CAPITOLUL 5

TEHNICI DE DICȚIONAR

În capitolele anterioare s-au studiat tehnici de codare care folosesc presupunerea că mesajele sursei sunt independente şi identic distribuite. Cum multe surse generează secvențe de mesaje care sunt corelate, codarea este în general precedată de decorelare.

Tehnicile de dicționar țin seama de structura datelor pentru a creşte volumul compresiei. Aceste tehnici (atât cele statice, cât şi cele dinamice) se bazează pe construcția unei liste cu cele mai frecvente structuri care apar, numite modele, şi le codează prin transmiterea unui cuvânt care indică poziția lor în listă. Aceste metode de codare sunt utile pentru surse care generează un număr mic de modele cu frecvență ridicată, cum sunt textele şi comenzile de calculator.

5.1. Introducere

În multe aplicații ieşirea unei surse este formată din secvențe de mesaje care formează modele care se repetă. Un exemplu clasic este un text în care anumite expresii sau cuvinte (modele) revin constant, în timp ce alte modele apar foarte rar. Esența codării pe bază de dicționar constă în întocmirea unei liste cu cele mai frecvente modele, acestea

fiind codate prin indicarea poziției lor în listă, cunoscută sub numele de index. Astfel, intrarea în codor se împarte în două părți:

- cu modele cu apariție frecventă;
- cu modele cu apariție rară.

Pentru ca această tehnică să fie eficientă, clasa modelelor frecvente şi, implicit, mărimea dicționarului, trebuie să fie mult mai mică decât numărul total de modele.

De exemplu, dacă se presupune existența a 32 de caractere (litere şi semne de punctuație) şi se codează caracterele individual, fiecare fiind egal probabile, ar fi necesari 5 biți/caracter. Dacă se presupun modele formate din 4 caractere, ar rezulta 32^4=2048576 modele, necesitând 20 biți/model. Dacă se presupune că 256 de modele sunt mai frecvente, acestea se grupează într-un dicționar. Pentru codarea fiecărui model din cele 256 sunt necesari 8 biți. Pentru transmiterea unui model care există în dicționar, se va transmite un bit de semnalizare (fie acesta 0), urmat de un index de 8 biți, care identifică cuvântul din dicționar. Dacă cuvântul nu este în dicționar se trimite un bit de semnalizare (fie acesta 1), urmat de 20 de biți care codează modelul. Dacă modelul nu este în dicționar, se trimit mai mulți biți decât în schema originală (adică 21 în loc de 20), iar dacă este în dicționar, se transmit 9 biți în loc de 20.

Eficiența tehnicii depinde de procentul cuvintelor care se află în dicționar. Dacă probabilitatea unui model din dicționar este p şi, evident, 1-p este probabilitatea ca modelul să nu fie în dicționar, atunci numărul mediu de biți/model este

$$R = 9p + 21(1-p) = 21 - 12p \qquad (5.1)$$

Pentru ca schema de codare să fie eficientă, R trebuie să fie mai mic decât 20, ceea ce se întâmplă pentru $p \geq 0,08(3)$. Dacă toate

modelele ar fi egal probabile, probabilitatea fiecăruia este de aproximativ 0,00025.

În practică se doreşte o rată cât mai mică, ceea ce conduce la condiția ca *p* să fie cât mai mare. Aceasta înseamnă că modelele trebuie selectate atent dintre cele mai frecvente, ceea ce impune o bună cunoaştere a structurii sursei.

În funcție de cunoştințele disponibile pentru construcția dicționarului, se poate folosi o codare *statică* sau una *dinamică*.

5.2. Dicționar static

Tehnica de codare bazată pe dicționar static este potrivită când sunt disponibile cunoştințe a priori despre sursă.

O tehnică de dicționar static este *codarea digram*. Dicționarul este format din toate literele alfabetului sursei şi cele mai frecvente perechi de două litere, numite *digrame*. De exemplu, se presupune că s-a construit un dicționar de mărime 256, format din 95 caractere ASCII şi 161 digrame care sunt cele mai frecvent utilizate perechi de două caractere. Se codează apoi cele 256 de modele pe 8 biți corespunzând celor 95 de caractere ASCII şi celor 161 de digrame. Dacă se doreşte codarea unei secvențe de caractere ASCII se citesc primele două caractere din secvență şi se caută dacă există în dicționar. Dacă da, indexul digramei este codat şi transmis. Dacă nu, primul caracter al perechii este transmis prin codul corespunzător indexului caracterului, iar al doilea caracter devine primul caracter al digramei următoare. Codorul mai citeşte un caracter pentru a completa digrama şi procedura se repetă.

