2

COMPRESIA ENTROPICĂ

2.1 CONCEPTE FUNDAMENTALE

2.1.1 Entropia unei surse staționare

Se consideră un *alfabet* finit **X** constituit din M simboluri şi se defineşte un *mesaj* ca o secvență de simboluri. O sursă discretă de informație este numită stochastică dacă selecția simbolurilor care formează mesajele este făcută din cadrul alfabetului **X** în conformitate cu o distribuție de probabilitate $p_i \triangleq P(x_i)$. Din punct de vedere probabilistic, mulțimea mesajelor poate fi privită ca un proces aleatoriu discret adică, precum o secvență $(\xi_n)_{n=0}^{\infty}$ de variabile aleatoare, fiecare dintre ele luând valori în mulțimea **X** cu probabilitatea de distribuție $\{p_i\}$.

Se presupune în plus că sursa este *staționară*, adică

$$P\{\xi_{i1} = x_1, \ldots, \xi_{ik} = x_k\} = P\{\xi_{i1+h} = x_1, \ldots, \xi_{ik+h} = x_k\} \qquad (2.1)$$

pentru toate numerele naturale $i_1, \ldots, i_k, h$ şi toate $x_1, \ldots, x_k \in \mathbf{X}$.

O caracteristică globală a unei astfel de surse din punctul de vedere al informației este *entropia sursei.*

$$H(x) \triangleq \sum_{i=1}^{M} p_i \log_2 \frac{1}{p_i} \qquad (2.2)$$

care este exprimată în biţi/simbol şi indică cantitatea medie de informaţie a alfabetului **X** [16].

Teorema 2.1. - Entropia H(x) a unei surse cu un alfabet conţinând **M** simboluri, satisface inegalitatea

$$H(x) \leq \log_2 M \tag{2.3}$$

egalitatea fiind satisfăcută pentru simboluri echiprobabile.
În scopul transmiterii la distanţă sau pentru stocarea informaţiei sursei este eficient să se treacă de la reprezentarea oferită de alfabetul sursei **X** la o reprezentare printr-un alt alfabet **Y**. Procesul este numit *codarea datelor.* Un cod este o corespondenţă între *mulţimea mesajelor sursei şi mulţimea cuvintelor de cod.*

Se consideră alfabetul sursei **X** constituit din simbolurile x_1, ..., x_M. Fiecărui simbol i se asociază o secvenţă finită de simboluri ale alfabetului codului **Y**. În practică, alfabetul **Y** este constituit din două simboluri **0** sau **1**. În acest fel fiecărui simbol i se asociază o secvenţă finită de biţi numită *cuvânt de cod*.

O codare eficientă (compresie) urmăreşte minimizarea *lungimii medii* a cuvintelor de cod.

$$\overline{n} \triangleq E\{n\} = \sum_{i=1}^{M} p_i n_i \tag{2.4}$$

unde n_i este lungimea cuvântului de cod (număr de biţi) care reprezintă simbolul x_i, iar n este o variabilă aleatoare care reprezintă lungimea cuvântului de cod (luând valoarea n_i cu probabilitatea p_i , i = 1, 2, ..., M).

Codurile realizează o corespondenţă între mesajele sursei (secvenţe de simboluri - cuvinte ale alfabetului sursei) şi cuvintele de cod (secvenţe de biţi - cuvinte ale alfabetului **Y**). După modul de stabilire a acestei corespondenţe codurile pot fi clasificate în:

i) bloc-bloc;
ii) bloc-variabile;
iii) variabile-bloc;
iv) variabile-variabile;

Codurile bloc-bloc sunt coduri care stabilesc o corespondenţă între mesaje ale sursei de lungime fixată şi cuvintele de cod de lungime fixată.

Codurile variabile-variabile stabilesc o corespondenţă între mesaje ale sursei de lungime variabilă şi cuvinte de cod variabile.

■ **EXEMPLUL 2.1**

Se consideră
X = { a, b, c, d, e, f, g, h, spaţiu }
Y = { 0, 1}

Simboluri	Cuvinte de cod
a	000
b	001
c	010
d	011
e	100
f	101
g	110
Spaţiu	111

Tab. 2.1 Tabel de codare bloc-bloc

Codul din tabelul 2.1 este un cod bloc-bloc şi în aceeaşi categorie se încadrează codurile ASCII şi EBCDIC care pun în corespondenţă un alfabet de 64 (sau 256) simboluri cu cuvinte de cod de 6 (sau 8) biţi.

■ **EXEMPLUL 2.2**

Se consideră
X = { a, b, c, d, e, f, g, h, spaţiu }
Y = { 0, 1}

Simboluri	Cuvinte de cod
aa	0
bbb	1
cccc	10
ddddd	11
eeeeee	100
fffffff	101
gggggggg	110
spaţiu	111

Tab. 2.2. Tabel de codare variabile-variabile

Se observă că pentru un mesaj :
aa_bbb_cccc_ddddd_eeeeee_fffffffgggggggg (40 simboluri in total)
se obţine o lungime cod de 30 faţă de o lungime de cod de 120 dacă codarea s-ar fi făcut conform tabelului 2.1 .
Observaţie : S-a folosit simbolul ”_” (underscore) pentru a nota caracterul spaţiu ■

Un cod este *distinct* dacă fiecare cuvânt de cod este discernabil în raport cu celelalte (corespondența între mesajele sursei şi cuvintele de cod este biunivocă).

Un cod distinct este *unic decodabil* dacă fiecare cuvânt de cod este identificabil când se află imersat într-o secvență de cuvinte de cod.

Codurile din Exemplul 2.1 şi Exemplul 2.2 sunt ambele distincte, dar codul din Exemplul 2.2 nu este unic decodabil. Spre exemplu codul 11 poate fi decodat fie ca ddddd fie ca bbbbbb.

Un cod unic decodabil este un *cod prefix* dacă are proprietatea de prefix care pretinde ca nici un cuvânt de cod să nu fie prefixul altui cuvânt de cod. Codul care are drept cuvinte de cod {1, 100000, 00} este un exemplu de cod care este unic decodabil dar care nu are proprietatea de prefix.

Codurile prefix sunt *decodabile instantaneu* adică se bucură de proprietatea că mesajul codat poate fi despărțit în cuvinte de cod fără a fi necesară examinarea în avans (lookahead) a următorilor biți. Deci decodarea este posibilă imediat ce s-a primit ultimul bit al cuvântului de cod fără a mai fi necesară recepționarea altor biți. Spre exemplu dacă se foloseşte codul care are cuvintele de cod {1, 100000, 00}, decodarea mesajului 1000000001 şi identificarea lui 1 ca fiind primul cuvânt de cod necesită examinarea celui de-al zecelea simbol (era posibil ca primul cuvânt de cod să fie 100000).

Un *cod prefix minimal* este un cod prefix cu proprietatea ca dacă x este un prefix al unui cuvânt de cod atunci şi x sigma este fie un cuvânt de cod fie un prefix al unui cuvânt de cod oricare ar fi litera sigma aparținând alfabetului **Y**. Spre exemplu considerând codul {00, 01, 10} se observă că acesta este un cod prefix care nu este minimal. Astfel 1 este prefix al cuvântului de cod 10 şi ar impune ca şi 11 să fie un cuvânt de cod fie un prefix al unui cuvânt de cod valid. Dacă în exemplul de mai sus se înlocuieşte cuvântul de cod 10 cu cuvântul de cod 1 se obține un cod prefix minimal cu cuvinte de cod fie mai scurte.

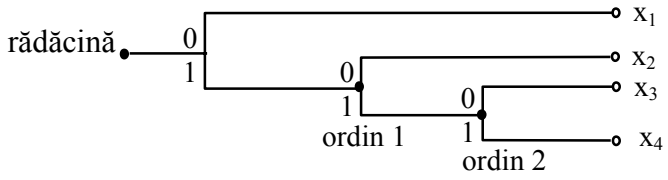
2.1.2 Codarea alfabetului sursei

O reprezentare grafică utilă a codurilor prefix constă în asocierea fiecărui cuvânt de cod cu un nod terminal (frunză) dintr-un arbore binar. Pornind de la rădăcina arborelui primele două ramuri corespund selecției între 0 şi 1 ca primă cifră a cuvântului de cod. Cele două ramuri pornind din nodurile de ordinul întâi corespund celei de a doua cifre a cuvântului de cod şi procesul continuă mai departe (a se vedea fig.2.1).

Cuvintele de cod fiind asignate numai nodurilor terminale, nici un cuvânt de cod nu poate fi prefix al unui alt cuvânt de cod.

O condiție necesară şi suficientă ca un cod să fie cod prefix este dată de teorema următoare[18]:

Fig.2.1 Arbore binar de codare



Simboluri	Cuvinte de cod
x_1	0
x_2	10
x_3	110
x_4	111

Tab.2.3 Tabelul de codare asociată arborelului

TEOREMA 2.2 Inegalitatea lui Kraft

Un cod binar care este un cod prefix având lungimile cuvintelor de cod n_1, n_2, ..., n_M există dacă şi numai dacă

$$\sum_{i=1}^{M} 2^{-n_i} \leq 1 \qquad (2.5)$$

Observaţie:

Dacă $p_i=2^{-n_i}$, i=1, 2, ..., M condiția (2.5) devine o egalitate.

În acest caz se obține pe baza relațiilor (2.2) şi (2.4) că $\overline{n} = H(x)$.

În general însă $\overline{n} \geq H(x)$ şi se urmăreşte ca acest n să fie cât mai aproape de H(x). Un cod prefix poate fi determinat astfel încât n să satisfacă teorema următoare:

TEOREMA 2.3

Se poate determina un cod binar prefix pentru orice alfabet al sursei de entropie H(x); codul având o lungime medie care să satisfacă inegalitatea:

$$H(x) \leq \overline{n} < H(x)+1 \qquad (2.6)$$

2.1.3 Clasificarea metodelor

Există mai multe criterii de clasificare a metodelor de compresie entropică.Toate pleacă însă de la ideea de a reduce redundanța naturală a mesajului inițial, şi acest lucru se poate face doar dacă anumite simboluri sau grupe (subşiruri) de simboluri, pe care le vom numi in general *forme* se repetă. In acest sens, putem clasifica metodele de compresie entropică în două categorii:

a) metode bazate pe frecvenţa de apariţie a unei forme; aceste metode se realizează în principiu în două treceri asupra textului, prima trecere permiţând tocmai stabilirea acestor frecvenţe
b) metode bazate pe apariţia grupată (consecutivă sau diseminată) a aceleiaşi forme; de regulă aceste metode se realizează printr-o singură trecere pe text, eventual cu reveniri pe o zonă predeterminată

După un alt criteriu, cel al construcţiei arborilor de codare-decodare, metodele de compresie entropică pot fi clasificate în:

(i) metode statice
(ii) metode dinamice

O metodă *statică* este o metodă care stabileşte corespondenţa între mulţimea mesajelor şi mulţimea cuvintelor de cod, care nu se modifică în timp. Este fixată înainte de a începe codarea. În acest fel un mesaj dat este reprezentat de un acelaşi cuvânt de cod de fiecare dată când el apare în cadrul mesajului. Se urmăreşte construcţia unui cod unic decodabil care să minimizeze lungimea medie de cod. Un asemenea procedeu este optimal.

O metodă este dinamică, dacă corespondenţa între mulţimea mesajelor şi mulţimea cuvintelor de cod se modifică în timp. Asignarea cuvintelor de cod se bazează pe frecvenţa relativă de apariţie a mesajelor, evaluată până la momentul curent al codării.

Nu se poate face însă o delimitare clară între cele doua grupe de metode, chiar dacă în categoria (*a)* predomină metodele dinamice, iar in categoria (*b)* cele statice.

În continuare vom prezenta mai întâi doi algoritmi clasici de compresie statică bazaţi pe calculul frecvenţei de apariţie a unui simbol: algoritmul Huffman şi algoritmul Shannon-Fano. Pe baza acestora se vor discuta prin comparaţie metode dinamice cu proprietatea de adaptare, se va prezenta o categorie de metode cu aplicabilitate universală, iar în finalul capitolului se vor relua diverse metode bazate pe repetări de forme, cu ideea includerii acestora în tehnici complexe de codare.

2.2 COMPRESIA STATICĂ

2.2.1 Algoritmul Huffman static

Pas 1. Se ordonează cele M simboluri în ordine descrescătoare a probabilităţii.

Pas 2. Se grupează ultimele două simboluri x_{M-1} şi x_M într-un *simbol echivalent* (contopire) cu probabilitatea $p_{M-1}+p_M$.

Pas 3. Se repetă paşii 1 şi 2 până ce se ajunge la un singur *simbol.*

Pas 4. În arborele obținut prin parcurgerea pașilor anteriori se asignează simbolurile binare 0 şi 1 pentru fiecare pereche de ramuri care pleacă dintr-un nod intermediar. Cuvântul de cod pentru fiecare simbol poate fi citit ca secvența binară determinată prin parcurgerea nodurilor dinspre rădăcină până la nodurile terminale asociate simbolurilor.

■ **Exemplul 2.3**

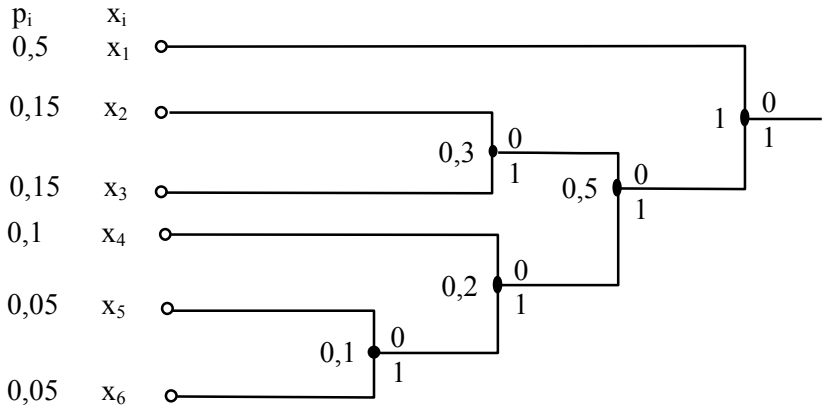


Fig. 2.2 Arbore binar de codare

Se obține codarea Huffman:

Simboluri	Cuvinte de cod
x_1	0
x_2	100
x_3	101
x_4	110
x_5	1110
x_6	1111

Tab. 2.4 Tabelul de codare Huffman

Se constată că entropia H(x) = 2,09 digiți/simbol iar lungimea medie $\overline{n}$ = 2,1 biți/simbol şi cele două valori satisfac inegalitatea (2.6).

■ **Exemplul 2.4**

Pentru codul prezentat în exemplul 2.3 să se folosească arborele pentru a decoda secvența recepționată (compactată) 1100100101110.

Soluție:

Pornind de la rădăcină se urmăresc ramurile la fiecare nod conform biților secvenței până se atinge un nod terminal (deci şi un simbol). Apoi se

reporneşte procedura de la rădăcină. Se constată că secvenţa decodată este $x_4x_1x_2x_3x_4$.

■

Trebuie precizat că algoritmul Huffman nu duce la o codare unică, dar toate codările obţinute conduc la o aceeaşi lungime medie a codului[8].

