23. COMPACTAREA DATELOR

23.1 Parametrii stocării datelor

Volumul datelor este dat în mai multe feluri. Pentru o colectivitate ale cărei indivizi se descriu cu acelaşi şablon, volumul informaţiilor este prezent prin numărul indivizilor. Avem o imagine suficient de clară, despre un fişier care conţine informaţii referitoare la 30.000 persoane sau despre o matrice are 50 linii şi 80 coloane, din punct de vedere al volumului de date.

Pentru o mai bună precizare, se vor lua în considerare:

- *N* - numărul de indivizi ai colectivităţii;
- *L* - lungimea structurii de date asociate unui individ;
- *B* - factorul de blocare;
- *R* - lungimea informaţiilor reziduale;

Volumul de date *V* este dat de relaţia:

$$V = f(N * L, B) + g(R) \quad (23.1)$$

unde *f* şi *g* sunt forme analitice ale dependenţei dintre factorii consideraţi, forme ce se determină pentru tipuri de suport extern de date, în mod corespunzător.

Volumul de date astfel calculat este exprimat în număr de baiţi. Documentaţiile tehnice consemnează capacitatea de memorare pentru fiecare tip de suport extern, ca număr maxim de baiţi.

Fie suportul extern *i*, având capacitatea C_i. Raportul:

$$p_i = \frac{V}{C_i} * 100 \qquad (23.2)$$

reprezintă ponderea pe care o au datele memorate în volumul *V*, faţă de capacitatea suportului.

De exemplu, pentru un suport de 1200 ko capacitate, un fişier care ocupă 400 ko, ocupă 33% din suport.

Dacă un programator trebuie să stocheze informaţii pe un suport de capacitate C_i, sub forma unor volume $V_1, V_2, \ldots, V_n$, el stochează numai *k* volume, întrucât:

$$\sum_{j=1}^{k} V_j \le C_i \qquad (23.3)$$

Apare însă problema existenţei unei diferenţe, care descrie funcţia obiectiv:

$$[\min] C_i - \sum_{j=1}^{k} V_j \qquad (23.4)$$

a unui model de optimizare a combinaţiei de volume, care să conducă la acest obiectiv.

De exemplu, se consideră:

$$C_i = 1000,\ V_1 = 200,\ V_2 = 500,\ V_3 = 400,\ V_4 = 300 \qquad (23.5)$$

Se calculează sumele:

$$S_1 = V_1 + V_2 + V_3 + V_4 = 1400 > C_i \qquad (23.6)$$
$$S_2 = 200 + 500 + 400 = 1100 > C_i \qquad (23.7)$$
$$S_3 = 200 + 500 + 300 = 1000 = C_i \qquad (23.8)$$
$$S_4 = 500 + 400 = 900 < C_i \qquad (23.9)$$
$$S_5 = 500 + 400 + 300 = 1200 > C_i \qquad (23.10)$$
$$S_6 = 200 + 500 = 700 < C_i \qquad (23.11)$$
$$S_7 = 500 + 300 = 800 < C_i \qquad (23.12)$$
$$S_8 = 200 + 300 = 500 < C_i \qquad (23.13)$$
$$S_9 = 200 + 400 = 600 < C_i \qquad (23.14)$$
$$S_{10} = 400 + 300 = 700 < C_i \qquad (23.15)$$

Din toate combinaţiile, S_3 reprezintă varianta care conduce la o bună umplere a capacităţii cu date.

Apar deci ca parametri de descriere ai utilizării unui suport, următorii:

- gradul de ocupare;
- numărul de fişiere;
- volumul de informaţii stocat în fişier exprimat prin intermediul numărului de indivizi pentru care se face stocarea sau numărul de cuvinte, depinzând de unităţile de măsură, de natura fişierelor.

Problema maximizării, apare pentru:

- creşterea gradului de umplere;
- creşterea numărului de fişiere ce se stochează;
- creşterea volumului de informaţii.

Există modalităţi specifice, care vin să amelioreze unul sau altul dintre parametrii consideraţi, ceea ce influenţează însă asupra tuturor parametrilor, este compactarea datelor.

Prin compactarea datelor, înţelegem totalitatea metodelor care conduc la reducerea lungimii exprimate în baiţi, a datelor. Fiind dată o mulţime de cuvinte:

$$A = \{a_1,\ a_2,\ \ldots\ a_n\},\ \text{de lungime } l_1, l_2, \ldots l_n \qquad (23.16)$$

compactarea datelor revine la a găsi funcţiile:

$$f : A \to K \quad \text{şi} \quad g : K \to A \qquad (23.17)$$

unde:

$$k = \{ k_1,\ k_2, \ldots, k_m\} \qquad (23.18)$$

este mulţimea cuvintelor compactate, aşa fel încât există o pereche (i, j), pentru care:

$$k_j = f(a_i) \qquad (23.19)$$
$$a_i = g(k_j) \qquad (23.20)$$

Funcţia *f( )* se numeşte funcţie de compactare, iar funcţia *g( )* este funcţia de decompactare.