Exemplul 5.1.

Fie sursa cu alfabetul $S = \{a, b, c, d, r\}$. Pe baza cunoaşterii sursei, se construieşte dicționarul din Tabelul 5.1.

Tabelul 5.1.

Intrare	Cod	Intrare	Cod
a	000	r	100
b	001	ab	101
c	010	ac	110
d	011	ad	111

Se presupune că se doreşte a coda secvența *a b r a c a d a b r a.*

Deoarece digrama *ab* este în dicționar, aceasta se codează cu 101, apoi se citeşte *ra*, care nu există în dicționar, se codează *r* cu 100, apoi se citeşte *ac*, care se codează cu 110. Continuând, se obține: 101 100 110 111 101 100 000.

Dacă s-ar transmite pentru fiecare din cele 11 simboluri ale sursei câte 3 biți (deoarece sunt 5 simboluri diferite), ar rezulta 33 biți. Prin folosirea acestui dicționar s-au transmis $7 \cdot 3 = 21$ simboluri.

În Tabelele 5.2 şi 5.3 se dau cele mai frecvente digrame dintr-un document LATEX, respectiv, dintr-un program C [86].

Tabelul 5.2. Cele mai frecvente 30 de perechi de caractere dintr-un document LATEX de 41.364 caractere

Pereche	Număr de apariții	Pereche	Număr de apariții
e␣	1128	*ar*	314
␣t	838	*at*	313

␣␣	823	*␣w*	309
th	817	*te*	296
he	712	*␣s*	295
in	512	*d␣*	272
s␣	494	*␣o*	266
er	433	*io*	257
␣a	425	*co*	256
t␣	401	*re*	247
en	392	*␣$*	246
on	385	*r␣*	239
n␣	353	*di*	230
ti	322	*ic*	229
␣i	317	*ct*	226

Tabelul 5.3. Cele mai frecvente 30 de perechi de caractere dintr-un program C de 65.983 caractere

Pereche	Număr de apariții	Pereche	Număr de apariții
␣␣	5728	st	442
nl␣	1471	le	440
;nl	1133	*ut*	440
in	985	*f(*	416
nt	739	*ar*	381
= ␣	687	*dr*	374
␣i	662	*r␣*	373
t␣	615	*en*	371
␣ =	612	*Er*	358
);	558	*ri*	357

,ƀ	554	*at*	352
nlnl	506	*pr*	351
ƀf	505	*te*	349
eƀ	500	*an*	348
*ƀ**	444	*lo*	347

Se observă că cele două tabele sunt foarte diferite. Este uşor de observat că un dicționar proiectat pentru compresia documentelor LATEX nu va fi foarte eficient pentru compresia unui program C, putând conduce la o extensie în loc de compresie.

Frecvent se doreşte a se dispune de tehnici de compresie care să fie capabile să comprime ieşirea mai multor tipuri de surse, fără cunoaşterea prealabilă a statisticii sursei. Pentru asemenea aplicații se folosesc tehnici de adaptare a dicționarului la caracteristicile sursei.

5.3. Dicționar adaptiv

Există două tehnici de dicționar adaptiv, LZ77 şi LZ78, care se datorează lui Jacob Ziv şi Abraham Lempel, pe care se bazează cele mai multe metode de compresie cu dicționar adaptiv.

5.3.1. Metoda LZ77

În metoda LZ77 dicționarul este o porțiune din secvența codată anterior. Codorul examinează secvența de intrare printr-o fereastră glisantă, care constă din două părți:

- un buffer de căutare, care conține o porțiune din secvența tocmai codată;

- un registru care conține următoarea porțiune a secvenței ce urmează a fi codată.

În Fig. 5.1, se consideră pentru simplitate că bufferul conține 8 simboluri, iar registrul, 7, în practică acestea fiind mult mai mari.