Algoritmul descris mai sus se ocupă de *codarea simbol cu simbol*. Pentru a obţine performanţe mai bune în ceea ce priveşte $\overline{n}$, se consideră o codare a unui bloc de ν simboluri şi asignarea unor cuvinte de cod pentru noul alfabet $Y=X^{\nu}$, care conţine M^{ν} simboluri y_i. Probabilitatea simbolurilor y_i se obţine ca un produs al probabilităţilor celor ν simboluri din X (independenţa). În baza teoremei 2.3 se construieşte codul pentru alfabetul Y cu lungimea de cod $\overline{n}_{\nu}$ care satisface

$$H(Y) \le \overline{n}_{\nu} < H(Y)+1 \tag{2.7}$$

Având în vedere independenţa simbolurilor, se demonstrează că $H(Y)=\nu H(X)$ şi se obţine din (2.7) relaţia

$$H(X) \le \overline{n}_{\nu}/\nu < H(X)+1/\nu \tag{2.8}$$

Mărimea $\overline{n}_{\nu}/\nu$ este numărul mediu de digiţi/simbol al lui X. Prin alegerea lui ν suficient de mare se poate face ca aceasta să se apropie de H(X).
Se defineşte eficienţa unui cod ca

$$\varepsilon \overset{\Delta}{=} \frac{[\nu H(x)]}{\overline{n}_{\nu}} \tag{2.9}$$

şi redundanţa $\eta \overset{\Delta}{=} 1-\varepsilon$

■ **EXEMPLUL 2.5**

Se consideră un alfabet al sursei $X=\{x_1, x_2, x_3\}$ cu $p_1=0,5$; $p_2=0,3$ şi $p_3=0,2$.
Să construim un nou alfabet $Y=X^2=\{x_1x_1, x_1x_2, ..., x_3x_3\}$.
Să se construiască tabelul simbolurilor y_i, i=1,...,9 cu probabilităţile ordonate descrescător şi apoi pentru acest alfabet să se aplice algoritmul Huffman.

Soluţie:

x_I	p_I	c_I
x_1	0,5	0
x_2	0,3	10
x_3	0,2	11

Tab. 2.5 Tabelul de codare pentru sursa X

$\overline{n} = 0,5 \cdot 2 + 0,5 \cdot 1 = 1,5$ biţi /simbol

0,5 x_1

0,3 x_2

0,2 x_3

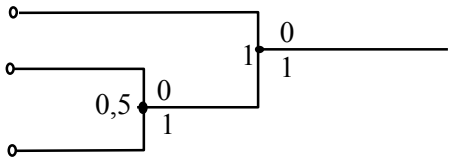
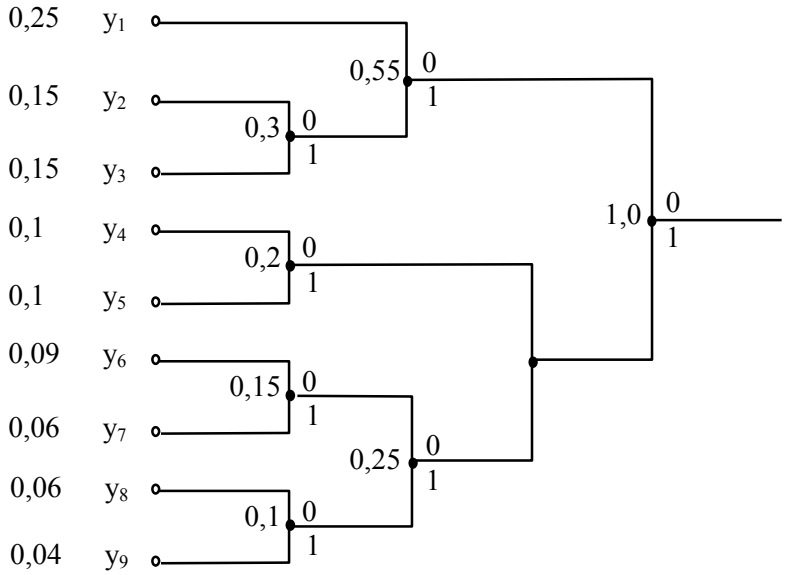


Fig. 2.3 Arbore de codare pentru alfabetul X

y_I	p_I	c_i
$Y_1=x_1\ x_1$	0,25	00
$Y_2=x_1\ x_2$	0,15	010
$Y_3=x_1\ x_3$	0,1	100
$Y_4=x_2\ x_1$	0,15	011
$Y_5=x_2\ x_2$	0,09	1100
$Y_6=x_2\ x_3$	0,06	1101
$Y_7=x_3\ x_1$	0,1	101
$Y_8=x_3\ x_2$	0,06	1110
$Y_9=x_3\ x_3$	0,04	1111

Tab. 2.6 Tabelul de codare pentru sursa Y

Fig. 2.4 Arbore de codare pentru alfabetul $Y = X^2$



Se obțin astfel lungimile medii ca fiind

$$\overline{n}_2 = 0,25 \cdot 2 + 0,5 \cdot 3 + 0,16 \cdot 4 = 2,24 \text{ biți/simbol}$$

$$\overline{n}_2/2 = 1,12 \text{ biți /simbol}$$

2.2.2 Algoritmul Shannon-Fano

Se presupune că cele M simboluri ale alfabetului sursei sunt ordonate în ordine descrescătoare a probabilității: $p_1 \geq p_2 \geq ... \geq p_M$

- Se notează cu $F_i = \sum_{k=1}^{i-1} p_k$, unde i =1, 2, ..., M şi $F_1 = 0$
- Se determină un întreg n_i astfel încât

$$\log_2(1/p_i) \leq n_i < 1+\log_2(1/p_i) \qquad (2.10)$$

pentru fiecare i = 1, 2, ..., M
- Cuvântul de cod asociat simbolului x_i este reprezentarea binară a fracției F_i pe n_i biți

Observație:
Prin reprezentarea binară a unei fracții se înțelege
$b_1b_2...b_k = b_1 \cdot 2^{-1} + b_2 \cdot 2^{-2} + ... + b_k \cdot 2^{-k}$, unde $b_i \in \{0, 1\}$, i = 1, ..., k

Se remarcă următoarele proprietăți ale algoritmului:
(i) Mesajele cu probabilitate mare de apariție sunt reprezentate prin cuvinte cod scurte
(ii) Cuvântul de cod pentru x_i va diferi de toate celelalte cuvinte de cod prin unul sau mai mulți biți. Prin aceasta se asigură decodarea unică. Pentru a demonstra această afirmație, se rescrie relația (2.10) sub forma

$$1/2^{n_i} \leq p_i < 1/2^{n_i+1}$$

Reprezentarea binară a lui F_i va fi diferită de toate celelalte prin unul sau mai mulți biți. Spre exemplu F_i va fi diferită față de F_{i+1} prin cel puțin bitul n_i deoarece $p_i \geq 1/2^{n_i}$. Prin aceasta cuvântul de cod pentru x_{i+1} va diferi de cel pentru x_i prin cel puțin un bit.

■ EXEMPLUL 2.6

Fiind dată o sursă care generează simboluri cu probabilitățile p_i specificate în coloana a doua a tabelului prezentat în continuare, să se aplice algoritmul de compresie Shannon-Fano.

Soluție:

Se remarcă ordonarea simbolurilor în sensul descreşterii probabilităților de apariție (coloanele 1 şi 2).

Se determină numărul de biți ai lungimii de reprezentare, completându-se astfel coloana 3 a tabelului.

Pentru simbolul x_1 numărul de biți se determină pe baza relației:

$$\log_2(128/27) \le n_1 < 1+\log_2(128/27)$$

sau

$$2{,}245 \le n_1 < 3{,}245$$

deducându-se că $n_1 = 3$ biți.

Cuvântul de cod c_1 se obține din $F_1 \triangleq 0$. În concluzie $c_1 = 000$. Pentru x_2 se determină $n_2 = 3$ biți și $F_2 = \sum_{i=1}^{1} p_i = p_1 = 27/128$. Reprezentarea binară a lui 27/128 este 0011011. Prin trunchierea acestei reprezentări la 3 biți se obține cuvântul de cod 001.

x_i	p_i	n_i	F_i	Reprezentare binară	cod c_i
x_1	27/128	3	0	.0000000	000
x_2	27/128	3	27/128	.0011011	001
x_3	9/128	4	54/128	.0110110	0110
x_4	9/128	4	63/128	.0111111	0111
x_5	9/128	4	72/128	.1001000	1001
x_6	9/128	4	81/128	.1001001	1010
x_7	9/128	4	90/128	.1011010	1011
x_8	9/128	4	99/128	.1100011	1100
x_9	3/128	6	108/128	.1101100	110110
x_{10}	3/128	6	111/128	.1101111	110111
x_{11}	3/128	6	114/128	.1110010	111001
x_{12}	3/128	6	117/128	.1110101	11010
x_{13}	3/128	6	120/128	.1111000	111100
x_{14}	3/128	6	123/128	.1111011	111101
x_{15}	2/128	6	126/128	.1111110	111111

Tab. 2.7 Tabelul de codare Shannon-Fano

Se constată că numărul de biți per simbol este $\overline{n}_v = 3{,}89$ biți / simbol și se obține o eficiență $\varepsilon = 62{,}4\%$, entropia fiind de 2,433 biți / simbol. Codarea unei secvențe x_8, x_5, x_7, x_2 este făcută, conform tabelului, în șirul de biți 110010011011001.

Pentru decodare, decodorul va examina mai întâi primii 3 biți 110 (lungimea minimă a acestui cod), va concluziona că nu este cuvânt de cod și va încerca un grup de 4 biți 1100 și în acest moment se va decoda simbolul x_8 și procedura va fi reluată începând cu cel de al cincilea bit.

■

O formulare alternativă a algoritmului Shannon-Fano este:

Pas 1: O listă de simboluri cu probabilități date se sortează în ordine descrescătoare a probabilităților;

Pas 2: Se divide lista în două părți, astfel încât probabilitatea cumulată a porțiunii superioare să fie cât mai apropiată de probabilitatea cumulată a părții inferioare;

Pas 3: Părții superioare i se asignează o cifră binară 0, iar părții inferioare cifra binară 1;

Pas 4: Se aplică recursiv paşii 2 şi 3 acestor două jumătăți divizându-se şi analizând biții codului până când fiecare simbol a devenit o frunză a arborelui

In exemplul următor este ilustrată aplicarea algoritmului Shannon-Fano în această formulare alternativă.

■ **EXEMPLUL 2.7**

x_i	p_I	c_I
a	1/2	0
b	1/4	10
c	1/8	110
d	1/16	1110
e	1/32	11110
f	1/32	11111

Tab. 2.8 Tabelul de codare Shannon-Fano

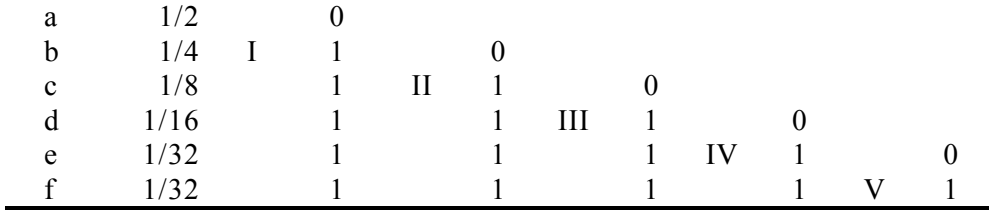


Fig. 2.5 Arborele binar ataşat algoritmului Shannon-Fano (formulare alternativă)

2.2.3 Comparație între metodele de compresie statică

Compresia datelor constă în asocierea dintre un simbol sau o secvență de simboluri şi un cuvânt de cod. Această asociere este efectuată pe baza unui model. Modelul este o colecție de date şi reguli folosite pentru procesarea simbolurilor de intrare.

Un program foloseşte modelul pentru a defini probabilitățile pentru fiecare simbol şi codorul generează un cod corespunzător bazat pe aceste probabilități.

Folosirea unui algoritm de tipul Huffman presupune procesul de compresie ca fiind scindat în

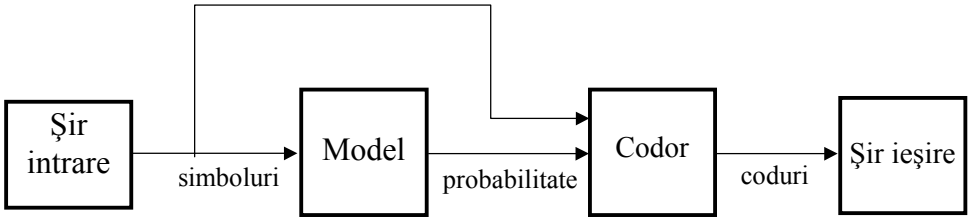


Fig. 2.6 Sistem de codare entropică

O măsură a performanțelor obținute printr-un algoritm de codare este *redundanța*, definită ca diferența dintre lungimea medie a codului şi conținutul de informație pe simbol (entropia).

$$\eta \triangleq \bar{n} - H(X) = \sum_{i=1}^{M} p_i n_i - \sum_{i=1}^{M} p_i \log_2 \frac{1}{p_i} \qquad (2.11)$$

Un cod care are o lungime medie minimă pentru o distribuție discretă de probabilitate dată, se numeşte un cod de *redundanță minimă* (minimum redundancy code).

Pentru a descrie redundanța determinată de proprietățile locale ale unei distribuții de probabilitate se utilizează termenul de redundanță locală. Deşi modelul utilizat pentru analiza algoritmilor de compresie presupune o distribuție aleatoare în ansamblul simbolurilor generate de sursă, există uneori o tendință de grupare a simbolurilor în secvențe predictibile. Prezența acestor secvențe predictibile poate fi exploatată pentru a minimiza redundanța locală.

Un cod este *asimptotic optimal* dacă are proprietatea că pentru o distribuție de probabilitate dată raportul dintre lungimea medie a codului şi entropie tinde către unitate când entropia tinde către infinit. Prin aceasta, optimalitatea asimptotică garantează că lungimea medie a codului se apropie de minimul teoretic[5].

Eficiența compresiei poate fi măsurată prin raportul de compresie (compression ratio). Acesta poate fi definit în mai multe feluri:

$$(\text{i}) \qquad C = \frac{\text{lungimea medie a mesajului}}{\text{lungimea medie a codului}}$$

definiție care pune în evidență comparația dintre lungimea mesajului codat cu lungimea mesajului original.

Considerând exemplul 7, codarea Huffman a mesajului din exemplul 2.2 conduce la 117 biți sau 2,9 biți / simbol.

Dacă se consideră o reprezentare primară a mesajului pe 6 biți în cod ASCII, atunci lungimea medie este de evident tot de 6 biți. Se obține un raport de compresie C=6/2,9=2,06 sau echivalent mesajul compactat reprezintă 49% din cel original (codat ASCII).

(ii) $C = (S - O - OR)/S$

În această definiție:

S - lungimea mesajului generat de sursă;

O - lungimea mesajului codat (compactat);

OR - număr de biți necesar a fi transmis de către codor către decodor pentru a-i permite decodarea. Această cantitate este o *penalitate* a algoritmului de codare. Intenția este de a se oferi o măsură a dimensiunii totale a mesajului codat (sau a fişierului compactat).

Algoritmul Huffman generează un cod optimal (redundanță minimă). Trebuie subliniat că afirmația că o codare Huffman conduce la cel mai bun raport de compresie posibil este corectă cu condiția ca fiecărui simbol să i se asocieze un cuvânt de cod unic şi rezultatul compresiei este obținut prin concatenarea tuturor cuvintelor de cod.

O margine superioară a redundanței unui cod Huffman este $p(n)+0,086$ unde $p(n)$ este probabilitatea cea mai mică a simbolurilor sursei.

Codarea Shannon-Fano poate conduce la soluții suboptimale în timp ce codarea Huffman este întotdeauna optimă.

■ **EXEMPLUL 2.8**

Se consideră o sursă pentru care se efectuează atât codarea Shannon-Fano cât şi codarea Huffman. Se obțin rezultatele formulate în tabelul 2.9 .

