6. MATRICE RARE

6.1 Concepte de bază

Matricele rare îşi găsesc aplicabilitatea în modelarea unor procese de natură industrială, economică, tehnică, socială etc. Capitolul de faţă îşi propune să trateze modalităţile de reprezentare în structuri de date a matricelor rare, precum şi principalele operaţii matriceale implementate într-un limbaj orientat pe obiecte. În final este prezentată o aplicaţie concretă – estimarea parametrilor unei regresii statistice.

În rezolvarea multor probleme de natură economică, tehnică, socială, a diverselor probleme de optimizare, precum şi în modelarea unor procese industriale şi tehnologice este necesar să se determine modelul matematic care descrie funcţionarea procesului respectiv. Descrierea acestor sisteme fizice conduce la obţinerea unor modele matematice care fie în mod direct, prin modelare, fie prin metoda de rezolvare implică sisteme de ecuaţii algebrice liniare sau probleme de programare liniară a căror matrice a coeficienţilor este *rară (sparse)*, în sensul că ponderea elementelor nenule în totalul elementelor matricei este mică.

Din punct de vedere practic trebuie remarcat faptul că analiza sistemelor mai sus amintite conduce la obţinerea unor modele matematice de mari dimensiuni care implică sisteme de ecuaţii algebrice liniare de mii de ecuaţii, pentru a căror rezolvare sunt necesare resurse mari de memorie şi timp de calcul. În multe cazuri practice, cum sunt sistemele în timp real, timpul de calcul este o resursă critică, nefiind permis să depăşească o valoare limită.

Modelele matematice ale proceselor reale implică un număr foarte mare de variabile şi restricţii care prezintă fenomenul de *raritate* ,*sparsity*, adică o slabă interconectare a elementelor sale. Luarea în consideraţie a fenomenului de raritate furnizează un nou mod de abordare foarte eficient, ce implică în dezvoltarea aplicaţiilor informatice folosirea unor structuri de date speciale, care să conducă la reducerea resurselor de memorie şi a timpului de calcul.

În general, o matrice (n,n) - dimensională este rară atunci când conţine un număr mic de elemente nenule τ , adică $\tau << n^2$. Cantitativ, matricele rare sunt caracterizate de ponderea numărului de elemente nenule în totalul de elemente, pondere ce defineşte gradul de umplere al matricei. În aplicaţiile curente se întâlnesc matrice rare cu grade de umplere între 0,15% şi 3%.

6.2 Memorarea matricelor rare

Se consideră matricea:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \end{pmatrix} \qquad (6.1)$$

Matricea *A* este un exemplu de matrice rară, ea conţinând 16 elemente nule din totalul de 20.

Se defineşte gradul de umplere, densitatea, unei matrice prin raportul dintre numărul elementelor nenule şi numărul total al elementelor sale:

$$G = \frac{p}{n \times m} \times 100\,(\%) \qquad (6.2)$$

unde:

p – numărul de elemente nenule;
n – numărul de linii;
m – numărul de coloane.

În general se acceptă că o matrice este rară dacă densitatea sa este de cel mult 3%. Densitatea matricei A este $G(A) = 20\%$, ea fiind prezentată aici în scopul ilustrării conceptului de matrice rară.

Structura de date clasică folosită pentru manipularea matricelor, tabloul de dimensiune *(n, m)* alocat la compilare, se dovedeşte a fi ineficientă în cazul în care matricea este rară. Un prim avantaj este legat de folosirea neeconomică a spaţiului de memorie prin alocarea de zone mari pentru memorarea elementelor nule, care nu sunt purtătoare de informaţie. Ocuparea unor zone de memorie cu elemente nule nu se justifică deoarece acestea nu contribuie la formarea rezultatului operaţiilor cu matrice, adunare, înmulţire etc., conducând totodată şi la mărirea duratei de realizare a acestor operaţii prin ocuparea procesorului cu adunări şi înmulţiri scalare cu zero. Acest inconvenient se manifestă cu atât mai pregnant cu cât dimensiunea matricei este mai mare.

Prin urmare, pentru probleme de dimensiuni mari, s-a căutat găsirea unor modalităţi de reprezentare compactă a matricelor rare, în care să se renunţe la memorarea elementelor nule. În acest caz este necesar ca tehnicile de memorare să încorporeze pe lângă elementele nenule şi mijloacele de identificare a poziţiilor acestor elemente în matrice.

Sunt prezentate în continuare câteva posibilităţi de memorare compactă a matricelor rare *MR*. Se face, de asemenea, o analiză a oportunităţii folosirii fiecărei tehnici în parte, în funcţie de densitatea matricei.

Memorarea prin identificare binară se bazează pe natura binară a sistemului de calcul, constând în memorarea numai a elementelor nenule ale

matricei într-o zonă primară *ZP* având tipul de bază corespunzător tipului elementelor matricei şi dimensiunea egală cu numărul elementelor nenule.

Structura matricei este indicată printr-o secvenţă binară memorată într-o zonă secundară *ZS*.

Matricea *A* prezentată anterior se memorează astfel:

Zona primară:

Locaţie	1	2	3	4
Valoare	1	-2	4	-1

Figura 6.1 *Structura ZP pentru matricea A*

Zona secundară:

Locaţie	1				5	6				10
Valoare	1	0	0	0	0	0	0	1	0	1
Locaţie	11				15	16				20
Valoare	0	0	0	0	0	0	1	0	0	0

Figura 6.2 *Structura ZS pentru matricea A*

Matricea *A* a fost memorată în ordinea liniilor, o altă posibilitate de memorare fiind în ordinea coloanelor. Pentru a reduce spaţiul ocupat de zona secundară se poate implementa soluţia dată de memorarea la nivel de bit a valorilor acesteia.