Pentru codarea secvenței din registru, codorul mută un pointer de căutare înapoi prin buffer, până găseşte începutul celei mai lungi asemănări cu începutul secvenței stocate în registru. Distanța dintre pointer şi registru se numeşte *offset*. Numărul simbolurilor consecutive din bufferul de căutare şi din registru care coincid cu simbolurile din registru, începând cu primul caracter se numeşte *lungimea secvenței* sau a *asemănării*. Dacă în registru mai există coincidențe între caracterele deja codate şi cele ce urmează a fi codate, începând cu primul caracter din registru, lungimea secvenței se poate prelungi în acesta. Odată găsită această lungime, codorul codează secvența cu tripletul $< o, l, c >$, unde o-offsetul, l-lungimea şi c-cuvântul de cod corespunzător simbolului din registru la care s-a oprit asemănarea.

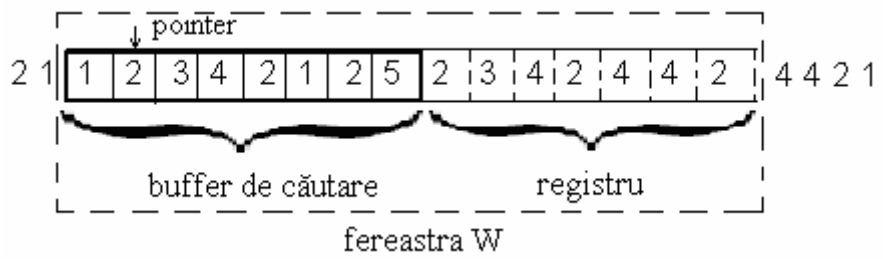

Fig. 5.1

Pentru exemplul din Fig. 5.1, $o = 7, l = 4, c = 4$. Motivul transmiterii celui de-al treilea element din triplet este de a evita situația

când simbolul din registru nu este în buffer. În acest caz $o = 0$, $l = 0$ şi c este codul simbolului însuşi.

Dacă mărimea bufferului de căutare este S, mărimea ferestrei W şi mărimea alfabetului sursei este A, numărul de biți necesari codării unui triplet este $\lceil \log_2 S \rceil + \lceil \log_2 W \rceil + \lceil \log_2 A \rceil$. Al doilea termen este $\log_2 \lceil W \rceil$, şi nu $\log_2 \lceil W - S \rceil$, deoarece lungimea asemănării poate depăşi dimensiunea bufferului de căutare.

În exemplul următor sunt evidențiate situațiile în care:

-nu există nici o potrivire a caracterului ce urmează a fi codat în fereastră;

- există potrivire;

- secvența potrivită se extinde şi în registru.

Exemplul 5.2.

Fie secvența *c a b r a c a d a b r a r r a r r a d*… care se codează cu metoda prezentată anterior.

Se presupun *S*=7, *W*=13. Poziționarea inițială a secvenței în fereastră este dată în Fig. 5.2. Secvența din buffereul de căutare se transmite în clar.

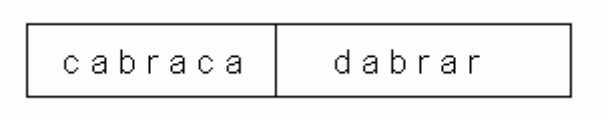

Fig. 5.2.

Pentru *d* nu se găseşte nici o asemănare în bufferul de căutare şi se transmite <0,0,c(d)>. Fereastra se deplasează cu o poziție, ca în Fig. 5.3.

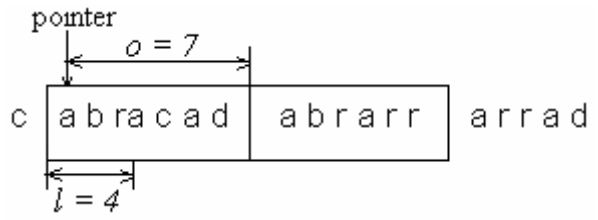

Fig. 5.3.

Căutând în buffer, se găseste potrivire cu *a* la un offset de 2 (cu *l*=1), de 4 (*l*=1) şi de 7, când lungimea asemănării este 4. Se transmite tripletul $< 7, 4, c(r) >$. Se mută fereastra cu 5 caractere, care reprezintă lungimea asemănării (l=4) + caracterul codat în clar c(r), ca în Fig. 5.4.

