

Compacto dos melhores momentos

AULA 16 e 17

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:

```
110001000001110101010011111011001110001010000  
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
```

```
00000000000000000000000000000000100001
```

```
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:

0000000000000001111111000000011111111111

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

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

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

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

bits = 224000 bits para representar o arquivo.

AULA 18

Algoritmo LZW

Lempel Ziv Algorithm Family

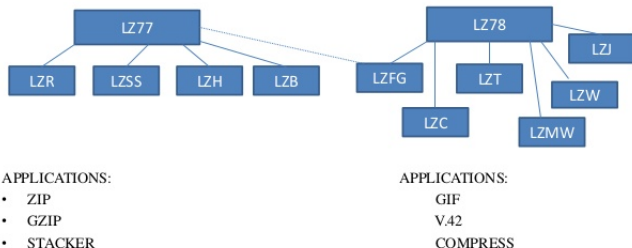


Fig 1: Lempel Ziv Algorithm Family

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

LZW compress(): exemplo

input
codificação

a	b	a	b	a	b	a	b	a	b	

dicionário

string	código
a	0
b	1
EOF	2

LZW compress(): exemplo

input
codificação

a	b	a	b	a	b	a	b	a	b	
0										

dicionário

string	código
a	0
b	1
EOF	2
ab	3

LZW compress(): exemplo

input
codificação

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

dicionário

string	código
a	0
b	1
EOF	2
ab	3
ba	4

LZW compress(): exemplo

input
codificação

a	b	a	b	a	b	a	b	a	b	
0	1	3								

dicionário

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

LZW compress(): exemplo

input
codificação

a	b	a	b	a	b	a	b	a	b	
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
codificação

a	b	a	b	a	b	a	b	a	b	
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
codificação

a	b	a	b	a	b	a	b	a	b	
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
codificação

a	b	a	b	a	b	a	b	a	b	
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

<i>input</i>	A	B	R	A	C	A	D	A	B	R	A	B	R	A	B	R	A	<i>EOF</i>
<i>matches</i>	A	B	R	A	C	A	D	AB		RA		BR		ABR			A	↓
<i>output</i>	41	42	52	41	43	41	44	81		83		82		88			41	80

		<i>codeword table</i>																	
		<i>key</i>																<i>value</i>	
	AB 81	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	81
		BR 82	BR	BR	BR	BR	BR	BR	BR	BR	BR	BR	BR	BR	BR	BR	BR	BR	82
			RA 83	RA	RA	RA	RA	RA	RA	RA	RA	RA	RA	RA	RA	RA	RA	RA	83
				AC 84	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	84
					CA 85	CA	CA	CA	CA	CA	CA	CA	CA	CA	CA	CA	CA	CA	85
						AD 86	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	86
							DA 87	DA	DA	DA	DA	DA	DA	DA	DA	DA	DA	DA	87
								ABR 88	ABR	ABR	ABR	ABR	ABR	ABR	ABR	ABR	ABR	ABR	88
									RA B 89	RAB	RAB	RAB	RAB	RAB	RAB	RAB	RAB	RAB	89
										BRA 8A	BRA	BRA	BRA	BRA	BRA	BRA	BRA	BRA	8A
											ABRA 8B	ABRA	ABRA	ABRA	ABRA	ABRA	ABRA	ABRA	8B

LZW compression for ABRACADABRABRABRA

Fonte: [algs4](#)

LZW compress(): mais um exemplo

A	B	R	A	C	A	D	A	B	R	A	C	A	D	A	B	R	A
A	B	R	A	C	A	D	A B		R A		C A		D A		B R		A
64	65	82	64	66	64	67	256		258		260		262		257		64
256 AB	AB	AB	AB	AB	AB	AB	AB		AB		AB		AB		AB		256 AB
	257 BR	BR	BR	BR	BR	BR	BR		BR		BR		BR		BR		257 BR
		258 RA	RA	RA	RA	RA	RA		RA		RA		RA		RA		258 RA
			259 AC	AC	AC	AC	AC		AC		AC		AC		AC		259 AC
				260 CA	CA	CA	CA		CA		CA		CA		CA		260 CA
					261 AD	AD	AD		AD		AD		AD		AD		261 AD
						262 DA	DA		DA		DA		DA		DA		262 DA
							263 ABR		ABR		ABR		ABR		ABR		263 ABR
								264 RAC	RAC		RAC		RAC		RAC		264 RAC
										265 CAD		CAD		CAD		CAD	265 CAD
											266 DAB		DAB		DAB		266 DAB
												267 BRA		BRA		BRA	267 BRA

LZW encoding for the bytestream "ABRACADABRACADABRA"

Fonte: [algs4](#)

Sobre o dicionário

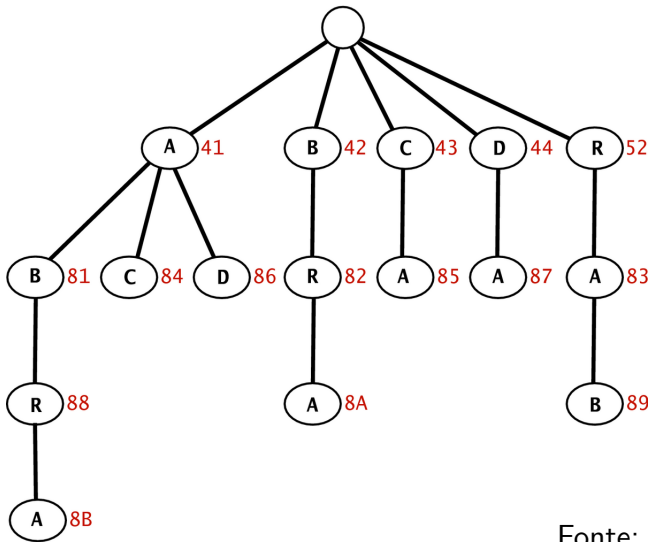
Dicionário de tamanho fixo:

- ▶ W bits de índices permite um dicionário de tamanho 2^W ;
- ▶ **difícilmente** 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 ($W \leftarrow W + 1$);
- ▶ **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
```

```
0001000000000000
```

```
136 bits
```


