96 bits

% java LZW - < abra.txt |java B...Dump 24
000001000001000001000010
000001010010000001000001
000001000011000001000001
000001000100000100000001
000100000011000000100001
000100000000000
136 bits
```

LZW expand(): exemplo

msg codificada

0	1	3	5	4	1	2

output

dicionário

código	string

LZW expand(): exemplo

msg codificada

0	1	3	5	4	1	2

output

dicionário

código	string
0	a
1	b
2	EOF

LZW expand(): exemplo

msg codificada	0	1	3	5	4	1	2
output	a						

dicionário

código	string
0	a
1	b
2	EOF
3	a?

LZW expand(): exemplo

msg codificada	0	1	3	5	4	1	2
output	a	b					

dicionário

código	string
0	a
1	b
2	EOF
3	ab

LZW expand(): exemplo

msg codificada	0	1	3	5	4	1	2
output	a	b					

dicionário

código	string
0	a
1	b
2	EOF
3	ab
4	b?

LZW expand(): exemplo

msg codificada

0	1	3		5	4	1	2
a	b	a	b				

output

dicionário

código	string
0	a
1	b
2	EOF
3	ab
4	ba

LZW expand(): exemplo

msg código	0	1	3		5	4	1	2
output	a	b	a	b				

dicionário

código	string
0	a
1	b
2	EOF
3	ab
4	ba
5	ab?

LZW expand(): exemplo

msg código	0	1	3		5	4	1	2
output	a	b	a	b				

dicionário

código	string
0	a
1	b
2	EOF
3	ab
4	ba
5	ab?
	Xiii!

LZW expand(): exemplo

msg código	0	1	3		5			4	1	2
output	a	b	a	b	a	b	?			

dicionário

código	string
0	a
1	b
2	EOF
3	ab
4	ba
5	ab?
	Xiii!

LZW expand(): exemplo

msg código	0	1	3		5			4	1	2
output	a	b	a	b	a	b	a			

dicionário

código	string
0	a
1	b
2	EOF
3	ab
4	ba
5	aba
	:-)

LZW expand(): exemplo

msg código	0	1	3		5			4	1	2
output	a	b	a	b	a	b	a			

dicionário

código	string
0	a
1	b
2	EOF
3	ab
4	ba
5	aba
6	aba?

LZW expand(): exemplo

msg código	0	1	3		5			4		1	2
output	a	b	a	b	a	b	a	b	a		

dicionário

código	string
0	a
1	b
2	EOF
3	ab
4	ba
5	aba
6	aba?

LZW expand(): exemplo

msg código	0	1	3		5			4		1	2
output	a	b	a	b	a	b	a	b	a		

dicionário

código	string
0	a
1	b
2	EOF
3	ab
4	ba
5	aba
6	abab

LZW expand(): exemplo

msg código	0	1	3		5			4		1	2
output	a	b	a	b	a	b	a	b	a		

dicionário

código	string
0	a
1	b
2	EOF
3	ab
4	ba
5	aba
6	abab
7	ba?

LZW expand(): exemplo

msg código	0	1	3		5			4		1	2
output	a	b	a	b	a	b	a	b	a	b	

dicionário

código	string
0	a
1	b
2	EOF
3	ab
4	ba
5	aba
6	abab
7	ba?

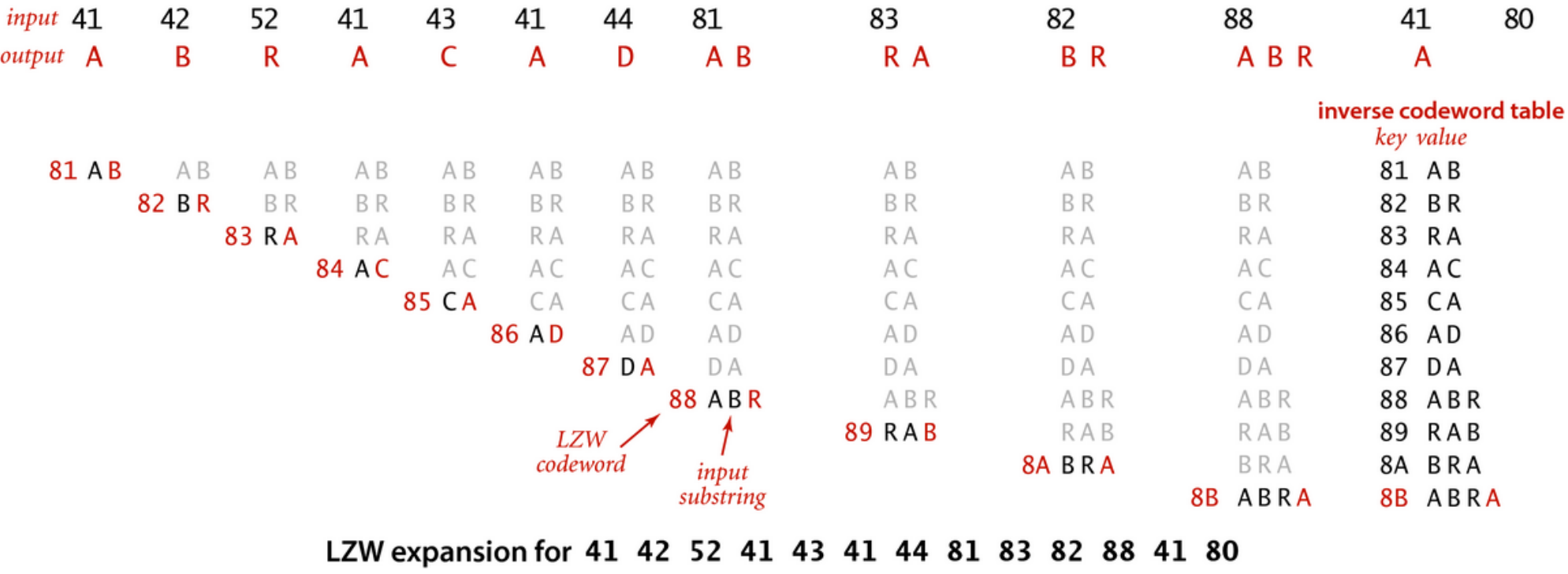
LZW expand(): exemplo

msg código	0	1	3		5			4		1	2
output	a	b	a	b	a	b	a	b	a	b	

dicionário

código	string
0	a
1	b
2	EOF
3	ab
4	ba
5	aba
6	abab
7	bab

LZW expand(): mais um exemplo



LZW expansion for 41 42 52 41 43 41 44 81 83 82 88 41 80

Fonte: algs4

LZW *lookahead*

compression

input	A	B	A	B	A	B	A	
matches	A	B	A B		A B A			
output	41	42	81		83			80

A B 81 A B A B

B A 82 B A

A B A 83

codeword table

key	*value*
A B	81
B A	82
A B A	83

expansion

input	41	42	81	83	80
output	A	B	A B	?	

must be A B A (see below)

81 A B A B A B

82 B A B A

83 A B ?

need lookahead character to complete entry

next character in output—the lookahead character!

LZW expansion: tricky situation

LZW `expand()`

`expand()`

- simula `compress()` para reconstruir o dicionário.
- anda um *um pouquinho atrás* de `compress()`, mas olhando para frente (*lookahead*).

LZW expand()