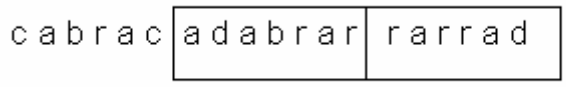

Fig. 5.4.

Acum se găseşte o asemănare pentru *r* la $o = 1\,(l = 1)$ la $o = 3\,(l = 3)$ în buffer, dar prelungind căutarea şi în registru se obține lungimea asemănării *l*=5 şi se transmite $< 3, 5, c(d) >$. În acest caz fereastra se mută cu 6 caractere.

Decodarea

Se presupune că s-a decodat secvența *c a b r a c a* şi s-au recepționat tripletele $< 0, 0, c(d) >, < 7, 4, c(r) >, < 3, 5, c(d) >$. Primul triplet este simplu de decodat, nu există nici o asemănare în secvența decodată şi următorul simbol este *d*. Secvența decodată este acum:

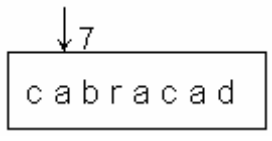

Fig. 5.5.

Primul element din al doilea triplet este 7, care spune unde se plasează pointerul şi apoi se copie 4 caractere din acel punct.

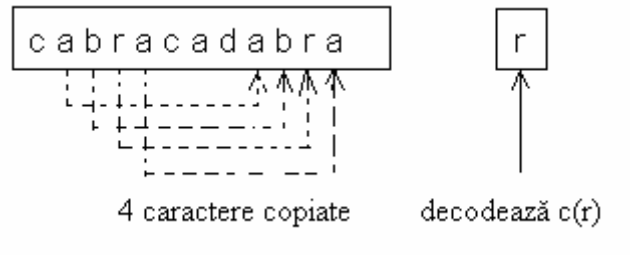

Fig. 5.6.

Secvența decodată este *c a b r a c a d a b r a r*. În final se decodează ultimul triplet. Se merge înapoi cu 3 poziții şi se copie 5 caractere începând cu acel punct.

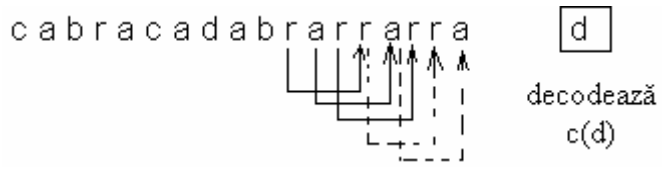

Fig. 5.7.

Se observă că asemănarea începe numai în bufferul de căutare, dar se poate extinde şi în registru. Dacă ultimul caracter din registru ar fi

fost *r* în loc de *d,* urmat de mai multe repetări de *r a r,* întreaga secvență de *r a r* -uri ar fi putut fi codată cu un singur triplet.

Algoritmul LZ77 este o schemă adaptivă foarte simplă, care nu necesită cunoaşterea a priori a sursei. Autorii algoritmului au arătat că performanța algoritmului tinde asimptotic la cel mai bun rezultat ce poate fi obținut folosind o schemă care cunoaşte în întregime statistica sursei.

În practică există modalități de îmbunătățire a performanțelor algoritmului LZ77. Cea mai simplă modificare a algoritmului LZ77 şi una dintre cele mai folosite este de a elimina folosirea unui triplet pentru a coda un singur caracter, ceea ce este foarte ineficient, în special când există un număr mare de caractere care apar rar. Modificarea pentru eliminarea acestei ineficiențe este de a adăuga un bit de semnalizare (flag) care să indice dacă ceea ce urmează este cuvânt de cod pentru un singur simbol. Folosind acest flag, se elimină necesitatea celui de-al treilea element din triplet. Rămâne de transmis o pereche de valori corespuzătoare offsetului şi lungimii. Acest algoritm este cunoscut sub denumirea de LZSS [2,91].

Dintre arhivatorii uzuali care folosesc algoritmul LZ77, urmat de o codare cu lungime variabilă sunt PKZip, Zip, LHarc, ARJ.