Simbol	p_i	Shannon-Fano	Huffman
x_1	0,35	00	1
x_2	0,17	01	011
x_3	0,17	10	010
x_4	0,16	110	001
x_5	0,15	111	000

Tab. 2.9 Tabel comparativ Shannon-Fano/Huffman

Lungimea medie a codului Shannon-Fano este 2,31 iar cea a codului Huffman este 2,30

■

O margine superioară a redundanței unui cod Shannon-Fano este 1.

Este important de remarcat că definiția redundanței dată prin relația (2.11) ignoră creşterea dimensiunii mesajului codat datorită tabelului ce conține codurile alfabetului sursei (exceptând cazul unei codări dinainte convenită). În

cadrul acestei tabele pot apare de exemplu cei $M \log_2 M$ biți necesari codării alfabetului sursei cu M simboluri.

Pentru un cod Shannon-Fano se poate transmite lista cuvintelor de cod ordonate astfel încât ele să corespundă listei alfabetului sursei. În acest caz creşterea mesajului este de $\sum_{i=1}^{M} l(x_i)$ unde $l(x_i)$ este lungimea cuvântului de cod asociat simbolului x_i.

Pentru un cod Huffman se poate transmite o codare a arborelui asociat, codare ce necesită $\log_2 4^n = 2n$ biți.

În cele mai multe cazuri dimensiunea totală a mesajului este foarte mare astfel încât numărul de biți necesar codării alfabetului sursei este nesemnificativ.

Dacă se acceptă un cod suboptimal, această creştere poate fi evitată printr-o înțelegere prealabilă între emițător şi receptor asupra codării. Astfel, se poate utiliza, spre exemplu, o codare Huffman bazată nu pe frecvențele de apariție a simbolurilor în mesajul respectiv, ci pe statistica acestor simboluri în clasa de mesaje din care face parte mesajul curent. În acest caz atât emițătorul cât şi receptorul cu acces la o carte de coduri având k tabele de corespondențe; una pentru o sursă Pascal, una pentru un text în limba engleză etc. La expedierea unui mesaj este necesar a se specifica care dintre aceste tabele este folosită. Această selecție necesită numai $\log_2 k$ biți. Dacă se presupune că este posibilă identificarea unor clase având caracteristici relative stabile, o asemenea abordare are avantajul diminuării substanțiale a redundanței fără a mai fi necesară transmiterea tabelului de codare. În plus efortul de calcul statistic este amortizat prin utilizarea rezultatelor în mod repetat.

Există evident un risc în utilizarea acestei metode, acesta derivând din identificarea corectă a clasei din care face parte mesajul şi din considerarea unor eşantioane reprezentative statistic pentru elaborarea tabelelor de coduri.

Cercetările experimentale desfăşurate asupra codării bazate pe cărțile de coduri, în comparație cu codările bazate pe codarea Huffman au arătat că:

(a) aplicarea unei codări conforme cu o carte de coduri, elaborate pentru o clasă de texte sursă conduce la rezultate foarte apropiate de acelea obținute printr-o codare Huffman.
(b) este posibil să se elaboreze o carte de coduri cu un univers de aplicabilitate suficient de larg (surse Pascal sau C de exemplu) care să conducă la performanțe bune în raport cu codarea Huffman[28].

2.3 METODE DE COMPRESIE DINAMICE

2.3.1 Algoritmul Huffman dinamic

Codurile dinamice sunt denumite în literatură şi drept coduri adaptive deoarece sunt capabile de a se adapta la modificările ansamblului în timp. Metodele de compresie adaptivă pot exploata fie modificările de comportament al sursei fie localitatea referirii. Prin localitatea referirii (locality of reference) se înţelege tendinţa, comună într-o varietate de texte, ca un anumit cuvânt să apară în mod frecvent într-o anumită zonă şi apoi să nu mai fie utilizat pentru perioade lungi de timp.

Toate metodele adaptive sunt metode de o singură trecere (one-pass). Metoda Huffman statică sau Shannon-Fano sunt metode de două treceri (two-passes), una pentru a determina probabilităţile şi a determina codurile asociate simbolurilor şi o a doua pentru a efectua codarea propriu-zisă.

O metodă de compresie este *dinamică* dacă între mulţimea simbolurilor sursei şi mulţimea cuvintelor de cod se stabileşte o corespondenţă care se modifică în timp. De exemplu o codare Huffman dinamică presupune calculul aproximării probabilităţilor de apariţie *"din mers"* (on the fly), pe măsură ce este generat ansamblul. Asignarea de cuvinte de cod se bazează pe frecvenţele relative de apariţie evaluate la un anumit moment de timp. Astfel un simbol poate fi reprezentat printr-un cuvânt de cod mai scurt la început deoarece a apărut mai frecvent la începutul mesajului deşi frecvenţa lui de apariţie în ansamblu este mică. Exemplul următor permite o comparaţie între aplicarea metodelor de compresie Huffman statică şi dinamică.

■ **EXEMPLUL 2.9**

Pentru sursa care generează mesajul de la exemplul 2.2 se aplică metoda Huffman statică obţinând codarea:

Simbol	Probabilitate	Cod
a	2/40	1001
b	3/40	1000
c	4/40	011
d	5/40	010
e	6/40	111
f	7/40	110
g	8/40	00
Spaţiu	5/40	101

Tab. 2.10 Tabel de codare Huffman static

Dacă însă se consideră o secvenţă aa_bbb care constituie începutul mesajului din exemplul 2.2., alocarea unor cuvinte de cod pe baza frecvenţei de apariţie de până la momentul curent face ca de exemplu pentru "b" să fie luată în considerare frecvenţa de apariţie 3/6 care este mai mare decât aceea pentru

"spaţiu", deşi în ansamblul mesajului "b" are o frecvenţă de apariţie mai mică decât aceea pentru "spaţiu". Se obţine tabelul 2.11:

Simbol	Probabilitate	Cod
a	2/6	10
b	3/6	0
spaţiu	1/6	11

Tab. 2.11 Tabel de codare Huffman dinamic

2.3.2. Algoritmul FGK

Codarea Huffman adaptivă a fost concepută în mod independent de Faller şi Gallager, iar Knuth a oferit rafinări ale algoritmului astfel încât algoritmul este denumit FGK. Această codare stabileşte o corespondenţă între mulţimea mesajelor sursei şi cuvintele de cod, pe baza unei estimări curente (efectuate până la acel moment) a frecvenţelor de apariţie a mesajelor. Codul este adaptiv, modificându-se astfel încât să rămână optimal în raport cu estimarea curentă. Prin aceasta, codarea Huffman adaptivă exploatează localitatea surselor. În esenţă acţiunea codorului este de a "*învăţa*" caracteristicile sursei. Decodorul trebuie să înveţe împreună cu codorul, efectuând mereu o reactualizare a arborelui Huffman astfel încât acesta să fie în sincronism cu codorul.

Un alt avantaj al acestor algoritmi este acela că ei pretind numai o singură parcurgere a datelor de compactat. Este interesant de remarcat că este posibil ca performanţa de compactare realizată printr-o asemenea metodă să fie mai bună decât aceea obţinută printr-o codare Huffman statică. Această afirmaţie nu contrazice caracterul optimal al metodelor statice deoarece metodele statice sunt optimale în ansamblul metodelor care presupun o corespondenţă invariantă în timp între simboluri şi cod. Metoda adaptivă Faller, Gallager şi Knuth constituie baza utilitarului de compactare UNIX, performanţa acestuia fiind un raport de compresie de 30 - 40%.

Baza algoritmului FGK o constituie *proprietatea de înfrăţire* (sibling property). Un cod binar are proprietatea de înfrăţire dacă fiecare nod (cu excepţia rădăcinii) are un frate (un nod care are un acelaşi părinte) şi dacă nodurile pot fi listate în ordinea descrescătoare a ponderilor astfel încât fiecare nod să fie adiacent fratelui său. Gallager a demonstrat că un cod binar prefix este un cod Huffman dacă şi numai dacă arborele asociat codului are proprietatea de înfrăţire. În algoritmul FGK atât emiţătorul cât şi receptorul menţin câte un cod Huffman modificat dinamic. Frunzele arborelui asociat codului reprezintă mesajele sursei, iar ponderile frunzelor reprezintă frecvenţele de apariţie ale simbolurilor până la acel moment de timp.

■ **EXEMPLUL 2.10**

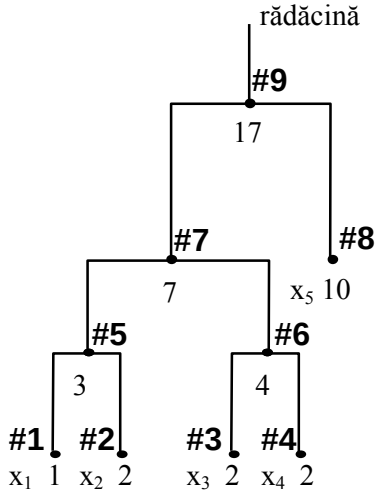


Fig.2.7 Arbore cu proprietatea de înfrățire

Numerotarea nodurilor s-a făcut în ordinea crescătoare a ponderilor pornind de la 1. Se constată că acest arbore respectă proprietatea de înfrățire, fiecare nod fiind adiacent fratelui său.

Actualizarea arborelui trebuie să conserve proprietatea de înfrățire astfel încât şi arborele actualizat să fie în continuare un arbore Huffman.

Actualizarea arborelui constă în două operații de bază. Prima este incrementarea ponderilor. Ea este începută de la frunze şi este transmisă către nodurile superioare (fiecare pondere a unui nod părinte este suma ponderilor copiilor săi). Această incrementare se transmite către nodul rădăcină şi este efectuată cu câte o unitate. A doua operație necesară în actualizarea arborelui este necesară când incrementarea ponderii nodului determină o violare a proprietății de înfrățire. Aceasta apare în cazul în care nodul care este incrementat are aceeaşi pondere ca şi următorul nod de mai sus din cadrul listei. În această situație este necesar să se mute nodul afectat pe o poziție mai înaltă în listă. Aceasta se face detaşând nodul din arbore şi schimbându-l cu unul de mai sus din cadrul listei.

■ **EXEMPLUL 2.11**

Se foloseşte arborele din exemplul anterior.

Se consideră că simbolul x_1 este recepționat încă o dată. Se efectuează o actualizare a contoarelor arborelui.

Sunt marcate nodurile în care s-au făcut incrementările, pornind de la frunză şi ajungând la rădăcină.

Se constată că în acest caz, în urma tuturor incrementărilor, nu s-a violat proprietatea de înfrățire.

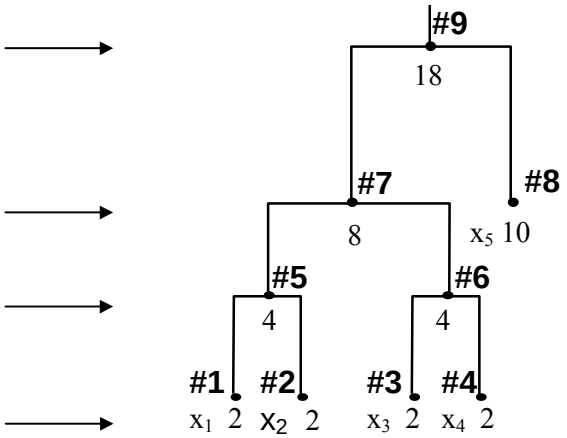


Fig 2.8 Arborele din exemplul 2.10 după incrementare

■ **EXEMPLUL 2.12**

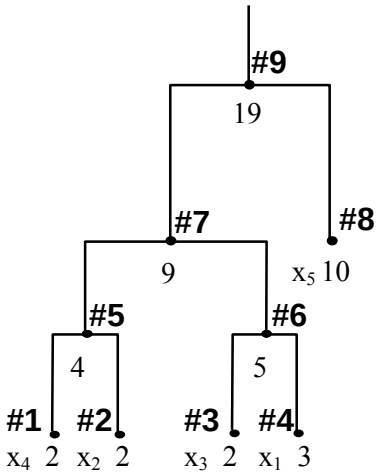


Fig 2.9 Arborele obținut prin interschimbarea nodurilor

Se recepționează încă un simbol x_1 şi în acest caz după incrementare este necesar să se interschimbe nodurile x_1 şi x_4 între ele. De ce între x_1 şi x_4 trebuie efectuată schimbarea ?

Pentru a micşora numărul operațiilor de schimbare se impune să se schimbe doar două noduri. Dacă nodul care tocmai a fost incrementat are ponderea W+1, următorul pe listă are ponderea W. Pot fi mai multe noduri care au ponderea W. Procedura de schimbare efectuează schimbarea cu *ultimul* nod care are ponderea W. Noul arbore va avea deci un şir de unul sau mai multe noduri cu ponderea W urmat de un nod cu pondere W+1.

După această schimbare trebuie incrementat nodul (nodurile) părinți şi eventual trebuie refăcut testul de bună ordonare cu eventuale schimbări.

■ **EXEMPLUL 2.13**

Dacă x_1 este recepționat în continuare de încă două ori, el va căpăta ponderea 5. În acest moment schimbarea este importantă el fiind schimbat cu nodul intern care are ponderea 4 (fost nr. 6).

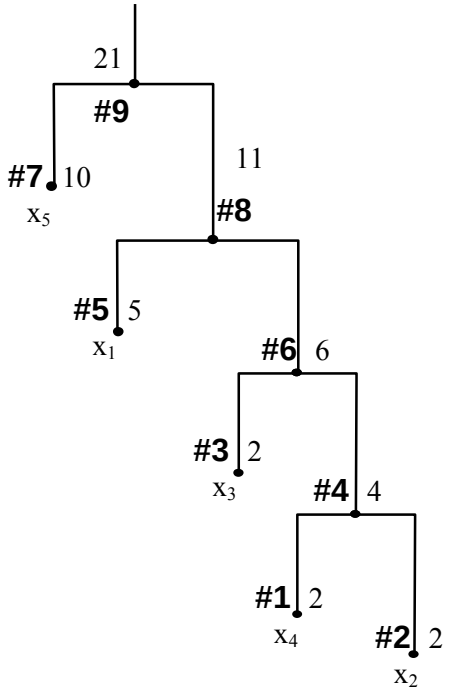


Fig 2.10 Arborele obținut după ce ponderea lui x_1 a devenit 5

■ **EXEMPLUL 2.14**

Să se aplice algoritmul FGK mesajului prezentat în exemplul 2.2

Soluție:

Aplicând algoritmul FGK mesajului din exemplul 2.2 se constată că numărul de biți necesari codării este 129. Algoritmul Huffman static a realizat codarea cu 117 biți. Totuşi numărul de biți necesari implementării efective a decodării (tabelul de codare, arborele) pentru o metodă adaptivă este mai mic decât pentru o metodă statică.

În cazul unei compresii adaptive este nevoie să se folosească $n \cdot \log_2 n$ biți pentru a reprezenta fiecare dintre cele n simboluri distincte ale sursei atunci când ele apar pentru prima oară. Această estimare este de fapt conservatoare; emițătorul poate transmite poziția simbolului în lista

simbolurilor rămase în loc de a transmite un cod unic pentru fiecare dintre cele n simboluri şi prin aceasta se economisesc în medie câțiva biți. Pentru codarea statică este necesară şi transmiterea formei arborelui. O reprezentare eficientă a arborelui s-a arătat că necesită 2n biți. Algoritmul FGK este comparabil ca performanță de compresie cu algoritmul Huffman static, dacă se ia în considerare şi codarea arborelui[17].