```
crie dicionário com símbolos do alfabeto
decodifique o primero índice code para a
            string val
coloque val+? no dicionário
repita
   decodifique o primeiro símbolo c
            do próximo índice code
   complete com c a última entrada
            da tabela
   termine de decodificar code e obtenha
            a string s
   transmita s para a saída
   ponha s+? no dicionário
até que input é vazio
```

expand()

```
public static void expand() {
   String[] st = new String[L];
   int i; próximo código disponível
   // inicialize a ST
   for (i = 0; i < R; i++)
       st[i] = "" + (char) i;
   st[i++] = " ";
   int codeword = BinaryStdIn.readInt(W);
   // mensagem é vazia?
   if (codeword == R) return;
   String val = st[codeword];
```

expand()

```
    while(true) {
        BinaryStdOut.write(val);
        codeword = BinaryStdIn.readInt(W);
        if (codeword == R) break;
        String s = st[codeword];
        if (i == codeword)
            // caso especial
            s = val + val.charAt(0);
        if (i < L)
            st[i++] = val + s.charAt(0);
        val = s;
    }
    BinaryStdOut.close();
}
```

Esqueleto da classe LZW

```
public class LZW {
   // tamanho do alfabeto
   private static final int R = 256;

   // número de bits dos códigos
   private static final int W = 12;

   // number códigos 2^W
   private static final int L = 4096;

   public static void compress() {...}
   public static void expand() {...}
}
```

LZW exemplos

virus (50000 bits)

```
% java Genome - < genomeVirus.txt | java PictureDump 512 25
```

```
12536 bits

% java LZW - < genomeVirus.txt | java PictureDump 512 36
```

```
18232 bits
```
← *not as good as 2-bit code because repetitive data is rare*

bitmap (6144 bits)