5.3.2. Algoritmul LZ78

Algoritmul LZ77 a presupus implicit că modele asemănătoare apar apropiate unele de altele, folosind această structură a secvenței stocate în buffer ca dicționar pentru codare. Aceasta înseamnă că orice model care apare pe o durată mai mare decât fereastra codorului, nu va fi captat. Cea mai defavorabilă situație este cea în care secvența de codat este periodică, cu perioada mai mare decât bufferul de căutare.

Alegerea lungimii bufferului se face în funcție de statistica datelor ce urmează a fi codate.

Pentru exemplificare, se presupune că secvența de caractere are perioada 9, aşa cum se arată în Fig. 5.8.

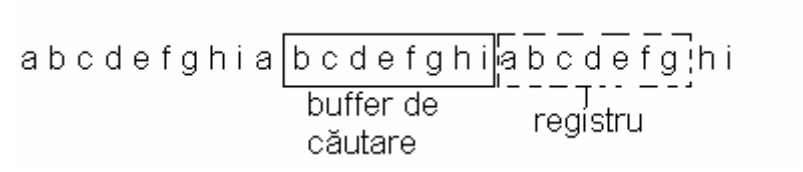

Fig. 5.8.

În situația considerată bufferul are lungimea 8, astfel încât nici un nou simbol nu se potriveşte cu vreunul din buffer şi va fi reprezentat separat. Cum aceasta implică transmiterea mai multor simboluri (1 bit de flag pentru LZSS şi un triplet pentru LZ77), rezultatul este o extensie, în loc de compresie. Dacă, însă, bufferul de căutare era mai lung cu o celulă, secvența era comprimată semnificativ.

Deşi aceasta este o situație extremă, sunt circumstanțe mai puțin drastice în care mărimea limitată a bufferului de căutare ar putea constitui un dezavantaj. Algoritmul LZ78 rezolvă această problemă, prin renunțarea la bufferul de căutare şi întocmirea unui dicționar explicit. Acest dicționar trebuie construit atât la emisie cât şi la recepție în acelaşi mod. Intrarea este codată cu un dublet $< i , c >$, unde:

i - este indexul corespunzător apariției în dicționar;

c – este codul caracterului ce urmează după secvența găsită în dicționar.

Indexul 0 se foloseşte în cazul negăsirii vreunei asemănări. Fiecare nouă apariție în dicționar este un nou simbol concatenat cu o apariție deja existentă în dicționar.

Exemplul 5.3.

Să se codeze secvența d a b b a ␣ d a b b a ␣ d a b b a ␣ d a b b a ␣ d u u ␣ d u u ␣ d u u. În secvența considerată caracterul ␣ semnifică spațiu. Inițial dicționarul este gol. Primele trei ieşiri ale codorului, pâna la repetarea unuia din caracterele deja codate, sunt codate cu valoarea indexului egal cu 0, astfel: $< 0, c(d) >, < 0, c(a) >, < 0, c(b) >$.

Dicționar inițial este

Index	apariție
1	d
2	a
3	b

al patrulea simbol este *b*, care este al treilea element din dicționar. Dacă se adaugă următorul simbol, se obține *ba*, care nu este în dicționar, aşa încât se codează aceste 2 simboluri ca $< 3, 2 >$ şi se adaugă ca al patrulea element în dicționar. Continuând astfel, se obține dicționarul din Tabelul 5.4.

Tabelul 5.4

Ieşire codată	Dicționar Index	apariție	Ieşire codată	Dicționar Index	apariție
<0,c(d)>	1	d	<4,5>	10	ba␣
<0,c(a)>	2	a	<9,3>	11	dabb
<0,c(b)>	3	b	<8,1>	12	a␣d
<3,2>	4	ba	<0,c(u)>	13	u
<0,c(␣)>	5	␣	<13,5>	14	u␣
<1,2>	6	da	<1,13>	15	du
<3,3>	7	bb	<14,1>	16	u␣
<2,5>	8	a␣	<13,13>	17	uu
<6,3>	9	*dab*			

Se observă că aparițiile din dicționar devin din ce în ce mai lungi şi dacă acestea se repetă des, (cum se întâmplă într-un cântec, de exemplu) după un timp, întreaga secvență ar putea fi o apariție în dicționar.