Dacă matricea *B* cu dimensiunea *(m, n)* are densitatea *G* şi dacă tipul de bază al matricei, respectiv tipul fiecăruia dintre elemente nenule ale matricei, este reprezentat printr-un cuvânt de *b* octeţi, atunci zona primară va necesita *m*n*G* cuvinte de *b* octeţi iar zona secundară *(m*n)/(8*b)* cuvinte. Numărul total de cuvinte necesare memorării matricei *B* prin intermediul celor două zone este

$$DMR_1 = m*n*G + (m*n)/(8*b) \quad (6.3)$$

Întrucât pentru memorarea matricei în forma clasică sunt necesare $DM = m*n$ cuvinte, raportul dintre cerinţele de memorie ale structurii de mai sus şi a celei standard este:

$$c_1 = \frac{DMR_1}{DM} = G + \frac{1}{8 \cdot b} \qquad (6.4)$$

În relaţia anterioară s-a considerat că memorarea zonei secundare se face la nivel de bit.

Considerând că elementele matricei *A* sunt reale şi se reprezintă pe 4 octeţi, rezultă:

$$c_{1A} = 0{,}2 + \frac{1}{32} = 0{,}23 \quad (6.5)$$

ceea ce indică că memorarea matricei *A* conform acestei structuri ocupă de aproximativ patru ori mai puţină memorie decât cea standard.

Egalând $c_1 = 1$ se determină limita superioară a densităţii unei matrice pentru care această structură necesită mai puţină memorie decât cea standard:

$$G_{\lim} = 1 - \frac{1}{8 \cdot b} \qquad (6.6)$$

Pentru matricea *A*:

$$G_{\lim} = 1 - \frac{1}{32} = 0{,}96 = 96\% \qquad (6.7)$$

Această structură de memorare diferă de abordări prin faptul că în zona secundară este alocată memorie şi pentru elementele nule ale matricei. Structura este mai puţin eficientă pentru matricele de mari dimensiuni foarte rare. Principala dificultate constă în complexitatea programelor de implementare a operaţiilor matriciale.

O altă modalitate de memorare prin identificare binară se obţine prin modificarea informaţiilor din zona secundară. Această zonă va conţine pe jumătăţi de cuvânt indicii de coloană a elementelor nenule din matrice, precum şi anumite informaţii de control pentru identificarea rapidă a poziţiei elementelor nenule în matrice. Structura *ZS* pe cuvinte este următoarea:

Tabelul nr. 6.1 *Structura ZS pe cuvinte*

Numărul cuvântului	**Jumătatea stângă**	**Jumătatea dreaptă**
1	Numărul de linii	Numărul de coloane
2	Numărul de elemente nenule	
3	Numărul de elemente nenule în linia 1	Numărul de elemente nenule în linia 2
4	Numărul de elemente nenule în linia 3	Numărul de elemente nenule în linia 4
…	…	…
k	Numărul de elemente nenule în linia m-1	Numărul de elemente nenule în linia m
k + 1	Indicele de coloană al primului element memorat	Indicele de coloană al celui de-al doilea element memorat
k + 2	Indicele de coloană al celui de-al treilea element	etc.

	memorat	
…	…	…
j	…	Indicele de coloană al ultimului element memorat

Pentru matricea *A*, zona secundară *ZS* are structura din figura 6.3.

Locaţie	1		2	3		4		5		6	
Valoare	4	5	4	1	2	0	1	1	3	5	2

Figura 6.3 *Structura ZS pentru matricea A*

În reprezentarea din figura 6.3 s-a considerat că elementele nenule sunt reprezentate pe 4 octeţi astfel că o jumătate de cuvânt în zona secundară se reprezintă pe 2 octeţi. Prin structura de memorare prezentată mai sus se memorează matrice a căror dimensiune maximă este de 9999 de linii sau coloane cu numărul maxim de elemente nenule memorate egal cu $10^8 - 1$. Se face observaţia că în cazul matricelor pătrate în primul cuvânt din *ZS* se va memora dimensiunea matricei.

Numărul total de cuvinte necesare zonei secundare este egal cu

$$(5+m+m*n*G)/2 \qquad (6.8)$$

valoarea fiind rotunjită la cel mai mare întreg. Numărul total de cuvinte necesar memorării unei matrice prin intermediul celor două zone *ZP* şi *ZS* este egal cu

$$DMR_2 = (5+m+3*m*n*G)/2 \qquad (6.9)$$

Raportul dintre cerinţele de memorie ale acestei structuri de identificare binară şi a celei standard este:

$$c_2 = \frac{3 \cdot G}{2} + \frac{5+m}{2*m*n} \qquad (6.10)$$

Pentru o matrice pătrată *(m=n)*, egalând c_2 = 1 şi trecând la limită pentru $m \to \infty$ rezultă valoarea maximă a densităţii unei matrice rare pentru care structura prezentată este eficientă:

$$G_{\lim_{m\to\infty}} \frac{2}{3}\left(1 - \frac{5+m}{2m^2}\right) = 0{,}666 = 66{,}6\% \qquad (6.11)$$

În relaţia anterioară se ajunge la acelaşi rezultat în cazul unei matrice nepătratică pentru care se trece la limită pentru $n \to \infty$ şi $m \to \infty$.