LZW expand(): exemplo

msg codificada

0	1	3	5	4	1	2

output

dicionário

string	código
a	0
b	1
EOF	2

LZW expand(): exemplo

msg codificada

0	1	3	5	4	1	2
a						

output

dicionário

string	código
a	0
b	1
EOF	2
a?	3

LZW expand(): exemplo

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

dicionário

string	código
a	0
b	1
EOF	2
ab	3

LZW expand(): exemplo

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

dicionário

string	código
a	0
b	1
EOF	2
ab	3
b?	4

LZW expand(): exemplo

msg codificada

0	1	3		5	4	1	2
---	---	---	--	---	---	---	---

output

a	b	a	b				
---	---	---	---	--	--	--	--

dicionário

string	código
a	0
b	1
EOF	2
ab	3
ba	4

LZW expand(): exemplo

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

dicionário

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

LZW expand(): exemplo

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

dicionário

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

LZW expand(): exemplo

msg código

output

0	1	3		5			4	1	2
a	b	a	b	a	b	?			

dicionário

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

LZW expand(): exemplo

msg código

output

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

dicionário

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

LZW expand(): exemplo

msg código

output

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

dicionário

string	código
--------	--------

a	0
---	---

b	1
---	---

EOF	2
-----	---

ab	3
----	---

ba	4
----	---

aba	5
-----	---

aba?	6
------	---

LZW expand(): exemplo

msg código

output

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

dicionário

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

LZW expand(): exemplo

msg código

output

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

dicionário

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

LZW expand(): exemplo

msg código

output

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

dicionário

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

LZW expand(): exemplo

msg código

output

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

dicionário

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

LZW expand(): exemplo

msg código

output

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

dicionário

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

LZW expand(): mais um exemplo

<i>input</i>	41	42	52	41	43	41	44	81		83	82	88	41	80
<i>output</i>	A	B	R	A	C	A	D	A B		R A	B R	A B R	A	

81 AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	81 AB
82 BR	BR	BR	BR	BR	BR	BR	BR	BR	BR	BR	BR	BR	82 BR
83 RA	RA	RA	RA	RA	RA	RA	RA	RA	RA	RA	RA	RA	83 RA
84 AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	84 AC
85 CA	CA	CA	CA	CA	CA	CA	CA	CA	CA	CA	CA	CA	85 CA
86 AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	86 AD
87 DA	DA	DA	DA	DA	DA	DA	DA	DA	DA	DA	DA	DA	87 DA
88 ABR	ABR	ABR	ABR	ABR	ABR	ABR	ABR	ABR	ABR	ABR	ABR	ABR	88 ABR
89 RAB	RAB	RAB	RAB	RAB	RAB	RAB	RAB	RAB	RAB	RAB	RAB	RAB	89 RAB
8A BRA	BRA	BRA	BRA	BRA	BRA	BRA	BRA	BRA	BRA	BRA	BRA	BRA	8A BRA
8B ABRA	ABRA	ABRA	ABRA	ABRA	ABRA	ABRA	ABRA	ABRA	ABRA	ABRA	ABRA	ABRA	8B ABRA

inverse codeword table

key value

LZW codeword ↗ ↖ *input substring*

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

Fonte: [algs4](#)

LZW expand()

expand()

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

LZW expand()

crie dicionário com símbolos do alfabeto
decodifique o primeiro índice code para o
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
o 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
```

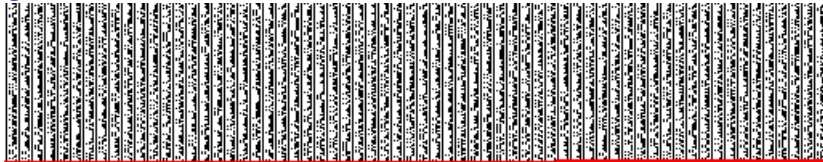
```
% java LZW - < tale.txt | java BinaryDump 0  
2667952 bits ← compression ratio 2667952/5812552 = 46% (best yet)
```

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
```

Q

```
% 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 são usadas

LZ77

Ziv e Lempel, 1977

Dicionário é implícito.

Usa o **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(0, n+k)` é substring de `input.substring(0, n)`, então podemos codificar `input.substring(0, n+k)` por um par de inteiros $\langle j, k \rangle$ onde j é índice do início da ocorrência.

Exemplo:

<u>ababababa</u>	babababa	bababaababa...
codificado	$\langle 1, 8 \rangle$	a codificar

Observações sobre LZ77

Muito popular 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 implícito e pares frequentes são codificados com menos bits.