Deşi algoritmul LZ78 are abilitatea de a capta modele şi de a le păstra, el are şi dezavantaje serioase. Cum se vede din exemplu, dicționarul poate creşte oricât de mult. În practică se doreşte creşterea dicționarului până la un anumit moment şi apoi fie simplificarea, fie tratarea lui ca un dicționar fix.

5.3.3. Variante ale algoritmului LZ78

Cea mai cunoscută modificare a algoritmului, cunoscută sub numele de varianta LZW, este făcută de Welch [97], care a propus o tehnică care înlătură necesitatea de codare a celui de-al doilea element al perechii $< i , c >$. Pentru aceasta, dicționarul trebuie încărcat cu toate literele alfabetului sursei. În timpul codării are loc şi completarea dicționarului, astfel: după ce se citeşte prima literă, al cărei "model", fie acesta *m*, se găseşte în dicționar, se concatenează aceasta cu cea de-a doua literă, fie aceasta *a*, pentru a forma un nou model, *m*a* (* - concatenare). Acest model nu se găseşte în dicționar, aşa că se codează *m* cu indexul din dicționar şi se adaugă modelul *m*a* în dicționar, la indexul care urmează literelor alfabetului inițial. Se începe apoi un nou model cu litera *a.* Cum aceasta este în dicționar, se concatenează cu următoarea literă din secvența de codat, fie aceasta *b*, pentru a forma modelul *a*b* care se trece în dicționar la indexul următor, şi aşa mai departe. Dacă în procesul de completare a dicționarului se ajunge la un model deja existent în dicționar, se concatenează acesta cu caracterul

următor, pentru a forma un nou model care se înscrie în dicționar şi procesul continuă în acelaşi mod.

În general, dacă prin adăugarea unei litere, *a*, la un model existent, *m*, rezultă modelul *m*a* care nu este în dicționar, atunci indexul lui *m* este transmis la receptor, modelul *m*a* este adăugat la dicționar şi se începe un alt model cu litera *a*.

Exemplul 5.4.

Se foloseşte aceeaşi secvență utilizată în exemplul anterior:
d a b b a ␣ d a b b a ␣ d a b b a ␣ d a b b a ␣ d u u ␣ d u u ␣ d u u şi se codează cu algoritmul LWZ.

Se presupune că alfabetul sursei este { ␣ , a, b, d, u } şi dicționarul inițial este dat în Tabelul 5.5.

Tabelul 5.5.

Index	Apariție
1	␣
2	a
3	b
4	d
5	u

Codorul întâlneşte întâi litera *d*. Acest "model" este în dicționar, aşa că se concateneaza cu litera următoare formând *da*. Acest model nu este în dicționar, aşa că se codează *d* cu indexul 4 din dicționar, se adaugă modelul *da* în dicționar la indexul 6 şi se începe un nou model plecând de la litera *a.* Cum *a* este în dicționar, se concatenează următorul simbol *b,* pentru a forma modelul *ab*, care nefiind în dicționar, se introduce la indexul 7 şi se începe construcția unui nou

model cu litera *b*. Se continuă în acest mod, construind modele de 2 litere până se întâlneşte litera *d* în a doua secvenţă *d a b b a*. În acest moment, ieşirea din codor constă în întregime din indici ai dicţionarului iniţial, adică 4 2 3 3 2 1 (la a 12-a apariţie în dicţionar). Următorul simbol este *a* care, concatenat cu *d,* conduce la modelul *da* care este în dicţionar la poziţia 6, aşa că se citeşte următorul simbol, *b*, obţinând modelul *dab* care nu este în dicţionar şi se include la poziţia 12 şi se începe un nou model de la *b*. De asemenea, se codează *da* cu 6. Lungimea aparițiilor în dicţionar creşte pe măsură ce are loc codarea. Cu cât aparițiile în dicţionar sunt mai lungi, cu atât dicţionarul captează mai mult din structura secvenței. Dicţionarul complet pentru codarea secvenței din exemplu este dat în Tabelul 5.6.

Tabelul 5.6.