```
% java RunLength - < q64x96.bin | java BinaryDump 0
2296 bits

% java LZW - < q64x96.bin | java BinaryDump 0
2824 bits
```
← *not as good as run-length code because file size is too small*

entire text of *Tale of Two Cities* (5812552 bits)

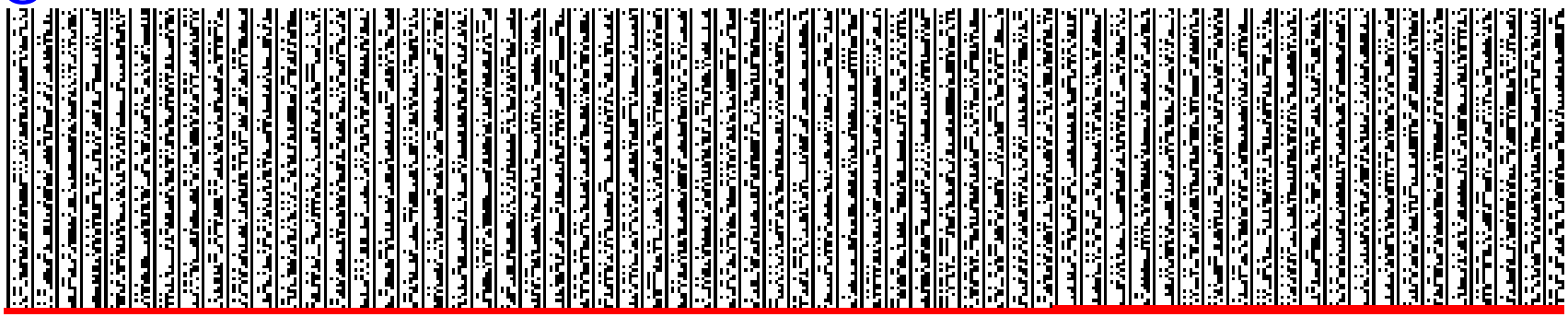
```
% java BinaryDump 0 < tale.txt
5812552 bits

% java Huffman - < tale.txt | java BinaryDump 0
3043928 bits

% java LZW - < tale.txt | java BinaryDump 0
2667952 bits
```
← *compression ratio 2667952/5812552 = 46% (best yet)*

Compressing and expanding various files with LZW 12-bit encoding

Fonte: algs4

Genome versus LZW

```
% java PictureDump 512 100 <
genomeVirus.txt
```



50008 bits

Genome versus LZW

```
% java Genome - < genomeVirus.txt | java
PictureDump 512 25
```

12536 bits

```
% java LZW - < genomeVirus.txt | java
PictureDump 512 25
```

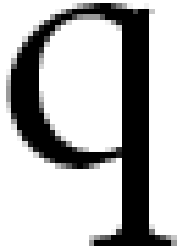
18232 bits

strings repetidas parecem ser raras.

RunLength versus LZW

```
% java PictureDump 32 48 < q32x48.bin
```

1536 bits



```
% java RunLength - < q32x48.bin | java
BinaryDump 0
```

1144 bits

```
% java LZW - < q32x48.bin | java
BinaryDump 0
```

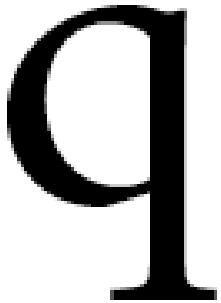
1176 bits

Tamanho do arquivo é muito pequeno?

RunLength versus LZW

```
% java PictureDump 64 96 < q32x48.bin
```

6144 bits



```
% java RunLength - < q32x48.bin | java
BinaryDump 0
```

2296 bits

```
% java LZW - < q32x48.bin| java
BinaryDump 0
```

2824 bits

Tamanho do arquivo é muito pequeno?

Huffman versus LZW