2.3.3 Aspecte ale implementării algoritmului Huffman

Evaluarea performanțelor de compresie se face adesea în practică folosind un set de date de test standard. Spre exemplu, pot fi considerate următoarele tipuri de date de test:
- text (programe, manuscrise);
- date binare (baze de date, fişiere executabile);
- fişiere grafice (format raw screen - dump).

În analiza eficienței programelor de compresie trebuiesc avute în vedere:
- cantitatea de memorie necesară efectuării compresiei;
- timpul necesar efectuării compresiei;
- raportul de compresie obținut.

Raportul de compresie se calculează folosind formula:

compression_percentage = (1-(compressed_size/raw_size))*100

Observații

(i) Conform acestei formule lipsa compresiei va conduce la o cifră a raportului de compresie 0.

(ii) O compresie care conduce la un fişier de dimensiune 0 conduce la o cifră a raportului de compresie de 100.

Implementarea computațională a algoritmilor de compresie necesită reformularea şi rafinarea lor în scopul obținerii unei soluții viabile. Pentru a permite cititorului formarea unei imagini corecte este făcut în continuare un studiu de caz pentru algoritmul Huffman[20].

O reformulare a algoritmului Huffman mai apropiată de necesitățile implementării este următoarea:

Simbolurile sunt structurate într-un şir de noduri frunză care sunt conectate printr-un arbore binar. Fiecare nod are o pondere care este frecvența de apariție a simbolului. Construirea arborelui se face parcurgând următoarele etape

- Se identifică cele două noduri care au cele mai mici ponderi;
- Se creează un nod părinte din aceste două noduri. I se asociază acestuia o pondere egală cu suma ponderilor copiilor;
- Unuia dintre nodurile copil i se asociază bitul 0 iar celuilalt bitul 1;
- Paşii anteriori sunt repetați până ce rămâne un singur nod liber. Acest nod liber este denumit nod rădăcină.

Structura de date implementată trebuie să modeleze arborele binar. Fiecare nod conține informație şi anume:

- ponderea nodului (frecvența de apariție);
- pointerii către nodurile copii;
- nodurile frunză au valoarea simbolului asociat acelei frunze.

Structura nodului este descrisă în C:

```
typedef struct tree_node(
     unsigned int count;
     unsigned int saved_count;
     int child_0;
     int child_1;
) NODE;
```

Se observă că lipseşte informație despre valoarea nodului frunză. Această informație derivă din poziția nodului în array-ului indicatorul de *end_of_stream* este ultimul cod scris în cadrul şirului de coduri şi arată că nu mai sunt date. Este necesar datorită structurii orientate către bit a datelor comprimate şi este un mijloc de a determina când s-a atins starea *end_of_file*. Altă metodă de a indica acest lucru este să se codeze lungimea fişierului ca prefix la datele comprimate.

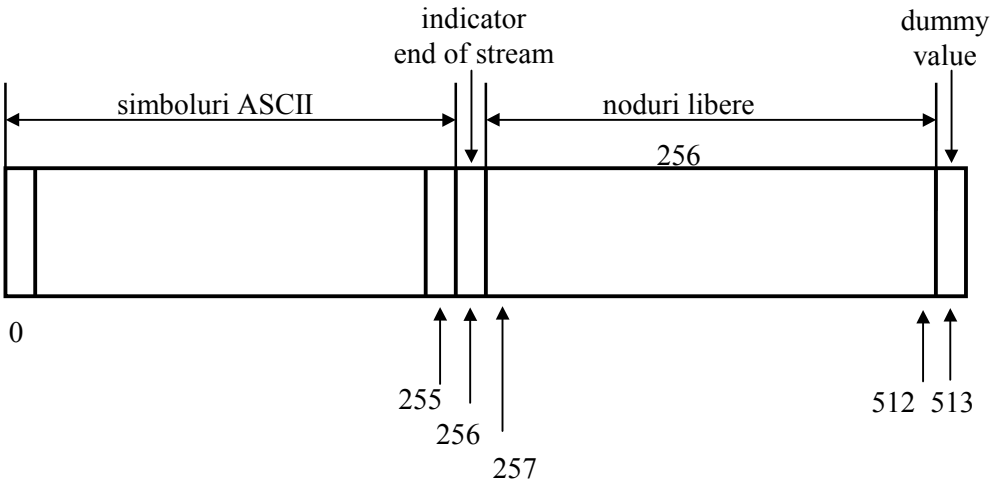


Fig 2.11 Nodurile folosite pentru a descrie arborele Huffman

Se constată că dimensiunea maximă a arborelui (numărul de noduri) este de 513, aceasta rezultând din cele 256 noduri interne plus cele 257 noduri frunză. Afirmația de mai sus este valabilă pentru cazul în care toate cele 256 simboluri sunt prezente. Un al 514-lea nod este folosit pentru comparație la crearea arborelui (dummy value).

Contorizarea simbolurilor

Se numără apariția simbolurilor şi se contorizează pentru fiecare simbol . Apoi se face o scalare prin împărțirea la cel mai mare contor existent. Se reduce astfel dimensiunea de memorare a contoarelor la 16 biți (de la 32). Trebuie avut grijă ca prin scalare să nu se obțină un contor zero, pentru că în acest caz simbolului nu i se asociază cod (soluție: se forțează un „1")

Programul de expandare are nevoie de o copie a arborelui Huffman identică cu aceea folosită de codor pentru a putea efectua operația duală.

Transmiterea arborelui

Sunt folosite in practica următoarele variante de decodare:

- varianta de decodare - pe baza arborelui binar;
- varianta de decodare - pe baza contorilor asociați nodurilor frunză, rezultând 256 noduri.

Un rafinament; în multe cazuri contorii unor caractere sunt nuli şi în general sunt nuli pe grupe contigue (frecvent circa 100 simboluri). De aceea se foloseşte un tabel în care este indicat primul caracter al grupei, ultimul caracter şi apoi contorii corespunzători.

Crearea arborelui

În nodul 513 se introduce un conținut de 65535. Când se începe căutarea nodului cu cea mai mică pondere se inițializează cu acesta pentru că orice nod activ valid este cu un conținut mai mic. Înainte de comparație se setează un index către următorul nod liber, care este 257.

Se intră apoi într-o buclă infinită (DO WHILE), în care se caută cele două noduri active cu ponderea cea mai scăzută. Dacă există numai unul, s-a terminat procesul rezultă că se iese din buclă. Dacă există două, se creează un nou nod folosind un index de următor nod liber, pointerii către nodurile copii sunt poziționați către cele două noduri descoperite, ponderea este suma lor, iar nodurile sunt făcute inactive prin ponderea lor setată la 0.

Observație: Variabila *Saved count* foloseşte la depanarea programului.

Folosirea arborelui

Decodarea se face pornind de la nodul rădăcină. Se citeşte un bit. Dacă este 0, următorul nod este cel pointat de indexul child_0. Dacă este 1, următorul nod este cel pointat de indexul child_1. Dacă noul nod este 256 sau mai puțin, rezultă că este nod frunză, deci se poate genera simbolul. Dacă este 256 s-a atins capătul fişierului.

Compresia cu ajutorul arborelui este mai dificilă. Structura nu este de ajutor; ar trebui să se salveze la contopire un pointer către nodul părinte. În practică se recurge la soluția mai simplă a tabelului de codare.

Se pleacă de la nodul rădăcină şi se adaugă un 0 sau 1 la fiecare bifurcație. Când se atinge o frunză, se depune codul pentru acea frunză în tabelul de coduri şi se revine la nodul anterior, unde se explorează cealaltă ramură.

10	Primul (LF)
13	Ultimul (CR)
19	Contor pentru 10
0	Contor pentru 11
0	Contor pentru 12
19	Contor pentru 13
32	Primul (‘ ’)
126	Ultimul (‘-’)
10	Contor pentru 32
18	Contor pentru 33
11	Contor pentru 126
0	Terminator

Tab. 2.12 Tabel de contoare

Observaţie:
Terminatorul este 0, dar la început 0 nu este tratat drept terminator.

O comparaţie între codarea Huffman statică şi codarea Huffman dinamică (adaptivă) trebuie să plece de la următoarele constatări.

Codarea Huffman statică foloseşte un model de ordin 0 (context free). Asta înseamnă că probabilitatea unui caracter este calculată fără a lua în considerare caracterele care îl preced în cadrul mesajului.

Codarea statică necesită pentru decodare ≈ 250 de octeţi (asigurarea corespondenţei). Această cantitate este neglijabilă în raport cu dimensiunile fişierului.

Dacă se doreşte îmbunătăţirea capacităţii de compresie prin considerarea unui model de ordin 1 este necesară transmiterea a 256 de tabele de probabilitate.

În concluzie, cu excepţia cazului unor fişiere foarte mari, rezultă că metoda nu este eficientă.

Codarea adaptivă rezolvă paradoxul aparent: compresie mai bună, necesită acumulare de elemente statistice mai bogate dar aceasta înseamnă transmiterea unui model mai mare.

Codarea adaptivă permite adoptarea unui model de ordin superior, nu foarte greoi şi constă în acordarea arborelui Huffman din mers (*on the fly*) bazată pe datele văzute anterior şi fără o cunoaştere a statisticii ulterioare.
Observaţie:

Codarea adaptivă nu este limitată la codarea Huffman. În principiu orice formă de codare poate fi modificată pentru a fi folosită ca metodă adaptivă.
Principial compresia constă în:

```
initialize_model();
do {
    c=getc(input);
    encode(c.output);
    update_model(c);
} while(c != EOF);
```

Principial decompresia poate fi descrisă prin:

```
initialize_model();
while((c = decode(input))!= EOF){
    putc(c.output);
    update_model(c);
}
```

La generarea primului simbol codat de către codor, decodorul trebuie să fie capabil a-l interpreta. Compresorul şi decompresorul pornesc cu modele identice în a coda şi a decoda. Caracterul adaptiv al algoritmului derivă din actualizarea arborelui Huffman pe măsura acumulării de caractere.

Actualizarea arborelui Huffman poate fi descrisă principial:

```
update_model(int c)
{
    counts[c]++;
    construct_tree(counts);
}
```

Un asemenea algoritm este evident foarte ineficient în timp. O soluţie de construcţie a arborelui mai eficientă se bazează pe proprietatea de înfrăţire (sibling), soluţie care a fost prezentată în cadrul Exemplelor 2.10 ÷ 2.14.

Algoritmul de incrementare a contorului unui nod este descris prin:

```
for (;  ;) {
    increment nodes[node].count;
        if(node == ROOT)
            break;
        if(nodes[node].count > nodes[node+1].count)
            swap_nodes();
        node = nodes[node].parent;
}
```

Procedura swap_nodes( ) trebuie să parcurgă lista nodurilor până la determinarea nodului cu care trebuie făcută schimbarea.

```
swap_node = node+1;
while(nodes[swap_node+1].count < nodes[node].count)
    swap_node++;
temp = nodes[swap_node].parent;
nodes[swap_node].parent = nodes[node].parent;
nodes[node].parent = temp
```

2.4. TEHNICI DE COMPRESIE BAZATE PE CODURI UNIVERSALE ŞI REPREZENTAREA NUMERELOR NATURALE

Un cod este *universal* dacă stabileşte o corespondenţă între simbolurile sursei şi cuvintele de cod astfel încât lungimea medie a codului este mărginită de $c_1(H(X)+c_2)$. Aceasta înseamnă că, fiind dată o sursă arbitrară cu o entropie nenulă, un cod universal obţine o lungime medie a codului care este un multiplu al lungimii optime. Compresia obţinută de un cod universal depinde în mod clar de mărimea constantelor c_1 şi c_2. Se observă că un cod universal cu $c_1 = c_2 = 1$ este un cod asimptotic optimal. Un avantaj al codurilor universale asupra codurilor Huffman este că nu necesită cunoaşterea exactă a probabilităţilor cu care apar simbolurile sursei ci numai ordinea probabilităţilor simbolurilor sursei. Un alt avantaj al codurilor universale este că ele sunt fixate şi de aici caracterul lor universal. Codarea şi decodarea sunt astfel mult simplificate. Deşi codurile universale pot fi folosite în cadrul metodelor statice în locul codului Huffman, de exemplu; cel mai uzual mod de utilizare al lor este în cadrul metodelor dinamice[3].

Deoarece parametrul esenţial în determinarea codurilor universale este ordinea frecvenţelor simbolurilor sursei, codarea universală poate fi concepută ca fiind o enumerare a mesajelor sursei sau ca o reprezentare a numerelor naturale care constituie un suport al enumerării.

Codurile Elias definesc o secvenţă de coduri universale stabilind o corespondenţă bijectivă între mulţimea numerelor naturale (nenule) şi mulţimea cuvintelor de cod binare[24].

Număr	Cod gama	Cod delta
1	1	1
2	010	0100
3	011	0101
4	00100	01100
5	00101	01101

6	00110	01110
7	00111	01111
8	0001000	00100000
16	000010000	001010000
17	000010001	001010001
32	00000100000	0011000000

Tab. 2.13 Tabel de codare pentru codurile gama şi delta

Codul gama este un cod simplu dar nu este optim. El asociază unui număr natural X (nenul) o codare în binar natural cu un prefix de $[\log_2 X]$ zerouri ([y] - parte întreagă a lui y). Codarea binară a numărului X este făcută pe un număr minim de biţi astfel încât ea începe întotdeauna printr-un "1" care o delimitează de prefix. Se obţine astfel un cod instantaneu decodabil, deoarece lungimea totală a cuvântului de cod este cu o unitate mai mare decât de două ori numărul de zerouri din cadrul prefixului; drept urmare îndată ce a fost determinat primul "1" al cuvântului de cod, lungimea totală a cuvântului de cod este determinată. Codul nu este de redundanţă minimă deoarece raportul dintre lungimea medie a cuvintelor de cod şi entropie tinde către 2 când entropia tinde la infinit.

Codul delta asociază unui număr natural X un cuvânt de cod constând din concatenarea lui gama ($[\log_2 X] + 1$) cu reprezentarea în binar a lui X din care s-a suprimat primul "1".

Lungimea codului rezultat este:

$$[\log_2 (X)] + 2\,[\log_2 (1 + [\log_2 X]\,) + 1] \qquad (2.12)$$

Codul delta este optimal asimptotic deoarece limita raportului dintre lungimea medie a codului şi entropie este 1.

■ **EXEMPLUL 2.15**

Pentru sursa de la exemplul 2.2 să se utilizeze codul Elias.

Soluţie:

Se constată că numărul de biţi necesar transmiterii mesajului de la exemplul 2.2 este de 161 faţă de 117 pentru o codare Huffman.

Simbol	Frecvenţă	Ordin	Cuvânt de cod
g	8	1	delta (1) = 1
f	7	2	delta (2) = 0100
e	6	3	delta (3) = 0101
d	5	4	delta (4) = 01100
Spaţiu	5	5	delta (5) = 01101

c	4	6	delta (6) = 01110
b	3	7	delta (7) = 01111
a	2	8	delta (8) = 00100000

Tab. 2.14 Tabel de codare Elias

■

O altă metodă de codare universală se bazează pe numerele lui Fibonacci. Deşi codurile Fibonacci nu sunt optimale asimptotic, performanţele lor sunt comparabile cu cele ale codurilor Elias, dacă numărul de simboluri ale sursei nu este prea mare. Codurile Fibonacci sunt caracterizate printr-o mare robustețe care se manifestă prin limitarea locală a efectelor erorilor.

Codarea se bazează pe numerele Fibonacci de ordin m ≥ 2. Prin numere Fibonacci de ordin 2 se înțeleg numerele Fibonacci obişnuite: 1, 1, 2, 3, 5, 8, 13,..Numerele Fibonacci de ordin m sunt definite prin relația de recurență:

(i) numerele de la F(-m+1) până la F(0) sunt egale cu 1;

(ii) numărul de ordin k ≥ 1 este suma tuturor celor m numere precedente.