Index	Apariție	Index	Apariție	Index	Apariție
1	*b*	9	ba	17	*b* da
2	a	10	a *b*	18	abb
3	b	11	*b* d	19	ba *b* d
4	d	12	dab	20	du
5	u	13	bba	21	uu
6	da	14	a *b* d	22	u *b*
7	ab	15	dabb	23	*b* du
8	bb	16	ba *b*	24	uu *b*
				25	*b* duu

Secvența codată este:

4 2 3 3 2 1 6 8 10 12 9 11 7 16 4 5 5 11 21 23 5

Decodarea

Se va decoda secvența anterioară, care este intrarea în decodor. Decodorul porneşte cu acelaşi dicționar inițial ca şi codorul. Indexul 4 corespunde literei *d*, aşa că se decodează prima literă din secvență şi, pentru a simula codorul, se construieste următorul element din dicționar, după cum urmează: Se începe cu prima literă decodată *d*, care există în dicționar, aşa că aceasta nu se mai adăugă. Următoarea intrare este 2, care corespunde lui *a*. Se decodează *a* şi se concatenează cu *d*, pentru a forma *da,* al şaselea element care se adaugă în dicționar. Următorul model va începe cu *a*. Următoarele patru intrări 3 3 2 1 corespund literelor *b b a ␣* şi generează modelele *ab, bb, ba* şi *a␣* care se trec în dicționar. Următoarea intrare este 6, care este indexul pentru *da* şi, prin urmare, se decodează un *d* şi un *a*. Întâi se concatenează *d* la modelul existent, *␣*, şi se formează *␣d*. Cum acesta nu există în dicționar, se introduce la poziția 11. Noul model va începe cu *d*. Anterior s-a decodat *a* care, concatenat cu *d,* a dat *da*, care este în dicționar, aşa că se decodează următoarea intrare, care este 8, corespunzător lui *bb*. Se decodează primul *b* şi se concatenează la modelul *da* pentru a obține *dab*, la poziția 12, apoi se începe un nou model cu litera *b*. Decodând al doilea *b* şi concatenându-l la noul model, rezultă modelul *bb*, care există în dicționar, aşa că se decodează următorul element din secvență.

Procedeul se continuă în mod similar, până la decodarea întregii secvențe, pe bază dicționarului construit arătat în Tabelul 5.7.

Tabel 5.7.

Index	Apariție	Index	Apariție	Index	Apariție
1	*␣*	9	*a*	18	*abb*

2	*a*	10	*a␢*	19	*ba␢d*
3	*b*	11	*␢d*	20	*uu*
4	*d*	12	*dab*	21	*uu*
5	*u*	13	*bba*	22	*u␢*
6	*da*	14	*a␢d*	23	*␢du*
7	*ab*	15	*dabb*	24	*uu␢*
8	*bb*	16	*ba␢*	25	*␢duu*
		17	*bda*		

Există o situație particulară în care algoritmul de decodare LZW nu funcționează în modul cum s-a arătat anterior. Se presupune o sursă cu alfabetul $A = \{a, b\}$ şi se doreşte codarea secvenței care începe cu *abababababab*... . Codarea se face la fel ca mai înainte, începând cu dicționarul inițial

Index	Intrare
1	a
2	b

Se construieşte apoi dicționarul, ca în Tabelul 5.8.

Tabelul 5.8.

Index	Apariție
1	*a*
2	*b*
3	*ab*
4	*ba*

5	*aba*
6	*abab*
7	*bab*
8	*baba*

Secvența transmisă va fi atunci: 1 2 3 5 ...

La recepție, se consideră că s-a primit secvența 1 2 3 5 şi se cunosc primele două indexuri din dicționar. Dacă se încearcă decodarea secvenței ca în exemplul anterior, se observă că se ajunge în situația de a decoda indexul 5, înainte de a construi modelul de la acel index. Primele două elemente sunt decodate ca *a* şi *b*, apoi se construieşte al treilea element al dicționarului, *ab*. Primul simbol al următorului model, cu indexul 4, este *b*. Dicționarul construit până în acest moment este prezentat în Tabelul 5.9.

Tabelul 5.9. Construcția intrării de la indexul 4

Index	Apariție
1	*a*
2	*b*
3	*ab*
4	*b*..

Următoarea intrare în decodor este 3, care corespunde intrării *ab*. Decodând fiecare simbol în parte, întâi se concatenează *a* la modelul anterior şi se obține *ba*. Acest model nu este în dicționar, aşa că la indexul 4, obținându-se dicționarul din Tabelul 5.10.