Pentru o matrice rară de dimensiune (100, 100), cu o medie de 66 elemente nenule pe linie, structura de mai sus necesită un total de 6600 + (5 + 100 + 6600)/2 = 9952 cuvinte, cu 0,6% mai puţin decât 10.000 cuvinte

necesare pentru memorarea standard. Întrucât densitatea elementelor nenule ale unei matrice rare este de obicei între 1% şi 3%. Structura se dovedeşte a fi deosebit de eficientă.

Memorarea compactă aleatoare constă în utilizarea unei zone primare *ZP*, conţinând numai elementele nenule ale matricei şi a două zone secundare conţinând indicii de linie şi de coloană corespunzătoare elementelor nenule.

Deoarece fiecare element nenul al matricei este identificat individual, este posibil ca matricea să fie memorată în ordine aleatoare. Matricea *A* se memorează astfel:

Locaţia	1	2	3	4
Valoare	1	-2	4	-1
Indice linie	1	2	2	4
Indice coloană	1	3	5	2

Figura 6.4 *Model de memorare compactă aleatoare a matricei A*

Avantajele memorării compacte aleatoare constau în faptul că noile elemente nenule ale matricei sunt adăugate la sfârşitul zonelor de memorare fără a afecta celelalte elemente, precum şi o manevrabilitate rapidă a datelor. În cazul matricelor simetrice această structură de memorare este simplificată prin memorarea numai a elementelor nenule de deasupra diagonalei principale, precum şi a elementelor nenule situate pe această diagonală.

Numărul total de cuvinte necesare memorării unei matrice de dimensiune *(m, n)* este în acest caz

$$DMR_3 = 3*m*n \qquad (6.12)$$

Raportul dintre cerinţele de memorie ale acestei structuri şi a celei standard este:

$$c_3 = 3 \cdot G \qquad (6.13)$$

Egalând relaţia anterioară cu unitatea se determină valoarea limită a densităţii matricei rare pentru care această structură este eficientă, $G_{\lim} = 33{,}3\%$.

În structura din figura 6.4, pentru identificarea elementelor nenule ale matricei rare au fost folosite două zone secundare corespunzătoare indicelui de linie şi de coloană. Se prezintă în continuare o altă posibilitate de memorare în care se va utiliza o singură zonă secundară de dimensiune egală cu numărul de elemente nenule ale matricei, conţinând simultan informaţii asupra indicilor de linie şi de coloană.

Astfel, fiecărui element din zona primară i se ataşează în zona secundară un număr întreg din care se determină indicii de linie şi de coloană. Dacă elementul $a_{ij} \neq 0$ este memorat în locaţia *k* a zonei primare atunci în zona secundară se va memora un indice agregat *ig* a cărui valoare este dată de relaţia

$$ig = i+(j-1)*n \qquad (6.14)$$

unde *n* este numărul de coloane a matricei. Acest număr este suficient pentru identificarea elementului în matrice.

Utilizând acest artificiu, matricea *A* se memorează astfel:

Locaţia	1	2	3	4
Valoare	1	-2	4	-1
Indice agregat, *ig*	1	12	22	9

Figura 6.5 *Model derivat de memorare compactă a matricei A*

Pentru a regăsi indicele de linie şi de coloană al oricărui element memorat în locaţia *k* se utilizează următoarea tehnică de calcul:

- coloana *j* este obţinută prin relaţia:

$$j \geq ig(k)/n \quad (6.15)$$

- linia *i* este determinată prin relaţia:

$$i = ig(k) - (j - 1) n \qquad (6.16)$$

Avantajul acestei structuri de memorare constă în faptul că necesită mai puţină memorie decât cea precedentă, fiind în schimb mai puţin rapidă în ce priveşte manevrarea datelor.

Numărul total de cuvinte necesar memorării matricei este

$$DMR_4 = 2*m*n*G \quad (6.17)$$

Raportul dintre cerinţele de memorie ale acestei structuri şi a celei standard este:

$$c_4 = 2G \qquad (6.18)$$

Valoarea limită a densităţii matricei pentru care această structură este eficientă este *G* = 50%.

Memorarea compactă sistematică presupune că elementele nenule ale unei matrice rare sunt memorate într-o anumită ordine, respectiv pe linii sau pe coloane. În acest caz nu este necesar să se memoreze în zonele secundare indicii de linie, respectiv de coloană. Pentru o memorare în ordinea liniilor, ne mai sunt necesari indicii de linie, însă se cere specificarea începutului fiecărei linii.

Şi în acest caz există mai multe structuri de memorare. Cea prezentată în continuare este caracterizată prin faptul că utilizează o singură zonă secundară *ZS*, care conţine indicii de coloană ale elementelor nenule din matricea considerată, precum şi elemente false care indică începutul fiecărei linii şi sfârşitul memorării întregii matrice. De exemplu, un element zero în *ZS*

marchează prezenţa unui element fals şi acesta specifică în *ZP* numărul liniei elementelor de la dreapta locaţiei. Sfârşitul matricei este marcat prin prezenţa în *ZP* a unui element fals cu valoarea zero.