```
% java BinaryDump 0 < les-miserables.txt
```
26581192 bits

```
% java Huffman - < les-miserables.txt |
java BinaryDump 0
```
15211904 bits

```
% java LZW - < les-miserables.txt | java
BinaryDump 0
```
15316048 bits **demorou muito para responder!**

Observações sobre LZW

- positivo: extremamente efetivo quando padrões repetidos ocorrem de maneira dispersa nos dados
- negativo: cria entradas no dicionário que podem não ser usadas

LZ77

Ziv e Lempel, 1977

Dicionário é implícito.

Usa a uma *janela* formada por um sufixo da string codificado até agora como dicionário

Se `input.substring(0,n)` foi codificado, queremos codificar `input.substring(0,n+k)` para o maior `k` possível

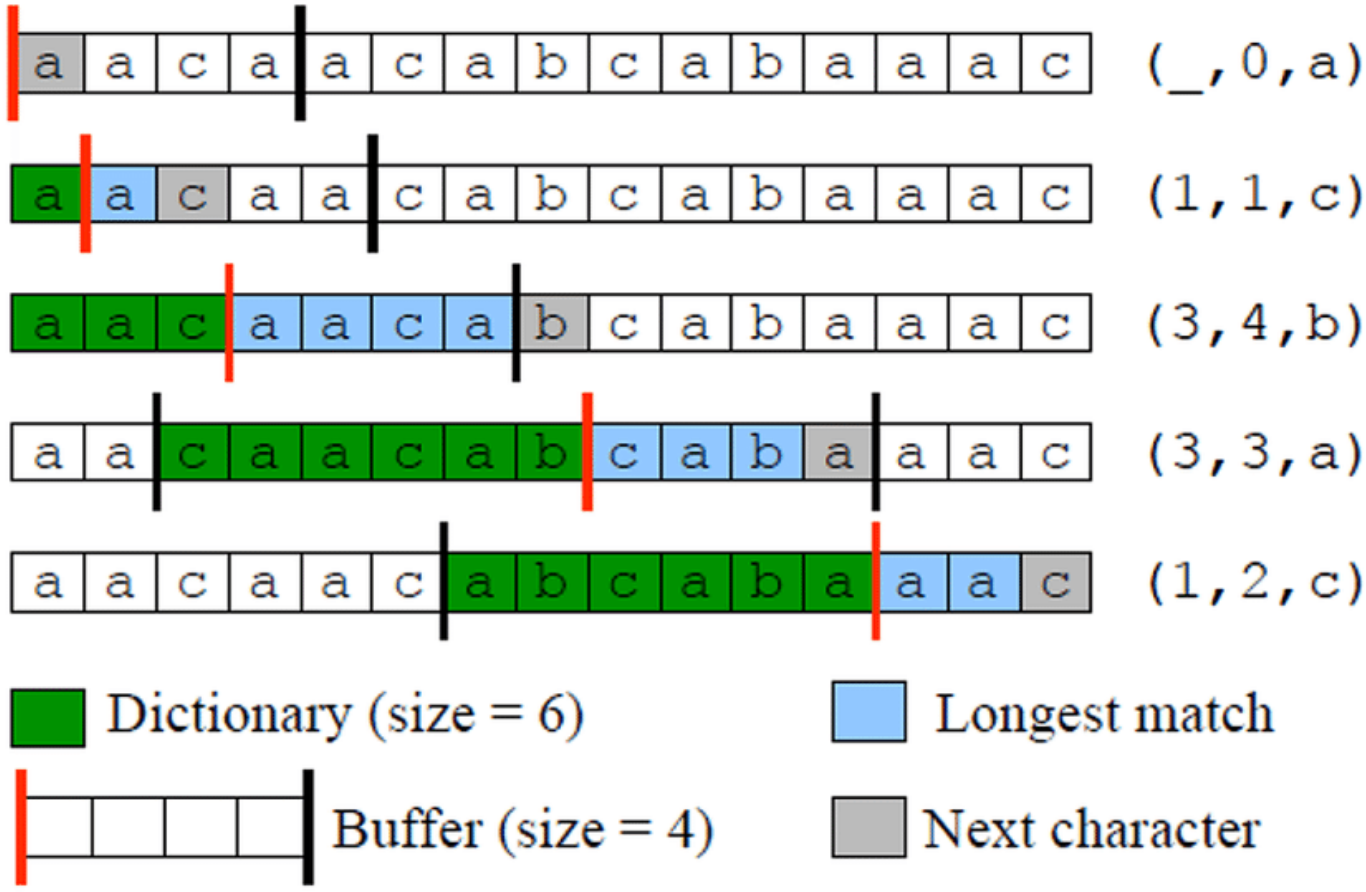
LZ77

Ideia: `input.substring(n,n+k)` é substring de `input.substring(0,n)`, então podemos codificar `input.substring(n,n+k)` por um par de inteiros $\langle$`j`, `k`$\rangle$ onde `j` é índice do início da ocorrência da ocorrência de `input.substring(n,n+k)` na janela. De manera semelhante a LZW, esse para é seguido do caractere `input[k+n]`

Exemplo:

abababa**ba**	**bababa**	**xbababaababa**...
codificado	$\langle 1, 8,$ `x`$\rangle$	a codificar

LZ77: exemplo



Fonte: Researchgate

Observações sobre LZ77

Muito polular no mundo Unix.

Muitas variantes implementadas: `zip`, `gzip`, `png`, `pkzip`, `lharc`, `arj`

Em geral melhor que LZW:

- LZW tem dicionário com entradas que não são usadas
- LZW tem substrings examinados que não estão no dicionário
- LZ77 tem um dicionário implicito e pares frequentes são codificados com menos bits.