Deci, orice metodă de compactare este complet definită, dacă s-a identificat şi modalitatea de a reduce setul de date în forma iniţială, prin decompactare.

Pentru perechea *(i , j)*:

$$lg(k_j) < lg(a_i) \qquad (23.21)$$

Rezultă că efectul compactării, pentru un text format din cuvintele:

$$a_1\ a_2\ a_3\ a_3\ a_3\ a_3 \qquad (23.22)$$

de lungime iniţială:

$$L_1 = 1g(a_1\ a_2\ a_3\ a_3\ a_3\ a_3\ a_3) = 1g(a_1) + 1g(a_2) + 4*1g(a_3) \qquad (23.23)$$

prin compactare este transformat în textul:

$$k_1\ k_2\ k_3\ k_3\ k_3\ k_3 \qquad (23.24)$$

de lungime:

$$L_2 = 1g(k_1) + 1g(k_2) + 4*1g(k_3) \qquad (23.25)$$

cu indicile de eficienţă a compactării:

$$p = \frac{L_1 - L_2}{L_1} * 100 \qquad (23.26)$$

23.2 Compactarea la nivel de caracter

Sistemele de coduri asociate caracterelor pornesc de la următoarele aspecte:

- mulţimea caracterelor ce sunt reprezentate este finită; de exemplu, codul ASCII permite reprezentarea unei mulţimi de caractere formate din 256 elemente;
- lungimea exprimată în biţi a unui element al mulţimii, este constantă; codul ASCII asociat unui caracter are 8 biţi.

Pentru un şir de *n* biţi, se asociază o mulţime formată din 2^n elemente distincte, ce sunt puse în corespondenţă cu simboluri sau cuvinte, ale unei mulţimi cunoscute.

Vom considera un roman, în care apar numai litere mici şi litere mari, semne de punctuaţie, separatorul blanc şi liniuţa corespunzătoare semnului de dialog.

Analiza textului ce formează romanul, pune în evidenţă următoarele aspecte:

- dintre literele mari sunt utilizate 15;
- dintre literele mici sunt utilizate 24;
- semnele de punctuaţie cu liniuţa de dialog, sunt în număr de 8.

Alfabetul nou cu care operăm, este format din 24+15+8 = 47 simboluri. Fiecare simbol are reprezentare pe 6 biţi, întrucât cel mai mic număr natural pentru care $2^n > 47$ este $n = 6$.

Se vor pune în corespondenţă cele 47 de caractere, cu coduri de câte 6 biţi şi textul romanului care avea o lungime iniţială L_i:

$$L_i = 8 * m \qquad (23.27)$$

unde m reprezintă numărul de caractere al textului.

După compactarea la nivel de caracter, textul compactat are o lungime finală L_f:

$$L_f = 6 * m \qquad (23.28)$$

Indicele de eficienţă al acestei compactări este:

$$p = \frac{L_i - L_f}{L_f} \cdot 100 = 25\% \qquad (23.29)$$

Decompactarea, presupune interpretarea succesiunilor de 6 biţi şi înlocuirea lor cu caractere ASCII corespunzătoare din stiva caracterelor utilizate în text.

23.3 Compactarea la nivel de cuvânt

Prin analiza textului, înţelegem construirea unei stive a cuvintelor diferite din text şi înregistrarea frecvenţei lor de apariţie.

Dacă menţinând codul ASCII pentru caracterele uzuale ale textului, punem în corespondenţă cuvintele având frecvenţele cele mai mari $C_1, C_2, \ldots, C_k$, cu coduri asociate unor caractere ce nu apar în text, indicile de eficienţă al compactării este:

$$p = \frac{L_i - \sum_{j=1}^{k} f_i * 1g(c_j) + \sum_{j=1}^{k} f_j}{L_i} * 100 \qquad (23.30)$$

Dacă stiva cu cuvintele definite $C_1, C_2, \ldots, C_k$, având frecvenţe ridicate $f_1, f_2, \ldots, f_k$, este suficient de mare, şi mulţimea G a simbolurilor neutilizate în text, este insuficientă, este necesară construirea cuvintelor $g_1, g_2, \ldots, g_k$ formate din $1, 2, \ldots, n_g$ simboluri neutilizate, atunci performanţa compactării este:

$$p = \frac{L_i - \sum_{j=1}^{k} f_j * [1g(C_j) - 1g_j)]}{L_i} * 100 \qquad (23.31)$$

Decompactarea revine la a înlocui cuvintele g_j din text, cu cuvintele c_j, întrucât atât algoritmul de compactare cât şi cel de decompactare presupun existenţa stivei cuvintelor diferite ale textului iniţial şi cuvintele

din mulţimea $\{g_1, g_2, \ldots, g_k\}$, a cuvintelor formate din simboluri neutilizate în text.