În continuare sunt descrise coduri Fibonacci de ordin 2, extinderea la cele de ordin superior fiind directă. Fiecare număr natural N are o reprezentare binară unică de forma:

$$R(n) = \sum_{i=0}^{k} d(i) \cdot F(i) \text{ unde } d(i) \in \{0,1\} \qquad (2.13)$$

în care k ≤ N; iar F(i) sunt numerele Fibonacci de ordinul 2, astfel încât să nu existe cifre "1" adiacente în această reprezentare. În tabelul următoare sunt indicate câteva astfel de reprezentări:

N	R(N)	F(N)
1	1	11
2	10	011
3	100	0011
4	101	1011
5	1000	00011
6	1001	10011
7	1010	01011
8	10000	000011
16	100100	0010011
32	1010100	00101011

Tab. 2.15 Tabel de codare Fibonacci

Se constată că o asemenea reprezentare nu constituie un cod prefix. De aceea se foloseşte un cod Fibonacci de ordin 2 pentru N definit ca fiind F(N) = D1 unde D = d(0) d(1) d(2)....d(k)...d(i) definit ca mai sus. În consecință codul Fibonacci pentru N este obținut din cel anterior prin schimbarea ordinii biților şi prin apendarea unui "1". Acesta este un cod prefix deoarece fiecare

cuvânt de cod se termină în "11" care nu pot apare în nici o altă poziţie în cadrul unui cuvânt de cod.

Codul Fibonacci de ordinul 2 este universal cu c_1=2 şi c_2=3. Nu este asimptotic optimal deoarece c_1>1. Codurile Fibonacci de ordin superior permit obţinerea unor expresii mai bune decât cele de ordin 2 dacă numărul de simboluri distincte este suficient de mare şi distribuţia de probabilitate este aproape uniformă. Trebuie precizat că nici un cod Fibonacci nu este asimptotic optimal. Codul Elias delta este asimptotic mai scurt decât orice cod Fibonacci pentru un N dat, dar pentru o plajă largă iniţială a lui N codurile Fibonacci sunt mai scurte. În tabelul următor sunt indicate codurile Fibonacci pentru exemplul 2.2 . Numărul de biţi ai codării este 153, deci mai bun decât pentru codul Elias, dar încă departe de cel obţinut prin codare Huffman.

Simbol	Frecvenţă	Ordin	Cuvânt de cod
g	8	1	F(1) = 11
f	7	2	F(2) = 011
e	6	3	F(3) = 0011
d	5	4	F(4) = 1011
spaţiu	5	5	F(5) = 00011
c	4	6	F(6) = 10011
b	3	7	F(7) = 01011
a	2	8	F(8) = 000011

Tab. 2.16 Tabel de codare Fibonacci de ordin doi

2.5 CODAREA ARITMETICĂ

În codarea aritmetică unei surse i se asociază o reprezentare prin intervalul [0, 1). Fiecare simbol al alfabetului sursei micşorează acest interval. Pe măsură ce intervalul devine mai mic, numărul de biţi necesar pentru a-l preciza creşte. Codarea aritmetică presupune un model probabilistic explicit pentru sursă. Sunt folosite probabilităţile simbolurilor sursei pentru a îngusta în mod succesiv intervalul asociat iniţial mulţimii tuturor simbolurilor. Un mesaj cu probabilitate ridicată îngustează acest interval cu mult mai puţin decât un mesaj cu probabilitate crescută, astfel încât un mesaj cu probabilitate ridicată contribuie cu mai puţini biţi la codul asociat mulţimii. Dreapta reală este partiţionată astfel în subintervale bazate pe probabilităţi cumulate.

Pentru a ilustra ideile codării aritmetice se consideră exemplul :

■ **EXEMPLUL 2.16**

Se consideră o sursă care generează simbolurile conform distribuției de probabilitate

Simbol	A	B	C	D	#
Probabilitate	0,2	0,4	0,1	0,2	0,1

Tab. 2.17 Tabelul de probabilitate pentru o sursă

În fig.2.12 este reprezentat procesul de partiționare a dreptei numerelor reale, iar în tabelul 2.18 sunt indicate probabilități asociate plajelor de valori.

Simbolul A corespunde primei porțiuni de 1/5 din intervalul [0,1); simbolul B următoarei porțiuni de 2/5; simbolul D subintervalului de dimensiune 1/5 care începe la 70% din intervalul [0,1) de la stânga la dreapta.

La începerea codării mulțimea simbolurilor sursei este reprezentată prin întregul interval [0,1). Pentru a coda AADB primul A reduce intervalul la [0; 0,2), al doilea A la [0; 0,04) (primii 1/5 din intervalul precedent). Pentru D se reduce intervalul la [0,028; 0,036) (1/5 din intervalul precedent începând la 70% din distanța de la stânga la dreapta).

Simbol	**Probabilitate**	**Probabilitate cumulată**	**Plajă valori**
A	0,2	0,2	[0; 0,2)
B	0,4	0,6	[0,2; 0,6)
C	0,1	0,7	[0,6; 0,7)
D	0,2	0,9	[0,7; 0,9)
#	0,1	1,0	[0,9; 1,0)

Tab. 2.18 Tabelul de codare simboluri-plaje de valori (primul simbol)

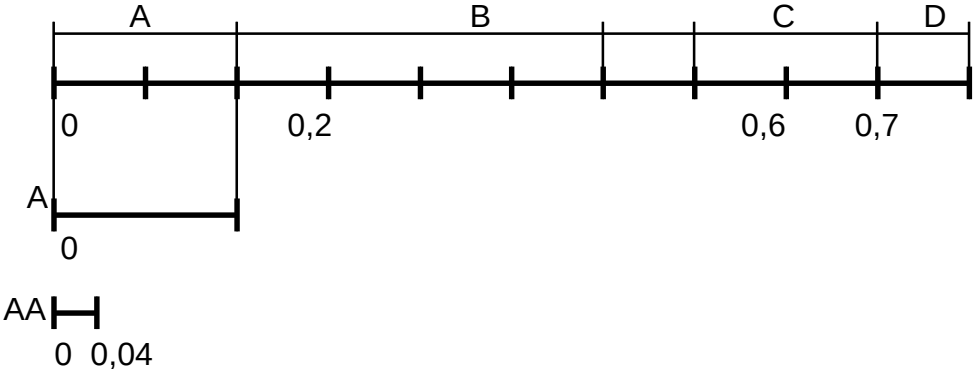


Fig 2.12 Segmentul de dreaptă [0,1) asociat simbolurilor sursei

$(70/100) \cdot (0,4 - 0,0) + 0 = 0,028$
$0,028 + (1/5) \cdot (0,04 - 0,0) = 0,036$

Pentru B se obţine intervalul [0,0296; 0,0328) (2/5 din intervalul precedent începând cu 20% din distanţa de la stânga la dreapta).

$(20/100) \cdot (0{,}36 - 0{,}028) + 0{,}028 = 0{,}0296$
$0{,}0296 + (2/5) \cdot (0{,}36 - 0{,}028) = 0{,}0328$

iar # conduce la un interval final [0,03248; 0,0328)

Intervalul sau orice număr din cadrul intervalului poate fi folosit pentru a reprezenta mulţimea simbolurilor sursei conţinute în mesaj.

Procesul de îngustare a intervalelor poate fi descris prin două ecuaţii:

*newleft = prevleft + msgleft * prevsize*
*newsize = prevsize * msgsize*

Prima ecuaţie stabileşte că punctul din stânga al noului interval este calculat din intervalul precedent şi simbolul curent, astfel pentru D noul punct din stânga este calculat ca fiind deplasat cu 0,7·0,4 (70% din dimensiunea intervalului precedent). Cea de a doua ecuaţie stabileşte dimensiunea noului interval pe baza dimensiunii precedente şi pe baza probabilităţii noului simbol. Pentru D dimensiunea noului interval este 0,04·0,2 şi punctul din dreapta al noului interval este 0,028 + 0,008 = 0,036 (punctul din stânga + dimensiune).

Dimensiunea subintervalului final determină numărul de biţi necesari pentru a specifica un număr în acea plajă. Numărul de biţi necesari pentru a specifica un subinterval al lui [0, 1) de dimensiune s este $-\log_2 s$. Deoarece dimensiunea subintervalului final este produsul probabilităţilor simbolurilor din cadrul mesajului ($\prod_{i=1}^{N} p(x_i)$, unde N este lungimea mesajului), se obţine:

$$-\log_2 s = -\sum_{i=1}^{N} \log_2 p(x_i) = -\sum_{i=1}^{M} p(a_i) \log_2 p(a_i) \qquad (2.14)$$

unde M este numărul de simboluri distincte $a_1, \ldots, a_M$.

Astfel numărul de biţi generat de codarea aritmetică este egal exact cu entropia sursei H(X). Aceasta demonstrează că prin codarea aritmetică se obţine o compresie care este exact aceea prezisă de entropia sursei.

Pentru a reface mesajul, decodorul trebuie să ştie modelul sursei folosite de codor (simbolurile sursei şi plajele de valori asociate) şi un singur număr din cadrul intervalului determinat de codor. Decodarea constă într-o serie de comparaţii ale numărului „i" cu plajele de valori reprezentând simbolurile sursei. Pentru exemplul curent acest număr „i" poate fi 0,0325 (dar şi oricare dintre numerele 0,03248, 0,0326 sau 0,0327). Decodorul

foloseşte „i” pentru a simula acţiunile codorului. Deoarece „i” se află în plaja [0; 0,2) decodorul deduce că primul simbol este A. Decodorul poate deduce că următorul simbol poate îngusta intervalul în unul dintre următoarele moduri:

Simbol	Plajă valori
A	[0; 0,04)
B	[0,04; 0,12)
C	[0,12; 0,14)
D	[0,14; 0,18)
#	[0,18; 0,2)

Tab. 2.19 Tabelul de codare simboluri-plaje de valori (al doilea simbol)

Deoarece „i” se află în intervalul [0; 0,4), decodorul deduce că cel de al doilea simbol este A. Procesul este continuat până ce este recuperat întregul mesaj.

În implementarea codării aritmetice se întâmpină următoarele dificultăţi. Prima este că decodorul trebuie să ştie când să se oprească. Astfel numărul 0 poate reprezenta oricare dintre mesajele A, AA, AAA etc. S-au sugerat două soluţii pentru această problemă. Una este de a transmite dimensiunea mesajului ca o parte a descrierii modelului. Alta este de a folosi un simbol special pentru a indica sfârşitul mesajului. Simbolul # din cadrul exemplului anterior are acest rol. A doua soluţie este preferabilă, din mai multe motive. Primul este că determinarea dimensiunii mesajului este un proces în două etape şi exclude codarea aritmetică ca o parte a codării hibride. Un al doilea motiv este că metodele de codare aritmetică adaptivă sunt uşor de dezvoltat şi o primă trecere pentru a determina dimensiunea mesajului este exclusă ca o parte a unui proces adaptiv on-line[4],[5].

O a doua dificultate a codării aritmetice este caracterul nonincremental al transmiterii şi recepţiei. Din descrierea anterioară a algoritmului rezultă că nu se transmite nimic până la momentul determinării intervalului final. Totuşi o astfel de întârziere nu este necesară. Pe măsură ce intervalul se îngustează, biţii cei mai semnificativ ai celor două capete ale intervalului devin identici. Orice biţi care au devenit identici pot fi transmişi imediat deoarece ei nu vor mai fi afectaţi de o îngustare ulterioară.

O a treia dificultate este legată de precizie. Din descrierea codării aritmetice rezultă că precizia necesară creşte nemărginit pe măsura creşterii lungimii mesajului. O aritmetică în virgulă fixă poate fi folosită atâta timp cât sunt detectate şi tratate depăşirile inferioare sau / şi superioare.

Este evident că metoda de codare aritmetică implementată în mod concret nu conduce chiar la o lungime medie egală cu H. Aceasta se datorează atât folosirii unui terminator cât şi folosirii unei aritmetici în virgulă fixă. S-a

estimat însă că o asemenea creştere a dimensiunii mesajului este de numai 10^{-4} biți /mesaj sursă.

În cadrul exemplului următor este efectuată codarea aritmetică pentru mesajul prezentat în exemplul 2.2 .

■ **EXEMPLUL 2.17**

Simbol	Probabilitate	Probabilitate cumulată	Plajă valori
a	0,05	0,05	[0; 0,05]
b	0,075	0,125	[0,05; 0,125]
c	0,1	0,225	[0,125; 0,225)
d	0,125	0,35	[0,225; 0,35)
e	0,15	0,5	[0,35; 0,5)
f	0,175	0,675	[0,5; 0,675)
g	0,2	0,875	[0,675; 0,875)
spațiu	0,125	1,0	[0,875; 1,0)

Tab. 2.20 Tabelul de codare simboluri-plaje de valori

Se constată că dimensiunea finală a intervalului este $p^2(a)\cdot p^3(b)\cdot p^4(c)\cdot p^5(d)\cdot p^6(e)\cdot p^7(f)\cdot p^8(g)\cdot p^5(spațiu)$. Numărul de biți necesari precizării unei valori în acest interval este $-\log2(1{,}44\cdot10^{-35})=115{,}7$. Se constată că dacă nu se ia în considerare terminatorul, dimensiunea mesajului codat este de 116 biți (cu 1 bit mai puțin decât codarea Huffman !)

2.6 ALGORITMI DE COMPRESIE ENTROPICA BAZATI PE REPETARI DE FORME

Metodele de compresie incluse în această secțiune s-au dezvoltat într-o perioadă mare de timp. Ele vor fi discutate oarecum în ordinea cronologică a apariției, cu mențiunea că, dacă unele au mai degrabă importanță “istorică”, altele se regăsesc, individual sau în asociere cu alte metode, în programe moderne de compresie.

2.6.1 Algoritmul “anularea spațiilor”(blank supression - BS)

Suprimarea caracterelor de tip blanc este una din primele tehnici de compresie folosite. Această tehnică simplă este utilizată în protocolul de transmisie IBM 3780 BISYNC, precum şi, împreună cu alte tehnici de compresie de date orientate pe caracter, pentru reducerea dimensiunii memoriei necesare stocării şi a timpului de transmitere.

Această metodă presupune scanarea unui şir de date pentru depistarea spațiilor care se repetă. În ceea ce priveşte codarea unei asemenea secvențe, caracterele spațiu sunt înlocuite cu un sistem special de perechi de caractere, al cărui format este prezentat în fig. 2.13:

CIC	contor

Fig.2.13. Secvenţă comprimată

unde CIC - caracter indicator de compresie (Compression Indicator Character).

Primul, caracterul indicator de compresie - CIC, este utilizat pentru a indica faptul că s-a practicat suprimarea spaţiilor. Al doilea caracter este utilizat pentru a indica numărul de caractere spaţiu care au fost numărate şi înlocuite prin secvenţa comprimată de două caractere. Când secvenţa de două caractere este transmisă în cadrul şirului de date, dispozitivul de recepţie efectuează o căutare a CIC-ului. După detecţia acestui caracter, receptorul ştie că următorul caracter conţine un contor al numărului de spaţii care au fost suprimate.

■ EXEMPLUL 2.18

Se consideră şirul original de date : GFVA------OLM

Se obţine şirul de date comprimat : GFVA**cic**6OLM în care "**cic**" – reprezintă caracterul indicator de compresie.

Dacă se alocă un octet pentru contorul de caractere, acesta poate înregistra până la 256 de spaţii.

