Exercício : 1 _____ / _____ / _____

Algoritmo Lempel-Ziv Welch (LZW)

Acrônimo dos nomes dos seus inventores Lempel, Ziv e Welch. Lempel e Ziv publicaram suas idéias em 1977 (ZIV, Jacob, LEMPEL, Abraham; Compression of Individual Sequences Via Variable-Rate Coding, IEEE Transactions on Information Theory, September 1978) e Terry Welch refinou o algoritmo em 1984. (WELCH, T. A. (June 1984). "A technique for high-performance data compression."Computer. Vol. 17, pp. 8-19.)

Em síntese, o algoritmo troca conjuntos de caracteres por códigos simples. Não há análise prévia do conteúdo e a compressão ocorre quando um conjunto grande de símbolos é trocado por um único código. Os códigos LZW podem ter qualquer número de bits, mas precisam ser maiores do que a representação de 1 caracter (ou seja, maiores do que 8 bits). Os primeiros 256 códigos são assinalados para o conjunto padrão inicial. Em um exemplo com códigos de 12 bits, de 0 a 255 são caracteres simples e de 256 a 4095 são para sequências.

Este algoritmo foi o escolhido para comprimir imagens do tipo GIF.

Sejam os algoritmos

```
 1: função COMPRIME {le caracteres e imprime códigos numéricos}
 2: ST ← primeiro caracter
 3: enquanto há caracteres na entrada faça
 4:     CA ← próximo caracter
 5:     se ST + CA está no dicionário então
 6:         ST ← ST + CA
 7:     senão
 8:         imprima o número da linha de ST {vem a ser o codigo de ST}
 9:         adicione ST + CA no dicionário {na primeira linha disponivel}
10:         ST ← CA
11:     fim se
12: fim enquanto
13: imprima o número da linha de ST {vem a ser o código}
14: fim {função}
```

```
 1: função DESCOMPRIME {le códigos numéricos e imprime caracteres}
 2: OL ← primeiro código
 3: CA ← caracter que está na linha OL
 4: imprima CA
 5: enquanto há códigos na entrada faça
 6:     NE ← próximo código
 7:     se NE aponta para linha vazia em dic então
 8:         ST ← caracter (es) da linha OL de dic
 9:         ST ← ST + CA
10:     senão
11:         ST ← caracter (es) da linha NE de dic
12:     fim se
13:     imprima ST
14:     CA ← primeiro caractere de ST
15:     inclua o caracter "OL"+ CA na primeira linha livre de dic
16:     OL ← NE
17: fim enquanto
18: fim {função}
```

Problemas

O clássico problema dos algoritmos baseados em dicionários é a memória necessária. Quanto mais dados do arquivo são lidos, mais seqüências precisam ser armazenadas no dicionário, e com isso, necessita-se de mais memória. O outro problema que isso causa é de espaço de endereçamento: os códigos das entradas no dicionário crescem junto com ele (em geral ocupam $log_2(N)$ bits, onde N é o tamanho, ou número de entradas, do dicionário). Várias abordagens podem ser adotadas para lidar com esse problema, as mais simples são:

- Congelamento do dicionário (quando o dicionário atinge o tamanho limite, ele fica "congelado"e mais nenhuma entrada pode ser adicionada nele).
- Esvaziamento (o dicionário é esvaziado e a compressão começa toda de novo). Essa técnica pressupõe que é mais vantajoso usar a redundância local que a redundância global para a compressão. Corresponde a separar o arquivo em "blocos"comprimidos separadamente.
- Uso de uma política de esvaziamento que seja comum tanto ao compressor quanto ao descompressor (apagar entradas mais antigas, por exemplo).

Exemplo

Para este exemplo:

- Considere textos construidos apenas com as 10 primeiras letras
- O dicionário terá 40 entradas. As 10 primeiras são ABCDEFGHIJ
- As mensagens terão 30 caracteres de comprimento

Seja comprimir a mensagem:

`BDEFFFFFFFECEGGAFHCFJDICEBHJCE`

Após a aplicação do algoritmo acima sobre esta mensagem o dicionário ficará

1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J
11	BD
12	DE
13	EF
14	FF
15	FFF
16	FFFF
17	FE
18	EC
19	CE
20	EG
21	GG
22	GA
23	AF
24	FH
25	HC
26	CF
27	FJ
28	JD
29	DI
30	IC
31	CEB
32	BH
33	HJ
34	JC
35	
36	
37	
38	
39	
40	

1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	

1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	

E o resultado será

`2 4 5 6 14 15 6 5 3 5 7 7 1 6 8 3 6 10 4 9 19 2 8 10 19`

☞ Para você fazer

Coloque em C os codigos resultados da compressão de

`G D J C F H H H H H H H H H H`
`D C A D J F A F H J D G A E G`

C[1]	C[11]	C[15]	C[17]	C[22]

(1 número)

Coloque em D os caracteres resultados da descompressão de

`10 10 4 13 14 1 2 3 4 2 4 7`
`22 22 9 8 23 6 1 7 8 4 9`

D[1]	D[2]	D[16]	D[29]	D[30]

(1 letra)

1


- 1 - /