23.4 Compactarea prin analiza caracteristicilor textului

Vom exemplifica modul de analiză al unui text, folosind reprezentarea în memorie a tabelului de mai jos ce trebuie imprimat:

Tabelul nr. 23.1 *Situaţia materialelor*

5	30	15
NR. CRT.	**DENUMIRE**	**VALOARE**
0	1	2
1	CUIE	100
2	TABLA	200
3	VAR	600
TOTAL		900

Pentru memorarea acestui tabel, se defineşte un articol de 74 baiţi şi în fişierul *TABEL.DAT*, vor fi memorate 20 de rânduri, incluzând şi blancurile dintre antet şi capul de tabel. În fişierul *TABEL.DAT* vor fi ocupaţi 20*74=1480 baiţi cu acest tabel.

Prin convenţie, întrucât asteriscul nu este utilizat, în continuare este folosit ca separator, iar două asteriscuri consecutive au semnificaţia de *CR*.

```
27    30b*SITUATIA MATERIALELOR
23    30b*22 - **74b**10b*
27    54 = **10b*!*5b!*30b*!*15b**
60    10b*!  NR.    !           DENUMIRE            !          VALOARE
!**
25    10b*!  CRT   !*30b*!*15b*!**
22    10b*!*5b*!*30b*!*15b**
69    10b*54 = **10b*!    0    !     1      !     2     !**
69    10b*54 = **10b*!    1    !  CUIE  !   100   !**
60    10b*!    2     !    TABLA   !  200  !**
60    10b*!    3     !    VAR        !  600  !**
  9   10b*54 = **
65    10b*!            TOTAL          !       900      !**
10b*54 =
----
523 baiți
```

Textul astfel codificat are lungimea de 523 baiţi.

Indicele de performanţă în acest caz este:

$$p = \frac{1480 - 523}{1480} * 100 = 64\% \qquad (23.32)$$

Compactarea merge mai în profunzime, prin identificarea elementelor invariante. Apar în mod repetat:

```
10b*54 = **
10b*!*5b*!*30b*!*15b*!**
```

Dacă aceste succesiuni vor fi înlocuite, prima cu caracterul *&* iar a doua cu @:

$$p' = \frac{1480 - (523 - 4*8 - 21)}{1480} * 100 = 68\% \qquad (23.33)$$

Dacă se pune în corespondenţă construcţia ***10b** cu *:*, care are 10 apariţii, încă se obţine o ameliorare a indicelui de eficienţă.

23.5 Compactare prin asocierea unor coduri ce permit eliminarea separatorilor

În textele de orice tip ar fi ele, se utilizează diverşi separatori, care ocupă un număr de poziţii, depinzând de regulile acceptate de utilizare, dintre care se enumeră:

- separatorii punct şi virgulă sunt urmaţi de un spaţiu;
- separatorul linie de dialog când este urmat de o literă mare sau precedat de punct sau două puncte, este precedat de 3-6 spaţii pentru a marca un dialog;
- cuvintele sunt separate prin cel puţin un spaţiu; în procesarea de texte, variabilitatea numărului de spaţii depinde de lăţimea textului şi de dorinţa de a realiza o aliniere la extremităţile definite pentru un rând.

Se consideră de exemplu, înregistrarea profesiilor persoanelor ce alcătuiesc o colectivitate. Din înregistrările efectuate, rezultă mulţimea de profesii distincte:

forjor
strungar
frezor
economist
mecanic
supraveghetor

Fişierul ce se creează, conţine înşiruirea acestor profesii, urmând ca programele pentru consultarea lui, să permită numărarea elementelor ce aparţin fiecărei meserii.

Înregistrarea brută a informaţiilor, conduce la ocuparea unei zone de memorie:

$$Lb = n_1*lg(forjor) + n_2*lg(strungar) + n_3*lg(frezor) + n_4*lg(economist) + n_5*lg(mecanic) + n_6*lg(supraveghetor) \qquad (23.34)$$

Dacă fiecare meserie este pusă în corespondenţa cu un mnemonic precum:

fo	*pentru*	*forjor*
st	*pentru*	*strungar*
fr	*pentru*	*frezor*
ec	*pentru*	*economist*
me	*pentru*	*mecanic*
su	*pentru*	*supraveghetor*

în mod sever, lungimea ocupată se reduce.

$$L_m = 2*(n_1 + n_2 + n_3 + n_4 + n_5 + n_6) \qquad (23.35)$$

Pentru eliminarea ambiguităţii generate de reducerea lungimii mnemonicelor de la 2 caractere la un singur caracter, se efectuează punerea în corespondenţă a meseriilor cu caracterele:

f – forjor
s – strungar
r – frezor
e – economist
m – mecanic
g – supraveghetor

În aceste condiţii, lungimea textului este:

$$L_s = n_1 + n_2 + n_3 + n_4 + n_5 + n_6 \qquad (23.36)$$

Forma brută a caracterelor, conduce la calculul lungimii fişierului în baiţi. Lungimea efectivă, exprimată în biţi, este în continuare redusă dacă meseriile sunt puse în corespondenţă cu şiruri de biţi, ce se bucură de o serie de proprietăţi suplimentare:

forjor	*1*
strungar	*101*
frezor	*1001*
economist	*10001*
mecanic	*10101*
supraveghetor	*100001*