Pentru matricea *A*, memorarea în această formă este următoarea:

Locaţia	1	2	3	4	5	6	7	8
ZP	1	1	2	-2	4	4	-1	0
ZS	0	1	0	3	5	0	2	0

Figura 6.6 *Model de memorare compactă sistematică a matricei A*

Pentru această structură de memorare numărul maxim de cuvinte necesar pentru a reţine o matrice rară de dimensiune *(m, n)* este

$$DMR_5 = 2*(m*n*r+m+1) \qquad (6.19)$$

Raportul de memorare este:

$$c_5 = 2*G + \frac{2*(m+1)}{m*n} \qquad (6.20)$$

Se constată că structura este eficientă pentru memorarea matricelor rare cu o densitate a elementelor nenule de maximum 50%.

Memorarea cu ajutorul listelor reprezintă o extensie a memorării compacte aleatoare. În timpul operaţiilor de inversare a matricelor rare, noi elemente nenule sunt continuu generate, iar altele sunt anulate şi deci structurile de memorare trebuie să fie capabile să execute aceste modificări într-un mod eficient. De aceea structurile de memorare bazate pe această tehnică sunt folosite pentru memorarea şi manipularea matricelor rare de mari dimensiuni.

Structura propusă utilizează o zonă principală *ZP* pentru memorarea elementelor nenule şi trei zone secundare:

ZSL – memorarea indicilor de linie ale elementelor nenule;
ZSC – indicii de coloană;
ZSU – memorarea adresei următorului element al matricei.

Matricea *A* se memorează după cum urmează:

Locaţia	1	2	3	4
ZP	1	-2	4	-1
ZSL	1	2	2	4
ZSC	1	3	5	2
ZSU	&2	&3	&4	NULL

Figura 6.7 *Model de memorare cu ajutorul listelor a matricei A*

unde prin *"&2"* se înţelege adresa celei de-a doua locaţii.

Raportul dintre cerinţele de memorare ale acestei structuri şi a celei standard este:

$$c_6 = 4*G \qquad (6.21)$$

Prin urmare această structură de memorare este eficientă pentru memorarea matricelor cu o densitate a elementelor nenule de maximum 25%.

6.3 Determinarea gradului de umplere al unei matrice rare

Pentru a deduce dacă o matrice este sau nu rară, se defineşte gradul de umplere al unei matrice, notat cu *p.* În cazul în care *p < 0,3*m*n*, se consideră că matricea este rară.

Problema matricelor rare comportă două abordări:

- abordarea statică, în care alocarea memoriei se efectuează în faza de compilare; aceasta presupune ca programatorul să cunoască cu o precizie bună numărul maxim al elementelor nenule;
- abordarea dinamică, în care alocarea se efectuează în timpul execuţiei, caz în care nu este necesară informaţia asupra numărului de elemente nenule; această abordare este dezvoltată în partea destinată listelor.

Memorarea elementelor matricei rare, presupune memorarea indicelui liniei, a indicelui coloanei şi, respectiv, valoarea nenulă a elementului.

Se consideră matricea:

$$A = \begin{bmatrix} 0 & 0 & 6 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 \\ 0 & 8 & 2 & 0 & 0 \end{bmatrix} \qquad (6.22)$$

Gradul de umplere al matricei *A* cu numărul de linii *m = 4*, numărul de coloane, *n= 5* şi numărul elementelor nenule *k = 5* este:

$$G = \frac{5}{5 \cdot 4} = 0.25 \qquad (6.23)$$

Se definesc 3 vectori:

lin [ ] – memorează poziţia liniei ce conţine elemente nenule;
col [ ] – memorează poziţia coloanei ce conţine elemente nenule;
val [ ] – memorează valoarea nenulă a elementelor.

Vectorii se iniţializează cu valorile:

Tabelul nr. 6.2 *Valorile iniţiale ale vectorilor LIN, COL şi VAL*

LIN	COL	VAL
1	3	6
2	1	7
3	4	9
4	2	8
4	3	2

Pentru efectuarea calculelor cu matrice rare definite în acest fel, un rol important îl au vectorii *LIN, COL*, iar pentru matricele rare rezultat se definesc vectori cu un număr de componente care să asigure şi stocarea noilor elemente ce apar.

Astfel, pentru adunarea matricelor rare definite prin:

Tabelul nr. 6.3 *Valorile matricei rare A*

LIN_A	COL_A	VAL_A
1	1	-4
2	2	7
4	4	8

şi

Tabelul nr. 6.4 *Valorile matricei rare B*

LIN_B	COL_B	VAL_B
1	1	4
2	2	-7
3	2	8
4	1	5
4	3	6

rezultatul final se stochează în vectorii:

Tabelul nr. 6.5 *Valorile matricei rare rezultat C*

LIN_C	COL_C	VAL_C
1	1	0
2	2	0
3	2	8
4	1	5
4	3	6
4	4	8
?	?	?
?	?	?

Vectorii *LIN_C, COL_C şi VAL_C* au un număr de componente definite, egal cu:

$$DIM\ (LIN\_A) + DIM\ (LIN\_A) \qquad (6.24)$$

unde *DIM()* este funcţia de extragere a dimensiunii unui masiv unidimensional:

Astfel, dacă:

$$int\ a[n\text{-}m]; \qquad (6.25)$$

atunci:

$$DIM\ (a) = n - m+1 \qquad (6.26)$$

Fiind abordată problematica matricelor rare, în mod natural se produce eliminarea elementelor nenule, obţinându-se în final:

Tabelul nr. 6.6 *Conţinutul final al matricei rare C*

LIN_C	COL_C	VAL_C
3	2	8
4	1	5
4	3	6
4	4	8
?	?	?
?	?	?
?	?	?
?	?	?