Variante practice pentru alegerea CIC-ului

1) În cazul unui set normal de caractere, se poate utiliza un cod ASCII nefolosit în textul sau fişierul ce urmează a fi comprimat.

2) Dacă trebuie să se utilizeze un set extins de caractere, se foloseşte o secvenţă de patru caractere pentru a coda o succesiune de spaţii consecutive, cum este arătat în fig. 2.14.

SI	CIC	CONTOR	SO

Fig. 2.14. Structura unei secvenţe de spaţii consecutive comprimată

în care:

SI (shift in) - caracter ce introduce o secvenţă de caractere speciale;
SO (shift out) - caracter ce indică ieşirea din secvenţa de caractere speciale;
CONTOR - numărător de spaţii;
CIC - indicator de compresie;

■ EXEMPLUL 2.19

Se consideră şirul de date original: BVT-----QRM. Folosind algoritmul indicat anterior se obţine şirul de date comprimat: BVT**sicic** 5**so**QRM.

Suprimarea caracterelor de tip spaţiu în acest caz este eficientă pentru secvenţe de cel puţin cinci spaţii consecutive. In general trebuie luată în considerare

statistica apariției spațiilor. Astfel, dacă frecvența de apariție este mai mare decât procentajul de reducere a datelor, algoritmul va avea ca rezultat extinderea dimensiunilor fişierului obținut chiar dacă, de fapt, comprimăm toate aparițiile de trei sau mai multe spații. Practica arată că suprimarea secvențelor de 5 sau mai multe spații conduce la o compresie eficientă[10].

3) Altă variantă de alegere a CIC-ului constă în dublarea unui caracter ASCII mai puțin folosit.

4) Folosirea entităților predefinite

Este posibil să se rezerve un grup de caractere din setul de caractere pentru a reprezenta câteva numere predefinite de spații. Astfel, un caracter poate reprezenta cinci spații în timp ce un al doilea caracter poate fi folosit pentru a reprezenta 20 de spații (pentru a sări la începutul unui paragraf într-un document).

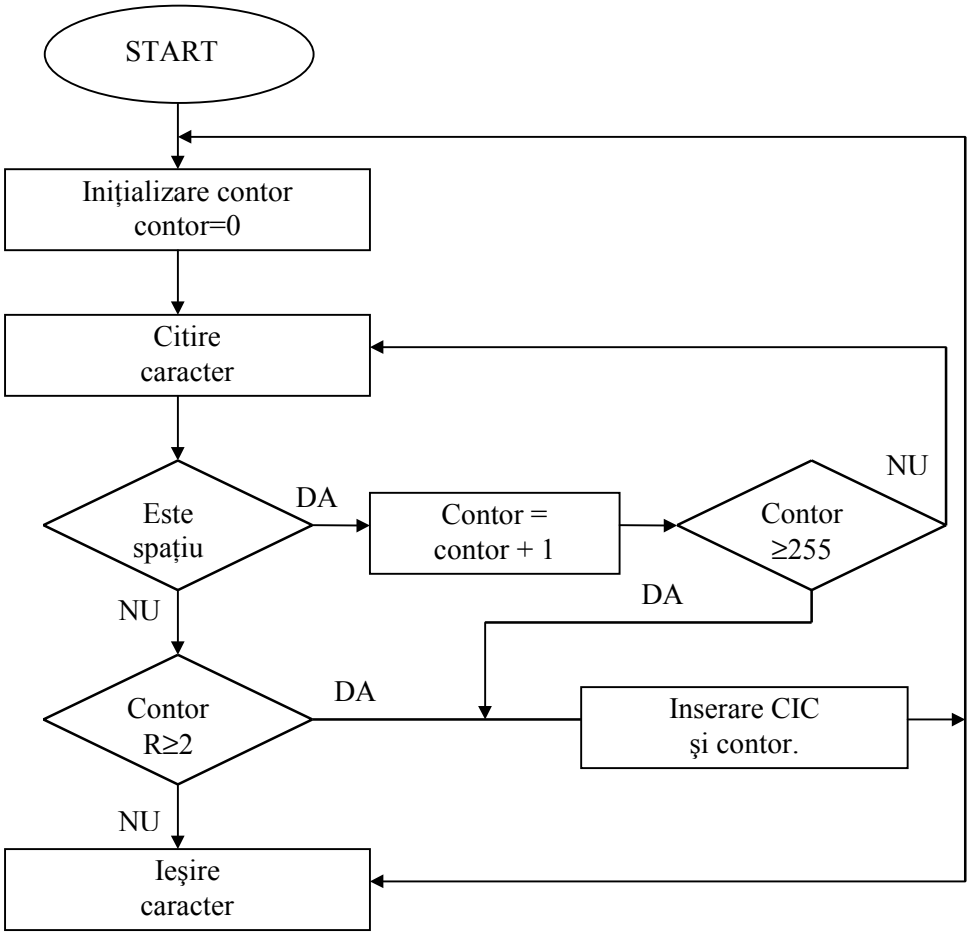


Fig 2.15. Algoritmul de compresie BS

Dacă se foloseşte o entitate predefinită pentru a reprezenta “tab stop positions” se obține o altă variantă a metodei, prin utilizarea caracterului “tab”. Dacă “tab stops” sunt predefinite, nu trebuie decât înlocuite secvențele de spații sau zerouri cu tab, ceea

ce arată că următorul caracter începe într-o anumită coloană din linie şi toate coloanele între ultimul caracter şi locaţia unde începe următorul caracter sunt spaţii[40].

Partea de compresie a algoritmului este uşor de implementat. Se presupune existenţa unui "contor" care este incrementat când este întâlnit un spaţiu în fişierul de intrare. Compresia se efectuează atunci când sunt întâlnite cel puţin trei caractere spaţiu succesive. Deoarece contorul este pe 8 biţi, el nu poate contoriza decât 255 de spaţii succesive. Când se atinge această valoare, cele 255 de spaţii sunt comprimate iar contorul este adus la valoarea 0. Când s-a citit un caracter care nu este spaţiu, se testează contorul "contor", şi în funcţie de valoarea lui se efectuează compresia sau nu şi apoi se transmite noul caracter citit la ieşire. La decompresie, când s-a întâlnit un CIC se citeşte următorul caracter care indică numărul de spaţii suprimate. În funcţie de acest număr se adaugă spaţii în fişierul de ieşire. În figura 2.15 se prezintă organigrama acestui algoritm, urmată de programele de compresie-decompresie în PASCAL.

Compresia

```
citire caracter "ch";
IF ch<>caracterul EOF THEN
  WHILE ( nu este sfărsitul fisierului de intrare )
    BEGIN
        IF( "contor" < 255 si "ch"=spatiu ) THEN  se incrementează "contor" ;
                  ELSE IF ("ch"=spatiu si "contor"=255) THEN
                    BEGIN
                      se transmite la iesire comprimatul celor 255 de spatii;
                       "contor"=0 ;
                  END ; {end if}
{ "ch" nu este spatiu :  }
 IF ("contor"=0) THEN  "ch" este trimis la iesire;
ELSE     IF ("contor">2)
                    BEGIN
                          se trimite CIC la iesire;
                          se trimite "contor" la iesire;
                          se trimite "ch" la iesire;
                          "contor"=0;
                    END
          ELSE
BEGIN
                          CASE "contor" OF
                                "contor"=1 :      se trimite un spatiu la iesire;
                                                   se trimite "ch" la iesire;
                                                    "contor"=0;

                                "contor"=2 :      se trimit doua spati la iesire ;
                                                   se trimite "ch" la iesire;
```

“contor”=0;
END
END { end while }
{ daca la sfărsitul fisierului se gaseste o secvența de spații }
IF (“contor”<> 0) THEN se trimite formatul de comprimare la iesire;
se închid fisierele;

Decompresia

citire caracter “ch”;
IF (“ch” este CIC) THEN
BEGIN
se decomprima secvența;
END
ELSE
Se trimite caracterul “ch” la iesire;

2.6.2 Algoritmul bazat pe “codarea grupurilor de caractere consecutive “(RUN LENGTH ENCODING - RLE)

Aceast algoritm comprimă orice secvență de caractere ce apar consecutiv şi presupune folosirea unui caracter special care să indice folosirea acestei metode. Acest caracter este urmat de unul dintre caracterele care se repetă. Dacă se folosesc coduri ASCII sau EBCDIC, caracterul special trebuie să difere de cele din şir. Pentru fiecare dintre aceste coduri există numeroase caractere neatribuite cu reprezentări unice care pot fi folosite. Pentru situațiile în care nu există un caracter nefolosit se alege un caracter care nu poate să apară de două ori succesiv pentru a indica compresia.

Algoritmul de codare

Metoda este convenabilă dacă acel caracter se repetă de cel puțin 4 ori. Ea foloseşte codarea din figura 2.16

C_C	X	S_C

Fig. 2.16 Codarea grupurilor de caractere consecutive

C_C = caracterul special care indică existența compresiei
X = caracterul care se repetă
S_C = contor care indică de câte ori se repetă caracterul

■ **EXEMPLUL 2.20**

a. \$****5572 → \$**C**$_C$ *45572

b. Abba------Queen → Abba**C**$_C$_**4**Queen

Principalii paşi ai algoritmului RLE, prezentaţi în fig. 2.18 sunt :
(1) , (2) se setează cu zero contorul pentru caractere şi cel pentru numărul de repetări ale unui caracter
(3) se primeşte un caracter
(4) se incrementează contorul de caractere
(5) contorul se compară cu 1
(6) dacă este primul caracter din şir, se stochează într-un buffer
(7) dacă nu este primul caracter se compară cu cel anterior stocat în buffer
(8) dacă sunt identice, se incrementeză contorul pentru numărul de repetări cu 1. Dacă caracterul obţinut nu este identic cu cel anterior se compară contorul de repetări cu 4. Dacă este mai mare decât 4 se activează formatul de compresie.

Numărul de repetări ale unui caracter este limitat în funcţie de numărul de biţi folosiţi pentru reprezentarea acelui caracter. Pentru un cod cu 8 biţi pe caracter maximul se situează între 255 şi 260 de caractere identice consecutive. Valoarea exactă va depinde de modul de folosire a contorului.

În cele mai multe situaţii valoarea contorului caracterului curent este folosită ca număr al caracterelor repetate. În acest mod valoarea maximă a contorului va fi 2^8-1 = 255. Dar, se poate folosi un contor cu toţi biţii 0 pentru a indica 4 caractere repetate, în timp ce un contor cu toţi biţii 1 va indica 260 de caractere repetate.

Cea mai răspândită implementare a metodei RLE prezintă o mică variaţie faţă de tehnica descrisă anterior. Un contor pentru repetări este inserat într-un şir de date pentru a reprezenta numărul de octeţi care s-au repetat şi care urmează în primele 3 secvenţe.

Această repetare a caracterului serveşte ca indicator de început de compresie RLE şi elimină folosirea unui caracter special.

C	**X**	**X**	**X**

Fig. 2.17 Codarea RLE îmbunătăţită

X = caracterul care se repetă;
C = contor de repetiţie (maxim 250)

Eficienţa metodei run-length depinde de numărul de caractere repetate apărute în şirul de date, media lungimii şirului de caractere repetate şi de tehnica folosită pentru a executa compresia. Există şi o variantă mai nouă, cu antet de 16 biţi (RLE-16), pentru care prezentăm schema de organizare:

File format:
13 bytes : denumire fişier original, urmat de:
[16 bit antet + date] [16 bit antet + date] [16 bit antet + date] etc.
.header:

[lo byte][hi byte] ==> se transforma în 16 bit ==>
bit 15: 1 dacă următorul byte este o secvență de caractere consecutive
bit 14 - 0: lungimea secvenței (max 32767, min 4)
data: 1 byte : caracterul care se repetă

2.6.3. Metoda înlocuirii formelor repetitive (Pattern Substitution - PS)

Este metoda cea mai generală dintre metodele care au ca idee substituția unor forme repetitive, nefiind necesar să avem caractere identice în secvență.. Compresia este cu atât mai bună cu cât modelul (forma care se repetă) este mai lung şi mai frecvent, fiecare model fiind înlocuit de un singur caracter special. Apar însă două probleme: să existe un număr mare de caractere speciale disponibile, şi să se găsească cea mai bună corespondență a setului de caractere speciale cu setul de modele. Prima problemă este în general rezolvată prin utilizarea setului extins de caractere ASCII (fişierele text nu conțin, în general, caractere cu codul ASCII mai mare de 127), a doua problemă îşi găseşte soluțiile în următoarele metode:

- Metoda PS cu tabel de substituire date (PS-TS). Tabelul de modele există la începutul compresiei ceea ce presupune că este o metodă text cu cunoaştere apriorică şi exactă. Cea mai cunoscută tehnică de compresie de tip PS-TS este codarea diatomică.
- Metoda PS adaptivă (PSA). Este o metodă a cărei particularitate constă în modul în care este gestionată aplicarea setului de caractere speciale pe modele. De fapt este o asociere între o metodă de compresie statică, de exemplu Huffman, prin care se determină la o primă trecere formele care apar cel mai frecvent. După parcurgerea integrală a textului (prima trecere) sunt găsite cele mai lungi şi mai frecvente modele şi caracterele speciale disponibile, care sunt asociate modelelor astfel găsite. Flexibilitatea acestei metode este mărită, dar apare o întârziere datorată extragerii modelelor şi necesității de transmitere a tabelului de substituție o dată cu mesajul comprimat. Această metodă se recomandă în special la comprimarea fişierelor cu programe sursă, în care apar frecvent forme repetitive (modele) de tip instrucțiuni: BEGIN, IF, THEN, ELSE, INTEGER, READ etc.; sunt exemple de forme care se pretează la substituție cu caractere speciale, ceea ce duce la o compresie puternică.

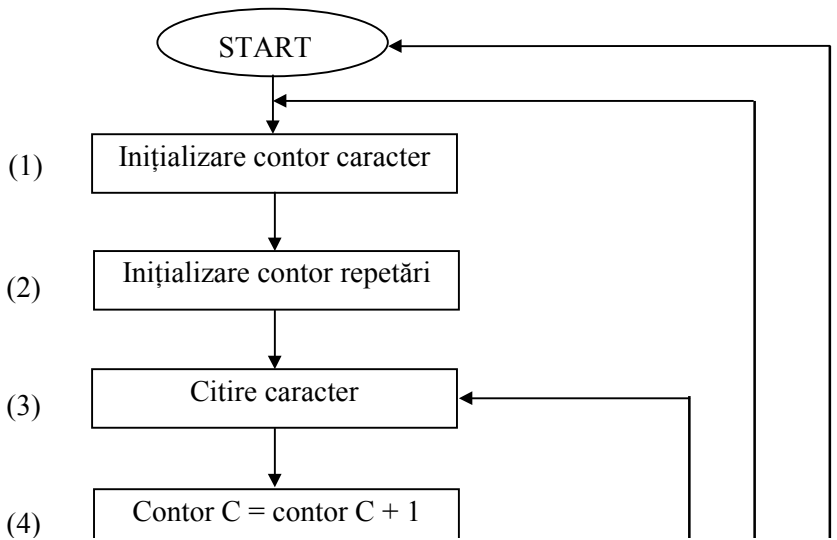


Fig. 2.18 Paşii algoritmului RLE

Pentru urmărirea modului în care operează o metodă PS, detaliem algoritmul de compresie denumit codare diatomică.