Aceste construcţii, se bucură de proprietatea ca prin concatenare, locul unde s-a produs această operaţie apar două cifre binare de 1. Astfel:

$$L_B = n_1*1 + n_2*3 + n_3*4 + n_4*5 + n_5*5 + n_6*6 \qquad (23.37)$$

Observăm că:

$$8*L_b > 8*L_s > L_B \qquad (23.38)$$

În multe cazuri, lungimea câmpului este dimensionată în aşa fel încât, să poată cuprinde meserii cu cele mai multe caractere, fişierul având articole de lungime fixă.

În exemplul dat, lungimea este dată de cuvântul "supraveghetor", care are 13 caractere.

$$L_F = 8 \cdot 13 \cdot (n_1 + n_2 + n_3 + n_4 + n_5 + n_6) \qquad (23.39)$$

Toţi indicii de performanţă, se calculează în raport cu lungimea L_F.

Dacă de exemplu, într-o colectivitate de 1000 persoane:

$$n_1=100,\ n_2=300,\ n_3=100,\ n_4=100,\ n_5=300,\ n_6=100 \qquad (23.40)$$

$$L_F=8*13*1000=104000 \text{ biţi} \qquad (23.41)$$

$$L_b=100*6+300*8+100*6+100*9+300*7+100*13=7900 \text{ baiţi} \qquad (23.42)$$

$$L_B=100+300*3+100*4+100*5+300*5+100*6=4000 \text{ biţi} \qquad (23.43)$$

În cazul în care lungimea codurilor asociate, iau în considerare frecvenţele aşa fel încât, meseria cu cea mai mare frecvenţă să fie codul de lungime cel mai mic, se obţine:

$$L_B'=300*1+300*3+100*4+100*5+100*5+100*6=3000 \text{ biţi} \qquad (23.44)$$

$$p_1 = \frac{104000-3000}{104000} * 100 = 97\% \qquad (23.45)$$

$$p_2 = \frac{7900*8-3000}{7900*8} * 100 = 95\% \qquad (23.46)$$

$$p_3 = \frac{4000-3000}{4000} * 100 = 25\% \qquad (23.47)$$

23.6 Compactarea prin identificarea de subşiruri repetitive

Pentru cele 27 litere ale alfabetului, se construieşte matricea frecvenţelor de apariţie a grupurilor de câte două litere, *X*. Astfel, x_{ij} reprezintă numărul de apariţii al literei cu poziţia *i*, urmată de literă cu poziţia *j* din alfabet.

Construirea matricei *X* se realizează, prin parcurgerea unei diversităţi de texte şi frecvenţele depind de particularităţile fonetice ale fiecărei limbi.

Dintre grupurile de câte două litere, vor fi extrase acelea cu frecvenţele cele mai mari şi vor fi dispuse pe linii într-un tabel, ale cărui coloane conţin literele alfabetului.

Se construieşte matricea *Y*, ale cărei elemente y_{ij}, conţin frecvenţele de apariţie ale grupului de două litere de pe linia *i*, urmat de litere de pe coloana *j* a tabelului.

Se are în vedere că totalitatea grupurilor de litere ce vor fi selectate în ordinea descrescătoare a frecvenţelor de apariţie, să nu depăşească un număr *K* aşa fel încât:

$$K + L < 256 \qquad (23.48)$$

unde *L* reprezintă numărul de caractere considerat necesar pentru introducerea unui text într-un fişier.

În continuare, se vor considera cele 27 litere ale alfabetului, cele 10 simboluri ale cifrelor şi caracterele: spaţiu, plus, minus, egal, punct, virgulă, două puncte, punct şi virgulă, semnul mirării, semnul întrebării şi asteriscul. În total sunt 48 de caractere.

Din analizele statistice efectuate pe texte, se reţin grupurile de litere alcătuind o mulţime formată din 64 de elemente dispuse în tabloul de mai jos, care au în dreptul liniilor şi coloanelor combinaţii de biţi care alcătuiesc codul asociat fiecărui şir.

Tabelul nr. 23.2 *Combinaţii de biţi asociate şirurilor de caractere*

	1000	1001	1010	1011	1100	1101	1110	1111
1000	u1	1e	pr	mb	mp	ni	in	lui
1001	lor	se	ut	re	tr	te	ta	nu
1010	it	la	ea	ta	ti	ca	oi	au
1011	am	ar	ei	ra	ne	un	ns	nt
1100	cr	sc	st	os	ti	at	ri	oa
1101	sa	ma	ne	tre	ist	tri	urile	ind
1110	nstr	eau	eam	esti	ndu	u-se	ati	nul
1111	asera	res	tit	ros	oasa	isem	tit	art

Aceste grupuri de litere au fost puse în corespondenţă cu codul unui caracter, altul decât cele 48 considerate.