Prin secvenţe de program adecvate, se face diferenţa între definirea unui masiv bidimensional şi componentele iniţializate ale acestora, cu care se operează pentru rezolvarea unei probleme concrete. Din punct de vedere al nivelului de umplere, tabelul 6.6 descrie o matrice rară cu un grad de umplere egal cu

$$G = \frac{12}{32} * 100 = 37.5\ \% \qquad (6.27)$$

Situaţia evidenţiază ineficienţă în utilizarea spaţiului de memorie alocat.

De exemplu, vectorii *LIN_A* şi *LIN_B* au 3, respectiv 5 componente în utilizare, dar la definire au rezervate zone de memorie ce corespund pentru câte 10 elemente. Rezultă că vectorul *LIN_C* trebuie definit cu 20 componente încât să preia şi cazul în care elementele celor două matrice rare au poziţii disjuncte.

Din punct de vedere al criteriului minimizării spaţiului ocupat, această abordare nu este eficientă deoarece presupune în cele mai multe situaţii

alocarea de spaţiu care nu este utilizat. Pentru a atinge acest obiectiv, implementarea unei clase asociate matricei rare va defini vectori alocaţi dinamic, iar operaţiile aritmetice vor genera vectori rezultat cu grad de umplere egal cu 100%.

În cazul operaţiilor de înmulţire sau inversare, este posibil ca matricele rezultat să nu mai îndeplinească cerinţa de matrice rară.

În acest scop, se efectuează calculele cu matrice rezultat complet definite şi numai după efectuarea calculelor se analizează gradul de umplere şi dacă acesta este redus, se trece la reprezentarea matricei complete ca matrice rară.

Funcţiile *full( )* şi *rar( )*, au rolul de a efectua trecerea la matricea completă, respectiv la matricea rară.

Funcţia *full( )* conţine secvenţa:

```
for( i = 0; i < n; i++)
       a [LIN_a[i]] [COL_a[i]] = val_a[i];
```

ce descrie iniţializarea elementelor matricei pe baza valorilor din vectori, iar funcţia *rar( )* conţine secvenţa:

```
k =1;
for( i = 0; i < m; i++)
    for( j = 0; j < n; j++)
        if (a[i][j] != 0)
   {
           LIN_a[k] = i;
           COL_a[k] = j;
           val_a[k] = a[i][j];
           k = k + i;
   }
```

În cazul în care gradul de umplere nu este suficient de mic astfel încât matricea să fie considerată rară, pentru memorare se utilizează o structură arborescentă care conţine pe nivelul al doilea poziţiile elementelor nenule, iar pe nivelul al treilea valorile.

Astfel matricei:

$$A=\begin{bmatrix} 7 & 3 & 5 & 0 & 0 \\ 0 & 2 & 4 & 8 & 0 \\ 0 & 0 & 9 & 0 & 5 \\ 6 & 8 & 9 & 8 & 1 \end{bmatrix} \qquad (6.28)$$

îi corespunde reprezentarea din figura 6.8.

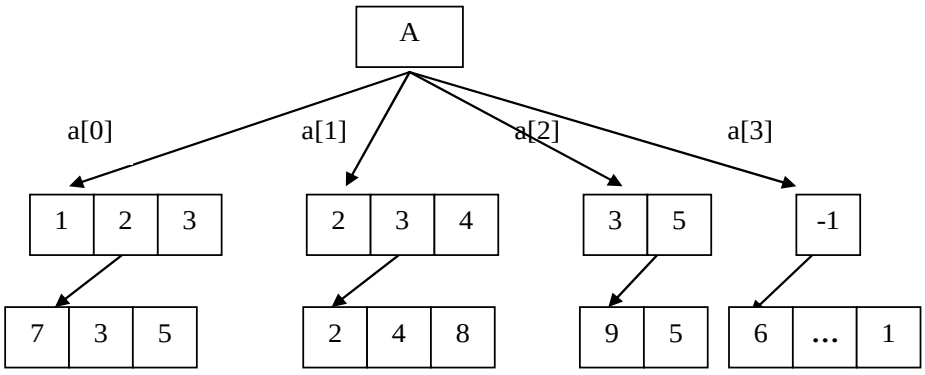

Figura 6.8 *Model grafic al matricei A*

Se elaborează convenţii asupra modului de stabilire a lungimii vectorului de poziţii, fie prin indicarea la început a numărului de componente iniţializate, fie prin definirea unui simbol terminal.

De asemenea. în cazul considerat s-a adoptat convenţia ca liniile complete să fie marcate cu simbolul *-1*, fără a mai specifica poziţiile elementelor nenule, care sunt de fapt termenii unei progresii aritmetice.

Liniarizarea masivelor bidimensionale conduce la ideea suprapunerii acestora peste vectori. Deci, punând în corespondenţă elementele unei matrice cu elementele unui vector, se pune problema transformării algoritmilor, în aşa fel încât operând cu elementele vectorilor să se obţină rezultate corecte pentru calcule matriceale.

Astfel, considerând matricea:

$$A=\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix} \quad (6.29)$$

prin punerea în corespondenţă cu elementele vectorului *b*, să se obţină interschimbul între două coloane oarecare k şi j ale matricei.