Codarea diatomică (Diatomic Encoding – DE)

Trăsătura principală a acestei metode este faptul că o pereche de caractere este înlocuită printr-un caracter special. Structura de biți a caracterului special constituie codarea perechii de caractere, deci rata compresiei poate tinde spre 2:1.Cum însă numărul de caractere speciale care pot fi folosite pentru a reprezenta diferite tipuri de caractere este limitat, în mod practic rata de 50% nu se poate obține. Odată ce se ştie frecvența de apariție a perechilor de caractere, atunci cele mai des întâlnite perechi pot fi selectate candidate pentru codare. Numărul efectiv de perechi selectate va depinde de numărul de caractere speciale disponibile pentru a reprezenta

caracterele cele mai frecvente. Considerațiile de pînă acum au plecat de la ipoteza că şirul de date disponibil la intrare este continuu. În realitate, buffer-ele de intrare şi de ieşire au dimensiune finită. Deoarece buffer-ul de ieşire va avea întotdeauna cel mult dimensiunea buffer-ului de intrare, se va putea atribui un pointer care va fi incrementat de către buffer-ul de intrare. Dacă se atinge sfârşitul acestui buffer, conținutul buffer-ului de ieşire va fi transmis, în timp ce buffer-ul de intrare va fi reumplut cu date necomprimate.

Frecvența de aparitie a perechilor de caractere.

O problemă majoră de implementare a metodei este alegerea perechilor care vor fi reprezentate prin caractere speciale. Pentru a face codificare diatomică şi a obține o rată de compresie importantă se cere atribuirea unor caractere speciale care să reprezinte cele mai frecvente perechi de caractere care se presupune că vor fi întâlnite în şirul de date. Aceasta înseamnă că trebuie avută o informație apriorică referitoare la tipul de date care vor fi prelucrate.

Implementare hardware.

Tehnica prezentată în continuare se inspiră din cea care a fost implementată de Infotron System în combinație cu alte tehnici de compresie pentru multiplexoare statistice.

Într-un multiplexor convențional divizor de timp, datele de pe fiecare canal de intrare sunt atribuite unui interval pe linia de ieşire de mare viteză a multiplexorului, indiferent dacă este sau nu folosită toată banda. Dacă compresia este implementată pe partea liniei de viteză mică, eficiența fiecărei legături comprimate de pe această parte va creşte atâta timp cât fiecărei legături îi este rezervat un interval fix pe partea de viteză mare.

La un multiplexor statistic, lărgimea benzii unui anumit canal, pe partea de mare viteză, este folosită doar atunci când canalul transmite date sau semnale de control. De aceea, compresia a una sau mai multe linii de mică viteză permite multiplexorului statistic să utilizeze mai puțin din lățimea benzii pe linia de mare viteză.

În tehnica Infotron, compresia cu multiplexor statistic se realizează prin utilizarea codării grupărilor de caractere consecutive, codării bazate pe înlocuirea formelor repetitive şi codării "half-byte" (care impachetează şiruri de cifre reprezentate inițial prin caractere ASCII la nivel de două cifre pe octet (byte) reprezentate doar prin cei patru biți semnificativi). Adaptorul de canal Infotron care efectuează compresia operează doar asupra datelor codate ASCII asincron. Pentru a obține un număr suficient de indicatori de compresie, bitul de paritate într-un cod ASCII pe 8 biți nu este folosit în transmisie. De aici rezultă 128 de caractere care pot fi utilizate pentru a reprezenta şi indica informații comprimate. Codurile sunt atribuite pentru a reprezenta grupuri de 2 până la 7 spații consecutive pentru variate scheme de coduri de compresie "multi-spaces". Aceste coduri sunt foarte eficiente atunci când datele transmise au fost formate în coloane separate de grupuri de spații sau pentru informații text care conțin paragrafe unde se justifică folosirea spațiilor[50].

Pentru a reprezenta grupările de caractere consecutive, au fost atribuite 16 coduri pentru a reprezenta grupuri de 3 până la 18 caractere identice consecutive. Un

astfel de cod este urmat de caracterul care se va repeta, în mod similar cu codarea RLE şi rezultă un cod pe 2 octeți. Pentru reprezentarea perechilor de caractere au fost atribuite 48 de coduri. Perechile de caractere utilizate de Infotron sunt reprezentate în figura 2.19.

S_	_T	D_	SE	HE
T_	_A	_I	TE	AN
E_	_N	_	ER	TI
R_	_O	AT	RE	ON
ED_	_IN	ES	TH	CRLF

Fig 2.19 Perechile de caractere folosite de INFOTRON

Alte 16 coduri au fost atribuite pentru a specifica situatii când 4 până la 19 caractere sunt în format half-byte. Caracterele sunt reprezentate de coduri de 4 biti.

Algoritmul de compresie diatomică este ilustrat în figura 2.20.

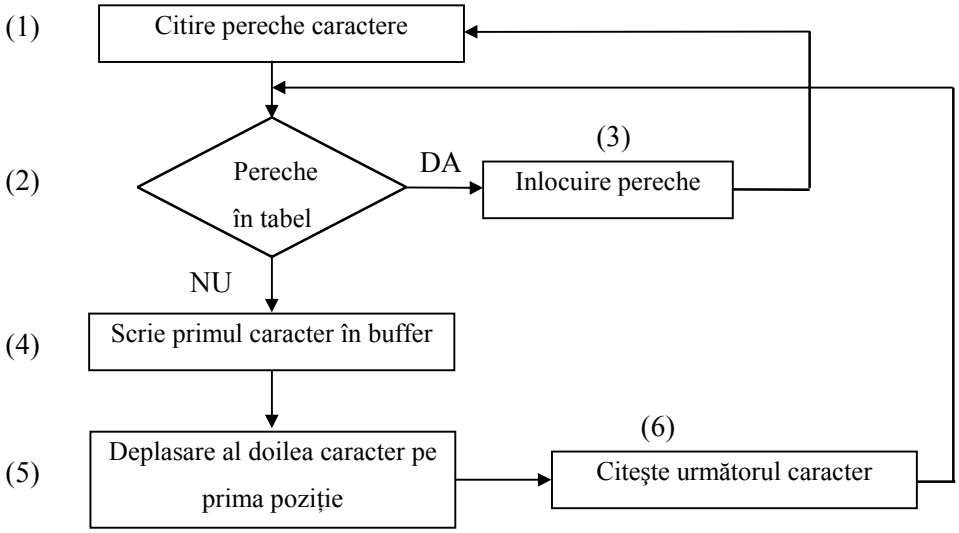


Fig.2.20 Algoritmul de compresie diatomică

2.6.4. Metode de compresie bazate pe algoritmi Lempel-Ziv

Generalități

În 1975, A. Lempel a dezvoltat un algoritm de compresie care a fost publicat în "IEEE Transactions on Information Theory" în mai 1977; acest algoritm caută forme repetitive într-un şir de caractere direct sub forma de (sub)şir de caractere. Un an mai

târziu, J. Ziv aduce o îmbunătățire substanțială în structurarea datelor pe baza cărora se face compresia. Algoritmul de compresie de şiruri, care este cunoscut de atunci sub numele de tehnica Lempel-Ziv (LZ), a cunoscut repetate şi substanțiale îmbunătățiri, ceea ce confirmă în primul rând ingeniozitatea concepției. În principiu, un algoritm LZ constă într-un set de reguli pentru împărțirea şirurilor de simboluri dintr-un alfabet finit în subşiruri sau cuvinte ale căror lungimi să nu depăşească un L_s predefinit şi o schemă de compresie care converteşte subşirurile rezultate în cuvinte de cod de lungime fixată L_c.

Relația dintre lungimea cuvântului de cod şi lungimea subşirului este

$$L_c = 1 + [\log(n - L_s)] + [\log L_s] \tag{2.15}$$

unde:

n - lungimea buffer-ului în care sunt stocate ultimele simboluri

L_c - lungimea cuvântului de cod

L_s - lungimea maximă a unui subşir din textul sursă care poate fi codat

În conformitate cu algoritmul Lempel-Ziv, un şir de simboluri S este împărțit în subşiruri succesive $S = S_1 S_2...S_n$, şi fiecărui S_i îi este asociat un cuvânt de cod C_i.

La începerea compresiei se presupune că, la ieşire, şirul sursă S, este precedat de un şir Z care conține n-L_s zero-uri. Acest şir este stocat în buffer ca un şir $B_1 = Z / S(1, L_s)$. Apoi, primele l_1 simboluri sunt scoase din buffer în timp ce alte l_2 simboluri intră în buffer pentru a obține un al doilea şir B_2 . Această operație este urmată de o analiză a şirului pentru depistarea secvențelor redundante.

Recomandarea V.42 bis

Algoritmul Lempel-Ziv a fost rafinat de către firmele IBM, British Telecom, Bell Laboratories, Unisys şi alte companii sub recomandarea CCITT V.42 bis pentru implementarea compresiei de date în modemuri. Conform recomandării V.42 bis, un şir cu o lungime medie între 2 şi 4 caractere este înlocuit printr-un cuvânt de cod bazat pe construcția unui dicționar care va conține 512 sau mai multe şiruri text şi cuvintele de cod asociate.

Procesul de compresie descris prin recomandarea V.42 bis include construcția unui dicționar având ca intrări perechi de date necomprimate şi rezultând cuvinte de cod care substituie şiruri de date. Bazat pe acelaşi dicționar, la recepție, cuvintele de cod sunt decodate.

Dicționarul V.42 bis conține, inițial, setul primar de caractere după care se extinde, formându-se noi şiruri prin adăugarea unui singur caracter la şirurile existente. Dicționarul este dinamic, noile şiruri fiind adăugate şi vechile şiruri eliminate.

Inițial, toate şirurile formate dintr-un singur caracter sunt predefinite într-un tabel. Oricum, aceasta înseamnă că sunt definite 256 şiruri de un caracter. Ca exemplu

se vor lua majusculele literelor de la A la Z. Dicţionarul va fi iniţiat ca o secvenţă de noduri rădăcină, după cum se observă în figura 2.21.

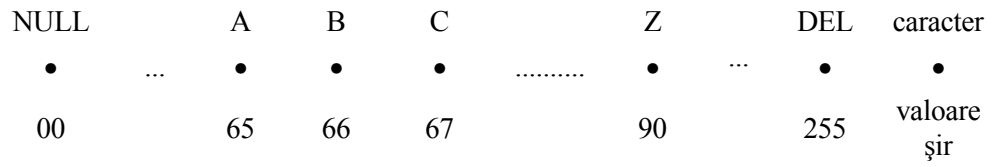


Fig 2.21 Dicţionarul iniţiat ca o secvenţă de noduri rădacină

Cuvântul de cod pentru fiecare şir este iniţial codul ASCII al fiecărui caracter. Astfel, A va avea numărul de şir 65, B va avea atribuită valoarea 66, şi aşa mai departe. Pe măsură ce datele sunt codate este verificat mereu conţinutul dicţionarului. Dacă şirul întâlnit nu este în dicţionar, acesta este adăugat dicţionarului şi îi este atribuit un cuvânt de cod.

Pentru a ilustra aceasta, presupunem că alfabetul constând din literele de la A la Z trebuie transmis secvenţial, literă cu literă. Primul nou şir întâlnit va fi AB, dar A era definit anterior iniţializării dicţionarului. Astfel, acest nou şir va fi stocat în dicţionar şi i se va atribui cuvântul de cod 257.

În acelaşi timp în care AB a intrat în dicţionarul codorului, cuvântul de cod pentru B este transmis la receptor. Acest fapt dă posibilitatea receptorului de a-şi construi un dicţionar imagine care este identic cu dicţionarul codorului. Apoi, ori de câte ori AB este codat, acest şir va fi transmis ca şi un cuvânt de cod 257, care la recepţie va fi recunoscut şi decomprimat corect.

Următorul nou şir care va fi identificat va fi BC. Acum, dicţionarul este actualizat prin adăugarea lui C la nodul B. În acelaşi timp, codorul transmite cuvântul de cod pentru C, care este 67, astfel încât receptorul va şti că al treilea caracter de la intrare a fost C. Atât codorul cât şi decodorul incrementează valoarea cuvântului de cod de la 1 la 258 şi atribuie această valoare şirului BC în dicţionar. Astfel, când şirul BC va fi reîntâlnit, el poate fi comprimat şi transmis ca 258.

Pe măsură ce datele sunt procesate, setul iniţial de caractere stocat în dicţionar se va extinde într-o structură de tip arbore. Dacă setul iniţial de caractere este considerat ca "noduri-radăcină", expansiunea dicţionarului va avea ca rezultat generarea de "noduri-frunză". În figura 2.22 sunt ilustrate frunzele formate de la nodurile A şi B (nodul T a fost adăugat în scop ilustrativ). Arborele din figura 2.19 reprezintă şirurile A, AB, B, BC, ... T, TH, THE, TO. Se observă că fiecare frunză depinde de nivelul superior (E depinde de H) sau de nodul rădăcină.

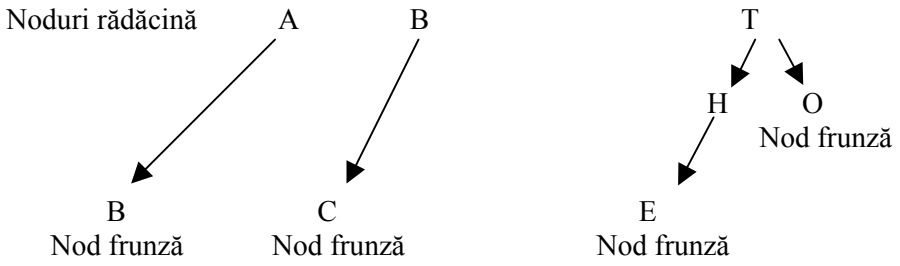


Fig. 2.22. Un exemplu de arbore LZ

În procesul de codare noile şiruri sunt construite din şirul curent, definit în dicţionar, prin adăugarea unui caracter. Astfel, o a doua secvenţă a alfabetului ca dată originală ar rezulta în adăugarea şirului ABC la dicţionar, câtă vreme AB este cunoscut anterior.

Când şirul ABC este adăugat prin formarea unei noi ramuri de la "nodul-frunză" B, cuvântul de cod pentru AB (257) este transmis urmat de cuvântul de cod pentru C, care va da posibilitatea receptorului să actualizeze dicţionarul. În acelaşi timp, următorul cuvânt de cod disponibil este atribuit şirului ABC atât la codor cât şi la decodor.

Se observă că această strategie de codare a datelor în şiruri poate genera un număr infinit de şiruri şi cuvinte de cod corespunzătoare. Pentru a preveni aceasta, CCITT V.42 bis specifică o limită maximă a lungimii şirului care este permisă şi care poate fi negociată între codor şi decodor.

Lungimea maximă permisă a şirului poate varia de la 6 la 250. În mod similar, V.42 bis specifică o valoare de 512 cuvinte de cod, care reprezintă numărul minim de cuvinte de cod sau de şiruri unic definite ce se pot produce în acelaşi moment. Deşi în această recomandare nu este specificată o valoare maximă, se doreşte creşterea numărului de cuvinte de cod.

Conform recomandării V.42 bis, fiecare dispozitiv de compresie de date păstrează două dicţionare (unul la codor şi altul la decodor).

Pentru funcţia de comparare şi actualizare a şirurilor, V.42 bis defineşte o procedură de ştergere a şirurilor cu frecvenţa de apariţie redusă. Această procedură este necesară atunci când ultima poziţie disponibilă din dicţionar a fost ocupată. În acest timp un contor (C_1) ce indică cuvântul de cod care ar putea fi atribuit următoarei intrări libere în dicţionar este comparat cu numărul total de cuvinte de cod N_2. Dacă depăşeşte N_2-1, atunci C_1 este setat la numărul index al primei intrări în dicţionar, utilizat să înregistreze un şir indicat de N_5. Dacă nodul identificat prin cuvântul de cod cu valoarea C_1 este utilizat şi nu este nod frunză, C_1 este incrementat şi apoi comparat cu N_2-1. Dacă nodul identificat prin cuvântul de cod cu valoarea C_1 este un nod frunză, acesta va fi detaşat de părintele său, eliberându-se o intrare pentru reutilizare.