Astfel, versurile eminesciene:

Dintre sute de catarge
Care leagă malurile
Câte oare le vor sparge
Vânturile, valurile

A fost odată ca-n poveşti
A fost ca niciodată
Din rude mari împărăteşti
O prea frumoasă fată.

care însumează 177 caractere incluzând şi spaţiile care separă cuvintele, vor fi compacte astfel:

Dx x sux de x x rge
17 64 26 36 27

x x x aga x lux x
26 24 12 62 57 12

Cix x x x vor spx ge
26 58 24 12 42

Vix ux x , valux x
48 57 12 57 12
A fox odată ca-n povex
53 73

A fox x x ciodată
53 26 16

Dx rude x x ix aratx
17 62 57 15 74

o x x fata
13 33 85

ceea ce conduce la un indice de performanţă:

$$p = \frac{177 - 137}{177} * 100 = 22\% \qquad (23.49)$$

Construirea matricei generale a subşirurilor, are avantajul că este unică pentru orice text care se compactează, dar există posibilitatea de apariţie a situaţiei ca frecvenţele grupurilor în textul de compact să nu urmeze nivelurile de frecvenţă a textelor care au stat la baza obţinerii ei.

Dacă pentru fiecare compactare se construieşte o matrice de subşiruri proprie, performanţa este cu totul alta.

Pentru textul analizat, subşirurile identificare se organizează într-un tabel, căruia i se ataşează o matrice *C* a codurilor, ce conduce la un text de 116 caractere, având un indice de performanţă:

$$p = \frac{177 - 116}{177} * 100 = 34\% \qquad (23.50)$$

Dacă pornim de la ideea că acest text este memorat folosind succesiuni de 6 biţi, pentru că el conţine 25 de combinaţii de biţi, corespunzătoare grupurilor de litere şi cele 27 de litere, comparativ cu memorarea ca text având fiecare caractere câte 8 biţi, indicele de performanţă este:

$$p = \frac{177 * 8 - 116 * 6}{177 * 8} * 100 = 52\% \qquad (23.51)$$

Se observă că algoritmii de compactare, vizează atât lungimea codului sub care se reprezintă un caracter din text, cât şi modul în care se identifică elementele invariante în texte.

Spaţiul ocupat de textul compactat, i se adaugă o zonă cu informaţii care permit reconstituirea textului iniţial. Aceste informaţii conţin:

- lungimea codurilor asociate caracterelor;

- mulţimea subşirurilor şi a codurilor cu care acestea se pun în corespondenţă.

În cazul în care se identifică pentru reguli complexe de scriere a textelor proceduri, acestea se pun în corespondenţă cu coduri şi ori de câte ori apar codurile respective în text, vor fi activate procedurile care vor prelucra textul adecvat.

De exemplu, dacă un text este centrat pe un rând, blancurile vor fi eliminate, respectivul text fiind precedat de un cod care odată identificat, preia textul, îl centrează, reconstituind blancurile eliminate la compactare.

În cazul dialogurilor, începerii unui nou paragraf sau scrierii unei formule, reconstituirea poziţiei reale a textului, se efectuează cu ajutorul procedurilor activate odată cu apariţia codurilor care semnalizează fiecare dintre situaţiile menţionate.

23.7 Compactarea programelor date în formă executabilă

Pentru utilizatori, prezintă importanţă programele executabile. Obiectivele programelor care efectuează compactarea acestui tip de text, sunt găsirea unor modalităţi care să determine stocarea unui număr cât mai mare de programe executabile pe un suport. În acelaşi timp, se urmăreşte şi găsirea posibilităţii de a efectua decompactarea textului transformat.

Programele de compactare a programelor executabile, iau în considerare următoarele aspecte:

- limbajul de asamblare are o mulţime finită de coduri de instrucţiuni, dintre care programatorii folosesc sub 40%, ca diversitate în programe;
- programele executabile sunt rezultate ale compilării şi editării de legături; în cea mai mare parte, aceste operaţii conduc la o pondere ridicată a secvenţelor construite mecanic, pe baza unor reguli tip;
- programele executabile, au ca entitate instrucţiunea şi aceasta este plasată într-o zonă de memorie de lungime şi structură fixă; toate analizele, vizează componentele în număr restrâns de pe o zonă restrânsă; frecvenţele construite vin să ajute alături de celelalte consideraţii, la dimensionarea codurilor cu care se pun în corespondenţă, instrucţiunile programului executabil.