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{10}	a_{11}		a_{20}	a_{21}	a_{22}	a_{23}	a_{24}
1	2	3	4	5	6	7	…	11	12	13	14	15
b_0	b_1	b_2	b_3	b_4	b_5	b_6		b_{10}	b_{11}	b_{12}	b_{13}	b_{14}

Figura 6.9 *Punerea în corespondenţă a matricei A cu vectorul b*

Dacă matricea are *M* linii şi *N* coloane şi elemente de tip *int*, atunci adresa elementului *a[i][j]* este dată de relaţia

*adr(a[i][j]) = adr(a[0][0]) + ((i-0) * N+j) * 1g(int) (6.30)*

iar din modul în care se efectuează punerea în corespondenţă a matricei *A* cu vectorul *b*, rezultă:

$$adr(b[0]) = adr(a[0][0]) \qquad (6.31)$$

Pentru o matrice liniarizată, adresa elementului *a[i][j]* în cadrul vectorului este dată de relaţia

$$adr(a[i][j]) = adr(b[0]) + ((i-0) * N+j) * lg(int) = adr(b[(i-0) * N+j]) \qquad (6.32)$$

Dacă se consideră problema interschimbării valorilor coloanelor *j* şi *k* pentru o matrice liniarizată atunci secvenţa de înlocuire a coloanelor

```
for( i = 0; i < M; i++)
{
  c = a[i][j];
  a[i][j] = a[i][k];
  a[i][k] = c;
}
```

este înlocuită prin secvenţa:

```
for( i = 0; i < M; i++)
{
      c = b[(i-0) * N+j];
      b [(i-0) * N+j] = b[(i-0) * N+k];
      b[(i-0) * N+k] = c;
}
```

Transformarea algoritmilor de lucru cu masive bidimensionale în algoritmi de lucru cu masive unidimensionale este benefică deoarece nu se mai impune cerinţa de transmitere ca parametru a dimensiunii efective a numărului de linii, dacă liniarizarea se face pe coloane, respectiv a numărului de coloane, dacă liniarizarea se face pe linii.

În cazul matricelor rare, aceeaşi problemă revine la interschimbarea valorilor de pe coloana a treia dintre elementele corespondente ale coloanelor *k* şi *j* cu posibilitatea inserării unor perechi şi, respectiv, ştergerii altora.

Pentru generalizare, un masiv *n*-dimensional rar, este reprezentat prin *n + 1* vectori, fiecare permiţând identificarea coordonatelor elementului diferit de zero, iar ultimul stocând valoarea acestuia.

În cazul în care se construieşte o matrice booleană ce se asociază matricei rare, o dată cu comprimarea acesteia se dispun elementele nenule într-un vector. Punerea în corespondenţă a elementelor vectorului are loc în acelaşi moment cu decomprimarea matricei booleene şi analiza acesteia.

De exemplu, matricei :

$$A = \begin{bmatrix} 8 & 0 & 0 & 0 & 4 & 0 \\ 3 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 7 & 0 \\ 5 & 6 & 9 & 0 & 0 & 0 \end{bmatrix} \qquad (6.33)$$

se asociază matricea booleană:

$$B=\begin{bmatrix}1 & 0 & 0 & 0 & 1 & 0\\ 1 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 1 & 0\\ 1 & 1 & 1 & 0 & 0 & 0\end{bmatrix} \qquad (6.34)$$

care prin compactare, ocupă primii 3 octeţi ai unei descrieri, urmaţi de componentele vectorului:

$$C = (\ 8,\ 4,\ 3,\ 3,\ 1,\ 7,\ 5,\ 6,\ 9) \qquad (6.35)$$

Compactarea este procedeul care asociază un bit fiecărei cifre din forma liniarizată a matricei *B*.

6.4 Software orientat spre lucrul cu matrice rare

Metodele de calcul cu matrice rară pentru a fi eficiente trebuie să beneficieze de proporţia mare de elemente nule din aceste matrice, ceea ce creează necesitatea considerării unor tehnici speciale de memorare, programare şi analiză numerică.

O cerinţă esenţială în programarea matricelor rare constă în memorarea şi executarea operaţiilor numerice numai cu elementele nenule ale matricei, de a salva memorie şi timp de calcul. În acest caz memorarea standard, devenind ineficientă, este abandonată şi înlocuită cu metode de memorare adecvate, câteva dintre acestea fiind prezentate în paragraful anterior.

Un program de calcul cu matrice rare este cu atât mai eficient cu cât timpul de calcul şi cerinţele de memorie necesare sunt mai reduse faţă de acelea ale unui program tradiţional. De aceea, tehnica de programare trebuie să realizeze o proporţie convenabilă între timpul de calcul şi memoria utilizată, cerinţe care de cele mai multe ori sunt contradictorii. În general, este recunoscută necesitatea unei anumite de structuri de memorare a datelor şi o anumită tehnică de manipulare a acestora în cadrul unui algoritm în care sunt implicate matricele rare.

Principiul fundamental de programare cu matrice rare constă în memorarea şi manipularea numai a elementelor nenule, de sortare şi ordonare în structuri speciale în vederea *menţinerii structurii de matrice rară* şi a stabilităţii numerice, de evitare a buclelor complete.