Tabelul 5.10. Construcția intrării de la indexul 5 (prima fază)

Index	Apariție
1	*a*
2	*b*
3	*ab*
4	*ba*
5	*a..*

Noua intrare de la indexul 5 începe cu litera *a*. S-a folosit numai prima literă din perechea *ab*. Prin urmare, se concatenează *b* cu *a* şi se obține modelul *ab*. Acesta este conținut în dicționar, aşa că se continuă. La acest moment dicționarul este dat în Tabelul 5.11. Pentru a decoda un index pentru care nu există o intrare completă, dicționarul se construieşte în mai multe etape, pe bază intrării parțiale cunoscute.

Tabelul 5.11. Construcția intrării de la indexul 5 (faza a doua)

Index	Apariție
1	*a*
2	*b*
3	*ab*
4	*ba*
5	*ab..*

Se observă că s-a ajuns în situația de a se decoda indexul 5 înainte de a construi modelul de la acel indice. Se cunoaşte numai începutul apariției de la numărul 5 şi anume *ab*. În acest moment,

apariția de la indexul 5, dacă ar fi cunoscută, s-ar concatena cu intrarea parțială *ab*, pentru a continua construcția dicționarului. Se concatenează întâi litera *a* la *ab*, obținându-se modelul *aba*, care nu este în dicționar, aşa că se trece acesta la indexul 5 şi se continuă. Decodorul LZW trebuie să conțină o operație de excepție pentru a rezolva cazul particular al decodării unui index care nu are o apariție completă în dicționarul decodorului.

5.4. Aplicații

1. Compresia Fişierelor - Compresia Unix

Comanda Unix `compress` este una din cele mai des folosite aplicații ale algoritmului LZW. Mărimea dicționarului este adaptivă. Pentru început se consideră un dicționar de mărime 512. Aceasta înseamnă că fiecare cuvânt de cod are o lungime de 9 biți. Odată ce dicționarul s-a umplut, mărimea dicționarului este dublată la 1024. Următoarele cuvinte de cod vor avea o lungime de 10 biți. Mărimea dicționarului este dublată pe măsură ce acesta se umple. Se impune ca lungime maximă a cuvântului de cod , $b_{\max}$, un număr ce poate fi setat de utilizator între 9 şi 16 biți. Odată ce dicționarul conține $2^{b_{\max}}$ intrări, compresia devine o tehnică de codare cu dicționar static. Din acest moment algoritmul monitorizează raportul de compresie. Dacă acesta scade sub un anumit prag, se reia procesul de întocmire a dicționarului. În acest fel, dicționarul reflectă tot timpul caracteristicile locale ale sursei.

2. Compresia imaginii - Formatul interfață grafică (GIF)

Formatul Interfață Grafică (GIF) a fost dezvoltat pentru a coda imaginile grafice. Acesta este o altă implementare a algoritmului LZW

şi este foarte asemănător cu comanda Unix `compress`. Primul byte al imaginilor compresate reprezintă numărul minim *b* de biți/pixel din imaginea originală. Numărul binar 2^b este definit ca fiind codul "curat" (CLEAR CODE). Acest cod este folosit pentru a reseta toți parametrii de compresie şi decompresie la un nou start. Mărimea inițială a dicționarului este 2^{b+1}. Când acesta se umple, mărimea dicționarului este dublată, aşa cum s-a procedat şi în algoritmul `compress`, până când este atinsă mărimea de 4096. Din acest moment algoritmul se comportă ca un algoritm cu dicționar static. Cuvintele de cod de la algoritmul LWZ sunt stocate în blocuri de caractere. Caracterele sunt de 8 biți, iar mărimea maximă a blocului este de 256. Fiecare bloc este precedat de un antet (header) care conține mărimea blocului. Blocul se termină cu un bloc final care constă în 8 de zero. Sfârşitul compresiei imaginii este anunțat de un cod de "sfârşit de informație" cu valoarea $2^b + 1$.

Formatul GIF a devenit destul de cunoscut pentru codarea unei diversități de imagini, atât a celor generate de calculator, cât şi a celor "naturale". Deşi GIF lucrează bine cu imaginile grafice generate de calculator şi imaginile color sau monocrome, în general nu este cea mai eficientă metodă de compresie fără pierderi a imaginilor naturale, fotografice, imagini din satelit şi aşa mai departe.