Se consideră în continuare programul:

```
            ORG   420  H
            MOV  D , E
            PUSH  B
            XRA  A
CICLU:      LDAX  B
            ADC  M
            DCR   E
            JZ   BETA
            STAX  B
            INX  B
            INX  H
            JMP  CICLU
BETA:       MOV
```

```
            LOAX  B
            XRA  M
            MOV  A , E
            STAX  B
            STC
            JM  GAMA
            MOV  A , M
            XRA  E
            STC
            JM  DELTA
GAMA: CMC
DELTA:      POP  B
            MOV  E , D
            RET
            END
```

Acestui program îi corespunde codul obiect:

```
0420  53
0421  C5
0422  AF
0423  OA
0424  8E
0425  1D
0426  CA  2F  04
0429  02
042A  03
042B  23
042C  C3  23  04
042F  5F
0430  OA
0431  AE
0432  7B
0433  02
0434  37
0435  FA  3F  04
0438  7E
0439  AB
0440  37
043B  FA  3F  04
043F  C1
0440  5A
0441  C9
0000
```

Textul obiect generat are 31 baiţi. Se vor scrie frecvenţele de apariţie a elementelor din acest text.

Tabelul nr. 23.3 *Frecvenţele de apariţie a elementelor textului*

Element	Frecvenţa
0	8
1	2
2	4
3	9
4	4
5	4
6	-
7	4
8	1
9	1
A	6
B	1
C	3
D	1
E	4
F	6

Împrăştierea cifrelor hexazecimale, reduce masiv posibilitatea efectuării unor prelucrări directe pe textul obiect.

Se procedează la efectuarea modificărilor:

a) se începe contorizarea de la zero şi se memorează 0420 şi textul devine:

```
0000  53
0001  C5
0002  AF
0003  OA
0004  8E
0005  1D
0006  CA  00  OF
0009  02
000A  03
000B  23
000C  C3  00  03
000F  5F
0010  0A
0011  AE
0012  7B
0013  02
0014  37
0015  FA  00  1F
0018  7E
0019  AB
001A  37
001B  FA  00  1F
001F  C1
```

```
0020  5A
0021  C9
```

Cea mai mică valoare din prima cifră hexazecimală a părţii de instrucţiune, este *0* şi cea mai mare este *F*.

Tabelul nr. 23.4 *Frecvenţele de apariţie a elementelor textului*

Element	Frecvenţa
0	5
1	1
2	-
3	2
4	-
5	2
6	-
7	2
8	1
9	-
A	2
B	-
C	5
D	-
E	-
F	2

Din 15 simboluri lipsesc 7. Construim pe 2 baiţi masca elementelor absente:

```
 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
-------------------------------------------------
 1  1  0  1  0  1  0  1  1  0  1  0  1  0  0  1
```

Se analizează frecvenţa de apariţie a cifrelor hexazecimale pe al II-lea bait al instrucţiunii.

Tabelul nr. 23.5 *Frecvenţele de apariţie a cifrelor hexazecimale pe baitul II al instrucţiunii*

Element	Frecvenţa
0	-
1	1
2	2
3	4
4	-
5	1
6	-
7	2
8	-
9	1

A	6
B	2
C	-
D	1
E	3
F	2

Se construieşte pe 2 baiţi masca elementelor absente:

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	*1*	*1*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*1*	*1*	*0*	*1*	*1*	*1*

Se observă că pentru un text de lungime redusă, elementele repetitive nu se identifică şi deci compactarea are efecte reduse, uneori nule sau chiar păgubitoare.

Limbajele de asamblare moderne, au în definire operaţii de tip *RR* (registru-registru) şi întrucât numărul registrelor este redus, sunt asociate coduri diferite pentru combinaţia de cod operaţie *R1, R2* ceea ce reprezintă o compactare la nivel de limbaj. Tot astfel, în cazul instrucţiunilor memorate pe trei baiţi, deplasarea operandului este reprezentată pe un singur baiţ, prin recalcularea ei.

În cazul unor texte mult mai lungi, dacă resursele sunt folosite convenabil şi programatorii stabilesc reguli precise de realizare a unor anumite prelucrări, compactarea este realizabilă.

Astfel, în toate subprogramele, secvenţa de preluare a parametrilor este standardizată şi deci este pusă în corespondenţă cu un cod şi înlocuită cu acesta.

Aplicând principiile de analiză ale textului, compactarea se realizează în principal prin tratarea elementelor repetitive şi în programele scrise în limbajul de asamblare sau generate de compilatoare, acestea nu au o pondere importantă.

23.8 Compactarea datelor numerice

Reprezentarea matricelor rare, este un exemplu de compactare a datelor. În cazul seriilor de date, problema compactării este importantă dacă seriile au variaţii mici sau dacă seriile au o lungime foarte mare.

Fie seria de date *X* având termenii:

11120	*11130*	*11131*	*11136*
11170	*11135*	*11131*	*11109*
11121	*11122*	*11120*	*11104*

Cei 12 termeni ai seriei de date, dacă sunt reprezentaţi binar, ocupă fiecare câte 2 baiţi, deci în total 24 baiţi. Observăm că termenul minim al seriei este 11104, iar termenul maxim este 11170.

$$D = X_{max} - X_{min} = 66 \qquad (23.52)$$

iar:

$$\frac{\mathrm{D}}{\mathrm{Xmin}} = \frac{60}{11108} = 0{,}005 \qquad (23.53)$$

ceea ce arată că datele se află într-un interval foarte îngust în raport cu mărimea lor.