În scopul ilustrării principalelor operaţii efectuate asupra matricelor rare s-a făcut implementarea acestora în C++, utilizând mediul de programare Visual C++. Pentru reprezentarea matricelor s-a ales memorarea compactă aleatoare, datorită flexibilităţii în manevrarea datelor. Este prezentată în continuare o parte a clasei *MatriceRara*, conţinând constructorii, destructorul, câteva dintre funcţiile şi operatorii implementaţi şi secţiunea privată.

```
class MatriceRara
{
/*******************************/
/*       Atribute              */
/*******************************/
private:
long dim;               //numarul de elemente nenule
int m,n;                //dimensiunea matricei
int * coloane;          //vectorul pentru index coloane
int * linii;            //vectorul pentru index linii
double * valori;        //vectorul pentru valori

/*******************************/
/* Constructor & Destructor    */
/*******************************/
public:
MatriceRara(void);
MatriceRara(const MatriceRara & MR);
MatriceRara(int M, int N, int D, double *val, int *lin, int *col);
MatriceRara(double **matrice, int M, int N);
virtual ~ MatriceRara( );

/*******************************/
/*    Metode auxiliare         */
/*******************************/
public:
bool EsteRara();
static MatriceRara Unitate(int);
double Urma();
double ** GetMatrice();

/*******************************/
/*    Metode de acces          */
/*******************************/
public:
inline int getDim();
inline int getLinii();
inline int getColoane();
inline double getValoareElement(int i);
inline int getColoanaElement(int i);
inline int getLinieElement(int i);

double getValoareElement(int i,int j);
bool setValoareElement(int i,int j, int valoare);
double operator ()(int i, int j);

friend ostream& operator <<(ostream&, MatriceRara &);
friend istream& operator >>(istream&, MatriceRara &);

/*******************************/
/*    Metode de prelucrare     */
/*******************************/

void Sortare();
MatriceRara operator =(MatriceRara &);
MatriceRara operator +(MatriceRara &);
MatriceRara operator -(MatriceRara &);
```

```
      MatriceRara operator *(MatriceRara &);
      MatriceRara operator *(double);
      MatriceRara operator !();
      MatriceRara Inversa();
      };
```

În cadrul secţiunii private sunt definite următoarele atribute:

m,n – dimensiunea matricei iniţiale;
dim – numărul de elemente nenule;
coloane – pointer la masive de întregi reprezentând coloana elementelor nenule;
linii – pointer la masive de întregi reprezentând linia elementelor nenule;
valori – pointer la un masiv având tipul de bază al elementelor matricei.

Aplicaţia informatică realizată vizează principalele operaţii necesare manipulării matricelor rare:

- construirea acestora prin introducerea datelor de la tastatură; acest obiectiv este atins prin supraîncărcarea operatorului >> prin rutina

```
istream& operator >>(istream& intrare, MatriceRara &MR)
{
      if(MR.dim)
      {
            delete[] MR.coloane;
            delete[] MR.linii;
            delete[] MR.valori;
      }
      cout<<"\n Numarul de linii ale matricei:";intrare>>MR.m;
      cout<<"\n Numarul de coloane ale matricei:";intrare>>MR.n;
      cout<<"\n Numarul de elemente nenule:";intrare>>MR.dim;
      MR.coloane = new int[MR.dim];
      MR.linii = new int[MR.dim];
      MR.valori = new double[MR.dim];
      for(int i=0;i<MR.dim;i++)
      {
            cout<<"\n Valoare a "<<i+1<<"-a este:";
            cout<<"\n\t Linia:";intrare>>MR.linii[i];
            cout<<"\n\t Coloana:";intrare>>MR.coloane[i];
            cout<<"\n\t Valoare:";intrare>>MR.valori[i];
      }
      return intrare;
}
```

- vizualizarea matricelor rare prin intermediul operatorului >>;

```
ostream& operator <<(ostream& iesire, MatriceRara & MR)
{
      iesire<<"\n Matricea rara de dimensiune ("<<
            MR.m<<","<<MR.n<<") este:";
      for(int k=0;k<MR.dim;k++)
            iesire<<"\n element["<<
            MR.linii[k]<<"]["<<MR.coloane[k]<<"] - "<<MR.valori[k];
```

```
	iesire<<"\n Vizualizare normala :\n";
	for(int i=0;i<MR.m;i++)
	{
		for(int j = 0;j<MR.n;j++)
			iesire<<"\t"<<MR.getValoareElement(i,j);
		iesire<<"\n";
	}
	return iesire;
}
```

- prin intermediul constructorilor clasei este posibilă crearea unei matrice rare iniţială fără valori, sau a unei matrice ce preia valorile dintr-o colecţie de trei vectori sau dintr-o matrice normală ce este validată

```
/*****************************/
/*  Constructori             */
/*****************************/
MatriceRara::MatriceRara(void):m(0),n(0),dim(0)
{
	coloane=NULL;
	linii=NULL;
	valori=NULL;
}

MatriceRara::MatriceRara(int M, int N, int D, double *val, int *lin, int
*col)
{
	m=M;
	n=N;
	dim = D;
	if(dim)
	{
		coloane = new int[dim];
		linii = new int[dim];
		valori = new double[dim];
		for(int i=0;i<dim;i++)
		{
			coloane[i] = col[i];
			linii[i] = lin[i];
			valori[i] = val[i];
		}
	}
	else
	{
		coloane = linii = NULL;
		valori = NULL;
	}

}
MatriceRara::MatriceRara(double **matrice, int M, int N)
{
	/*    validare matrice rara   */
	int nenule = 0;
```

```
      for(int i=0;i<M;i++)
            for(int j=0;j<N;j++)
                  if(matrice[i][j]) nenule++;
      if(((nenule*100)/(M*N))>100)
      {
            /*    matricea nu este rara   */
            coloane = linii = NULL;
            valori = NULL;
            m = n = dim = 0;
      }
      else
      {
            /*    matricea este rara      */
            coloane = new int[nenule];
            linii = new int[nenule];
            valori = new double[nenule];
            m = M;
            n = N;
            dim = nenule;

            int k=0;
            for(i=0;i<M;i++)
                  for(int j=0;j<N;j++)
                        if(matrice[i][j])
                        {
                              coloane[k]= j;
                              linii[k] = i;
                              valori[k] = matrice[i][j];
                              k++;
                        }
      }
}
```