Un exemplu de algoritm LEMPEL-ZIV este algoritmul WALSH, ai cărui paşi sunt prezentaţi în continuare:

Algoritmul de compresie

- Se porneşte cu un şir nul.
- Se citeşte un caracter, care este ataşat şirului.
- Dacă şirul se găseste în tabel, se continuă citirea şi ataşarea caracterelor la şir până când şirul nou format nu se mai găseste în tabel.
- Acest nou şir este adăugat în tabel.
- Se scrie codul pentru ultimul şir cunoscut care este identic cu ieşirea.
- Se foloseşte ultimul caracter ca bază pentru următorul şir şi se continuă citirea.

(De câte ori decodorul găseşte un şir nou, îl adaugă tabelului.)

Din păcate acest algoritm pretinde foarte multă memorie. Fiecare şir adăugat are lungime variabilă, ceea ce duce la o mare problemă de stocare. Există însă o soluție simplă care să evite această dificultate. După cum s-a observat, fiecare şir nou este, de fapt, un şir vechi plus un caracter. Deci practic se memorează codul şirului vechi plus caracterul adăugat.

■ **EXEMPLUL 2.21**

Compresorul "economiseşte" un cod prin transmiterea lui 258 în loc de "is" sub forma reprezentării literale.

Şirurile sunt stocate în tabelul LZW sub forma unor combinații de tip cod-caracter şi nu sub forma unor şiruri literale.

Intrare	Tabelul de compresie	Şir comprimat	Tabelul extinsă
T	-	-	-
h	256 → T + h	T	-
i	257 → h + i	h	256 → T + h
s	258 → i + s	i	257 → h + i
spațiu	259 → S + spațiu	s	258 → i + s
i	260 → spațiu + i	spațiu	259 → s+spațiu
s	-	-	-
spațiu	261 → 258 + spațiu	258	260 → spațiu+i
a	262→ spațiu + a	spațiu	261→ 258+spațiu

Tab. 2.21 Tabelul de compresie LZW

Compresorul citeşte un T inițial şi îl adaugă şirului nul. Şirul T este un şir literal, deci este în tabel. Compresorul citeşte următorul caracter, h **,** căutând subşirul Th în tabelul de subşiruri unde nu îl va găsi. Adaugă apoi subşirul Th tabelului pe următoarea poziție disponibilă şi trimite la ieşire ultimul şir cunoscut, T. Se continuă apoi citirea caracterelor şi adăugarea (sub)şirurilor până când secvența de intrare este epuizată. Această secvență de intrare simplă ilustrează exemplul de compresie în care codul 258 este trimis la ieşire în loc de şirul "is".

Din nefericire, acest algoritm simplu este mare consumator de memorie. În fiecare moment, compresorul gaseşte un nou (sub)şir pe care îl adaugă la tabel. Fiecare

(sub)şir care este adăugat are o lungime variabilă care poate duce la o acumulare uriaşă de informaţie. Există însă o cale de a evita această proliferare a tabelului. Fiecare şir nou este considerat ca fiind un şir vechi plus un nou caracter. În loc de stocarea şirurilor în mod explicit, acestea se pot stoca sub forma unui cod şi a unei combinaţii de caractere.

Exemplul anterior ilustrează această metodă de stocare. Codul 261, de exemplu, este stocat sub forma 258 + (spaţiu) şi nu sub forma *is*(spaţiu).

Algoritmul de decompresie

Algoritmul de decompresie porneşte cu un tabel (întocmai ca şi la compresie) ce conţine numai date literale definite. Se efectuează citirea primului caracter de la intrare. Se trimite acest caracter la ieşire sau se păstrează pentru a forma baza pentru următorul şir. Se generează un şir şi se efectează o actualizare a tabelului de subşiruri pentru fiecare cod ce urmează primului cod citit de programul de decompresie. Decompresorul utilizează tabelul pentru a traduce valoarea codului într-un şir de ieşire. Pentru şiruri neliterale, acesta urmareşte, prin combinaţiile cod-caracter din tabel, plasarea caracterelor într-o stivă. Pentru al doilea cod, decompresorul adaugă un cod obţinut pe baza primului caracter şi a primului caracter din şirul descris prin cel de-al doilea cod. Dupa aceea, pentru fiecare cod, decompresorul adaugă ultimul cod tradus plus primul caracter din şirul curent, la tabelul. Tabelul rezultată este un duplicat al tabelului de compresie şi este un tabel care se schimbă cu fiecare cod primit. Exista situaţia în care un anumit tip de şir poate genera la ieşirea compresorului un cod, înainte ca decompresorul să-l aibă în tabelul sa. Această situaţie este întâlnită atunci când apar şiruri de forma: XandXandX, iar şirul Xand se găseşte deja în tabel. În acest caz, compresorul va trimite codul pentru Xand (deoarece acesta cunoaşte deja şirul) şi apoi adaugă XandX la tabel. Apoi va începe cu X-ul din mijloc, găseşte următorul grup de caractere pe care îl ştie ca XandX şi va trimite codul pentru XandX înainte ca decodorul să ştie ce reprezintă. Acest caz special se poate rezolva prin tratarea separată în cadrul programului de decompresie. Astfel dacă decompresorul primeşte un cod pe care nu-l recunoaşte, îşi dă sema de faptul că a întalnit un caz singular. În exemplul de mai sus, decompresorul primeşte codul pentru Xand şi apoi un cod necunoscut. Acesta va transmite la ieşire ultimul cod tradus (înca o dată Xand) şi apoi primul caracter din acel cod, X. După aceasta, adaugă o combinaţie de aceste caractere (XandX) în tabel.

Metoda de compresie Lempel - Ziv îmbunătăţită

Există două căi de îmbunătăţire a algoritmului de compresie Lempel - Ziv**:**

- utilizarea unor coduri cu lungime variabilă
- reactualizarea tabelului

Majoritatea soluţiilor LZ moderne combină ambele variante de mai sus.

Pentru codurile de lungime fixă trebuie să se decidă de la început câţi biţi să se folosească pentru codificarea datelor. Dacă se foloseşte un număr mic de biţi, tabelul se va umple repede şi compresia se termină rapid. Dacă se foloseşte un număr mare de biţi, creşterea înregistrată pentru fiecare cod care nu este comprimat corect este enormă. Pentru fiecare cod după primul pe care decodorul îl citeşte se generează

un şir şi se actualizează tabelul. Decodorul mai întâi se foloseşte de tabel pentru a traduce valoarea codului într-un şir. Pentru codurile nonliterale se merge înapoi de la combinația cod/caracter a tabelului, împingând caracterele într-o stivă pe măsură ce algoritmul avansează. Când decodorul ajunge la un cod literal acesta este scos din stivă pentru a obține şirul de ieşire. În plus, fiecare cod după primul cauzează o reactualizare a tabelului. Pentru al doilea cod decodorul adaugă un cod obținut din primul cod plus primul caracter din şir descris de al doilea cod. Pentru fiecare cod decodorul adaugă ultimul cod tradus plus primul caracter din şirul curent. Tabelul rezultat este o copie exactă reactualizată a tabelului de compresie.

Majoritatea versiunilor îmbunătățite se regasesc asociate cu numele celor care le-au produs. Dintre numele mai cunoscute îi citățm pe: Kurt Haenen, Haruhiko Okumura, Haruyasu Yoshizaki, Stefan Westner, Danny Heijl. In toate aceste variante procedura LZ este asociată cu o compresie Huffman adaptivă pentru codarea lungimilor subşirurilor. Una din aceste variante, cunoscută sub denumirea LZRW1/KH sau numai LZH a fost elaborată de Kurt Haenen şi este inclusă în biblioteca mediului de dezvoltare de programe DELPHI. Rutina operează cu blocuri de lungime 32 kb şi este prezentată în continuare.

```
{$R-}
UNIT LZRW1KH;

INTERFACE

uses SysUtils;

{$IFDEF WIN32}
type Int16 = SmallInt;
{$ELSE}
type Int16 = Integer;
{$ENDIF}

CONST
   BufferMaxSize  = 32768;
   BufferMax      = BufferMaxSize-1;
   FLAG_Copied    = $80;
   FLAG_Compress  = $40;

TYPE
   BufferIndex    = 0..BufferMax + 15;
   BufferSize     = 0..BufferMaxSize;
     { extra bytes needed here if compression fails      *dh *}
   BufferArray    = ARRAY [BufferIndex] OF BYTE;
   BufferPtr      = ^BufferArray;
```

```
  ELzrw1KHCompressor = Class(Exception);

FUNCTION  Compression   (   Source,Dest   : BufferPtr;
                    SourceSize    : BufferSize  )   : BufferSize;

FUNCTION  Decompression (   Source,Dest   : BufferPtr;
                    SourceSize    : BufferSize  )   : BufferSize;

IMPLEMENTATION

type
 HashTable     = ARRAY [0..4095] OF Int16;
 HashTabPtr    = ^Hashtable;

VAR
 Hash                : HashTabPtr;

                    { check if this string has already been seen }
                    { in the current 4 KB window }
FUNCTION  GetMatch      (   Source       : BufferPtr;
                    X            : BufferIndex;
                    SourceSize    : BufferSize;
                    Hash         : HashTabPtr;
                  VAR Size         : WORD;
                  VAR Pos          : BufferIndex  )   : BOOLEAN;
VAR
 HashValue     : WORD;
 TmpHash       : Int16;
BEGIN
 HashValue := (40543*((((Source^[X] SHL 4) XOR Source^[X+1]) SHL 4) XOR
                        Source^[X+2]) SHR 4) AND $0FFF;
 Result := FALSE;
 TmpHash := Hash^[HashValue];
 IF (TmpHash <> -1) and (X - TmpHash < 4096) THEN BEGIN
  Pos := TmpHash;
  Size := 0;
  WHILE ((Size < 18) AND (Source^[X+Size] = Source^[Pos+Size])
              AND (X+Size < SourceSize)) DO begin
   INC(Size);
  end;
  Result := (Size >= 3)
 END;
 Hash^[HashValue] := X
END;
```

```
                        { compress a buffer of max. 32 KB }
FUNCTION  Compression(Source, Dest : BufferPtr;
                      SourceSize   : BufferSize) :BufferSize;
VAR
 Bit,Command,Size       : WORD;
 Key                    : Word;
 X,Y,Z,Pos              : BufferIndex;
BEGIN
 FillChar(Hash^,SizeOf(Hashtable), $FF);
 Dest^[0] := FLAG_Compress;
 X := 0;
 Y := 3;
 Z := 1;
 Bit := 0;
 Command := 0;
 WHILE (X < SourceSize) AND (Y <= SourceSize) DO BEGIN
  IF (Bit > 15) THEN BEGIN
   Dest^[Z] := HI(Command);
   Dest^[Z+1] := LO(Command);
   Z := Y;
   Bit := 0;
   INC(Y,2)
  END;
  Size := 1;
  WHILE ((Source^[X] = Source^[X+Size]) AND (Size < $FFF)
                AND (X+Size < SourceSize)) DO begin
        INC(Size);
  end;
  IF (Size >= 16) THEN BEGIN
   Dest^[Y] := 0;
   Dest^[Y+1] := HI(Size-16);
   Dest^[Y+2] := LO(Size-16);
   Dest^[Y+3] := Source^[X];
   INC(Y,4);
   INC(X,Size);
   Command := (Command SHL 1) + 1;
  END
  ELSE begin { not size >= 16 }
   IF (GetMatch(Source,X,SourceSize,Hash,Size,Pos)) THEN BEGIN
    Key := ((X-Pos) SHL 4) + (Size-3);
    Dest^[Y] := HI(Key);
    Dest^[Y+1] := LO(Key);
    INC(Y,2);
```

```
      INC(X,Size);
      Command := (Command SHL 1) + 1
     END
     ELSE BEGIN
      Dest^[Y] := Source^[X];
      INC(Y);
      INC(X);
      Command := Command SHL 1
     END;
   end; { size <= 16 }
   INC(Bit);
  END; { while x < sourcesize ... }
  Command := Command SHL (16-Bit);
  Dest^[Z] := HI(Command);
  Dest^[Z+1] := LO(Command);
  IF (Y > SourceSize) THEN BEGIN
   MOVE(Source^[0],Dest^[1],SourceSize);
   Dest^[0] := FLAG_Copied;
   Y := SUCC(SourceSize)
  END;
  Result := Y
END;

                          { decompress a buffer of max 32 KB }
FUNCTION  Decompression(Source,Dest    : BufferPtr;
                  SourceSize     : BufferSize) : BufferSize;
VAR
 X,Y,Pos               : BufferIndex;
 Command,Size,K        : WORD;
 Bit                   : BYTE;
 SaveY                 : BufferIndex; { * dh * unsafe for-loop variable Y }

BEGIN
 IF (Source^[0] = FLAG_Copied) THEN  begin
  FOR Y := 1 TO PRED(SourceSize) DO begin
   Dest^[PRED(Y)] := Source^[Y];
   SaveY := Y;
  end;
  Y := SaveY;
 end
 ELSE BEGIN
  Y := 0;
  X := 3;
  Command := (Source^[1] SHL 8) + Source^[2];
```

```
  Bit := 16;
  WHILE (X < SourceSize) DO BEGIN
   IF (Bit = 0) THEN BEGIN
    Command := (Source^[X] SHL 8) + Source^[X+1];
    Bit := 16;
    INC(X,2)
   END;
   IF ((Command AND $8000) = 0) THEN BEGIN
      Dest^[Y] := Source^[X];
      INC(X);
      INC(Y)
   END
   ELSE BEGIN  { command and $8000 }
    Pos := ((Source^[X] SHL 4)
         +(Source^[X+1] SHR 4));
    IF (Pos = 0) THEN BEGIN
     Size := (Source^[X+1] SHL 8) + Source^[X+2] + 15;
     FOR K := 0 TO Size DO begin
         Dest^[Y+K] := Source^[X+3];
     end;
     INC(X,4);
     INC(Y,Size+1)
    END
    ELSE BEGIN  { pos = 0 }
     Size := (Source^[X+1] AND $0F)+2;
     FOR K := 0 TO Size DO
         Dest^[Y+K] := Dest^[Y-Pos+K];
     INC(X,2);
     INC(Y,Size+1)
    END; { pos = 0 }
   END;  { command and $8000 }
   Command := Command SHL 1;
   DEC(Bit)
  END { while x < sourcesize }
 END;
 Result := Y
END;  { decompression }

{
 Unit "Finalization" as Delphi 2.0 would have it
}
var
 ExitSave : Pointer;
```

```
Procedure Cleanup; far;
begin
  ExitProc := ExitSave;
  if (Hash <> Nil) then
    Freemem(Hash, Sizeof(HashTable));
end;

Initialization

  Hash := Nil;
  try
    Getmem(Hash,Sizeof(Hashtable));
  except
    Raise ELzrw1KHCompressor.Create('LZRW1KH : no memory for HASH table');
  end;
  ExitSave := ExitProc;
  ExitProc := @Cleanup;
END.
```

Algoritmul Lempel-Ziv este asimptotic optimal. La demararea procesului de codare, ineficiența este maximă. Se constată că "secțiunea critică" a algoritmului o constituie scanarea repetată pentru a determina cel mai lung prefix utilizabil. Performanțele tipice obținute cu acest algoritm sunt rate de compresie de 50-60%[38].