Considerăm $X_{min} = 11104$. Se calculează o nouă serie x':

$$x_i' = x_i - x_{min} \qquad (23.54)$$

al cărei termeni sunt:

16	*26*	*27*	*32*	*66*	*51*
27	*5*	*17*	*18*	*16*	*0*

Numerele acestea se reprezintă pe 7 biţi. Deci, pentru cei 12 termeni sunt necesari 84 de biţi, iar pentru 11104 sunt necesari 2 biţi.

$$\mathrm{p} = \frac{84 + 2*8}{24*8} * 100 = \frac{100}{192} * 100 = 52\% \qquad (23.55)$$

Compactarea la nivel de biţi, se dovedeşte cu efecte mai importante dacă se combină cu alte procedee de compactare.

111	*111*	*111*	*112*	*112*	*112*
113	*113*	*111*	*112*	*112*	*112*

Elementul minim este 111, iar elementul maxim este 113. În reprezentare binară a întregilor, pentru fiecare termen este necesar un baiţ. Seria aceasta necesită 12 baiţi.

Observăm că există elementele repetitive 111, 112 şi 113, care vor fi puse în corespondenţă cu caracterele a, b, c, respectiv 00, 01, 10. Pentru cele trei numere sunt necesari 7 biţi.

$$\mathrm{p} = \frac{3*7 + 12*2}{12*8} * 100 = \frac{45}{96} * 100 = 46\% \qquad (23.56)$$

23.9 Programe care efectuează compactarea

Un program *P* de lungime *L*, se zice că este compactat de programul *X*, dacă forma obţinută *P'*, are o lungime *L'* cu peste 20% mai mică decât lungimea *L*.

Programul *X* execută complet compactarea, dacă încercând compactarea programului *P'*, se obţine un program *P"* de lungime *L"* şi dacă *L' = L"* şi *P"* este identic cu *P'*.

Programul *Y* realizează decompactarea programului *P'*, dacă după executarea sa se obţine programul *P' ' '* de lungime *L' ' '*, aşa fel încât *L' ' ' = L* şi *P' ' '* este identic cu *P'*.

Programele de compactare sunt specializate să aibă ca intrare, fie texte sursă, fie texte obiect, fie componente executabile. Există şi programe generale care efectuează compactarea şi decompactarea. Exemple de astfel de programe sunt PKZIP . EXE, PKUNZIP . EXE, ARJ . EXE, PKARC . EXE. Fiecare dintre acestea folosesc algoritmi specifici de compactare, în funcţie de tipul fişierelor supuse comprimării.

Formatul general al unei etichete de fişier arhivat, îl descriem prin structura următoare în C:

```
struct fisier_compactat
{
      int metoda_compactare[10];
      char nume_fisier[8];
      char extensie_fisier[3];
      unsigned char atribut_fisier;
      short int data;
      short int timp;
      short int cluster_start;
      long int lungime_fisier_initial;
      long int lungime_fis_compactat;
      int suma_control;
};
```

Se asociază fiecărei metode de compactare folosită un anumit cod, ca de exemplu:

- 0, dacă fişierul a fost memorat în forma sa iniţială, compactarea nefiind eficientă;
- 1, dacă fişierul a fost comprimat printr-un algoritm nerepetitiv;
- 2, dacă fişierul a fost comprimat în mai multe etape.

Aceste probleme sunt înzestrate cu funcţii specifice gestionării de fişiere arhivate, respectiv adăugare, ştergere, modificare, protecţie.

Unul dintre algoritmii utilizaţi de aceste programe de compactare este cel al lui Huffman. Acest algoritm are drept scop de a minimiza numărul de biţi necesar pentru reprezentarea oricărui simbol dintr-un alfabet. Astfel, se atribuie simbolurilor cu o frecvenţă de utilizare mare, coduri mai scurte, iar simbolurilor mai rar folosite, coduri mai lungi. În acest mod, media lungimii codului, este mai redusă.

Această tehnică a utilizat-o şi Samuel Morse când a definit alfabetul Morse.

În codul Huffman, frecvenţa de apariţie a caracterelor din textul sursă, trebuie cunoscută sau estimată. Apoi, se asociază fiecărui simbol o secvenţă de biţi unică, care este definită utilizând un arbore binar.

Un cod Huffman este construit astfel:

- se ordonează descrescător, după frecvenţă sau probabilitatea de apariţie, simbolurile din textul sursă;
- se consideră fiecare simbol un nod în arbore, fiecărui nod fiindu-i asociat o probabilitate (frecvenţa) de apariţie;
- se leagă două noduri cu probabilitatea de apariţie cea mai mică, într-un nod a cărui probabilitate este dată de suma probabilităţilor celor două noduri acest proces se încheie în momentul în care rămâne un nod nelegat.

Rezultatul acestei operaţii este un arbore binar, în care ramura stângă a unui nod părinte are eticheta *0*, iar ramura din dreapta, eticheta *1* .

Să presupunem că textul nostru iniţial este format din 40 de caractere, utilizând un alfabet din 8 simboluri; frecvenţele de apariţie ale fiecărui simbol, sunt calculate ca raport între numărul de apariţii a simbolului respectiv şi numărul total de caractere ale textului.

În tabelul de mai jos, presupunem numărul de apariţii ale simbolurilor şi în funcţie de acest număr calculăm numărul total de biţi pe care îl ocupă simbolul respectiv, în reprezentarea normală (8 biţi/caracter).

Tabelul nr. 23.6 *Varianta I de construire a codului Huffman*

Simbol	**Nr. de apariţii simbol**	**Total nr. de biţi pentru un simbol (reprez. normală)**	**Total nr. de biţi pentru un simbol (cod Huffman)**
0	12	96	12
1	8	64	16
2	6	48	18
3	5	40	20
4	4	32	20
5	3	24	18
6	2	16	14
7	0	0	0
Total	**40**	**320**	**118**

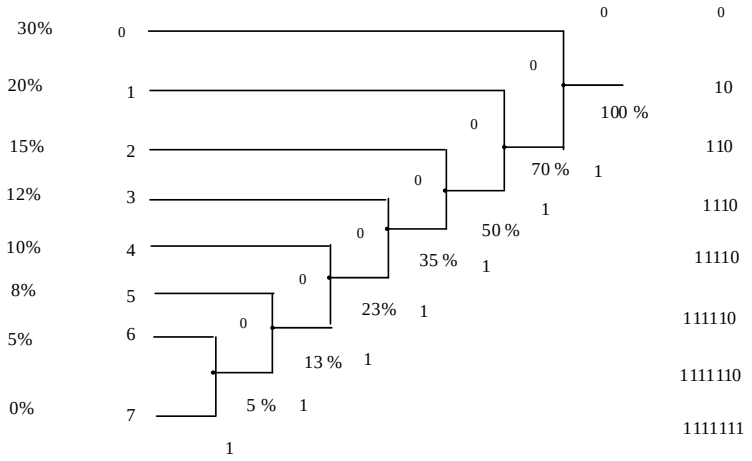

Figura 23.1 *Structura arborescentă a codurilor Huffman*

Dacă însă, cuplăm nodurile în felul următor:

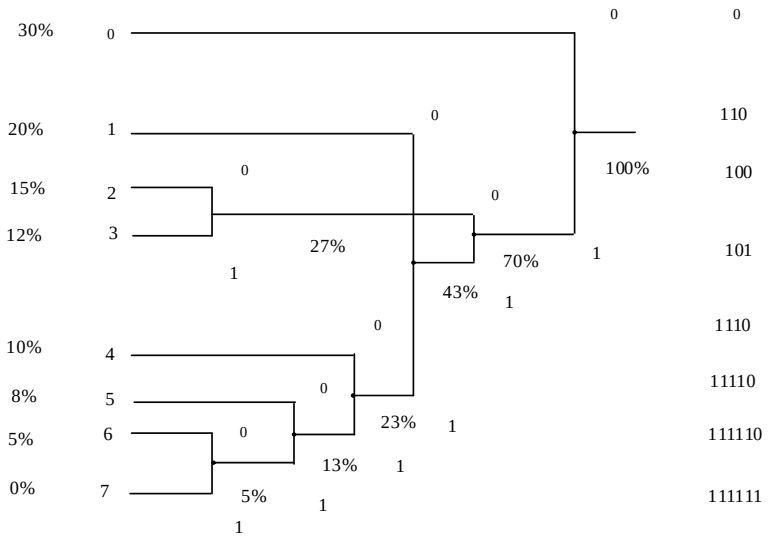

Figura 23.2 *Structura arborescentă a codurilor Huffman după cuplarea nodurilor*

vom obţine:

Tabelul nr. 23.7 *Varianta II de construire a codurilor Huffman*

Simbol	**Nr. de apariţii simbol**	**Total nr. de biţi pentru un simbol (reprez. normală)**	**Total nr. de biţi pentru un simbol (cod Huffman)**
0	12	96	12
1	8	64	24
2	6	48	18
3	5	40	15
4	4	32	16
5	3	24	15
6	2	16	12
7	0	0	0
Total	**40**	**320**	**112**

şi se observă că am obţinut un număr total de biţi mai mic decât în prima variantă, iar diferenţa este cu atât mai mare, cu cât avem texte de lungime mai mari.

Observăm, că pe lângă faptul că codul Huffman foloseşte în medie, un număr de biţi mai mic decât codul ASCII pentru reprezentarea unui caracter, mai are proprietatea că secvenţa binară asociată fiecărui simbol, nu este prefixul unui alt simbol şi de aceea, secvenţele de biţi sunt concatenate fără să se folosească nici o punctuaţie de delimitare a acestora.