- clasa permite copierea valorilor între diferite obiecte de tip *MatriceRara* prin intermediul constructorului de copiere şi a operatorului =;

```
/*****************************/
/*  Copy constructor         */
/*****************************/

MatriceRara::MatriceRara(const MatriceRara &MR)
{
      dim = MR.dim;
      m = MR.m;
      n = MR.n;

      if(dim)
      {
            coloane = new int[dim];
            linii = new int[dim];
            valori = new double[dim];
            for(int i=0;i<dim;i++)
            {
                  coloane[i] = MR.coloane[i];
                  linii[i] = MR.linii[i];
```

```
                  valori[i] = MR.valori[i];
            }
      }
      else
      {
            coloane = linii = NULL;
            valori = NULL;
      }
}

MatriceRara MatriceRara::operator =(MatriceRara & MR)
{
      if(dim)
      {
            delete[] coloane;
            delete[] linii;
            delete[] valori;
      }
      dim = MR.dim;
      m = MR.m;
      n = MR.n;
      if(dim)
      {
            coloane = new int[dim];
            linii = new int[dim];
            valori = new double[dim];
            for(int i=0;i<dim;i++)
            {
                  coloane[i] = MR.coloane[i];
                  linii[i] = MR.linii[i];
                  valori[i] = MR.valori[i];
            }
      }
      else
      {
            coloane = linii = NULL;
            valori = NULL;
      }
      return *this;
}
```

- pentru a asigura gestiunea eficientă a memoriei aplicaţiei se implementează destructorul clasei care asigură eliberarea memoriei rezervate de cele trei masiv de date;

```
/*****************************/
/*  Desstructor              */
/*****************************/
MatriceRara::~MatriceRara()
{
      delete[] coloane;
      delete[] linii;
      delete[] valori;
}
```

- principalele operaţii matriceale: adunarea, scăderea, transpunerea, înmulţirea şi inversarea.

Pe parcursul dezvoltării clasei *MatriceRara* s-a dovedit necesară implementarea unei funcţii *bool MatriceRara::EsteRara()*, care să verifice dacă o matrice este rară.

```
bool MatriceRara::EsteRara()
{
      if(((dim*100)/(n*m)>30)) return false;
      else return true;
}
```

În urma prelucrării matricelor şi prin generarea unor obiecte noi ca rezultate ale prelucrărilor aritmetice există situaţii în care matricea îşi pierde caracteristica de a fi rară. Pentru a implementa soluţii software eficiente, este indicat ca modul de stocare a matricei să fie ales în funcţie de nivelul de memorie ocupat. Prin validarea matricei rare cu metoda *EsteRara()*, datele pot fi stocate sub formă de matrice normală prin intermediul metodei *double ** GetMatrice()*

```
double** MatriceRara::GetMatrice()
{
double **matrice=NULL;
matrice = new double*[m];
for(int k=0;k<m;k++)
      matrice[k] = new double[n];
for(int i=0;i<m;i++)
      for(int j=0;j<n;j++)
      {
            matrice[i][j]=0;
      }
for(i=0;i<this->dim;i++)
      matrice[linii[i]][coloane[i]]=valori[i];
return matrice;
}
```

Produsul implementează metoda *void Sortare()* ce permite rearanjarea valorilor din matricea rară astfel încât acestea să corespundă unui mode de aranjare bazat pe parcurgerea pe linii a matricei. Această condiţie reprezintă o ipoteză de start în derularea operaţiilor aritmetice de adunare, scădere şi înmulţire deoarece contribuie la obţinerea unei metode de prelucrare mai eficiente din punctul de vedere al efortului procesor.

```
void MatriceRara::Sortare()
{
/*    metoda rearanjeaza elementele dupa linii  */
bool flag = true;
while(flag)
      {
      flag = false;
      for(int i=0;i<dim-1;i++)
```

```
                if(linii[i]>linii[i+1])
                {
                      int temp = linii[i];
                      linii[i] = linii[i+1];
                      linii[i+1] = temp;
                      temp = coloane[i];
                      coloane[i] = coloane[i+1];
                      coloane[i+1] = temp;
                      double valoaretemp = valori[i];
                      valori[i] = valori[i+1];
                      valori[i+1] = valoaretemp;
                      flag = true;
                }
                else
                      if(linii[i]==linii[i+1])
                            if(coloane[i]>coloane[i+1])
                            {
                                  int temp = coloane[i];
                                  coloane[i] = coloane[i+1];
                                  coloane[i+1] = temp;
                                  double valoaretemp = valori[i];
                                  valori[i] = valori[i+1];
                                  valori[i+1] = valoaretemp;
                                  flag = true;
                            }

          }
    }
```

Pentru a permite accesul programatorilor la atributele matricei rare, sunt implementate o serie de metode care returnează valorile acestor caracteristici.

```
    /*******************************/
    /*  Metode de acces            */
    /*******************************/
    int MatriceRara::getDim(){ return this->dim;}
    int MatriceRara::getLinii(){ return this->m;}
    int MatriceRara::getColoane(){ return this->n;}
    double MatriceRara::getValoareElement(int i){ return valori[i];}
    int MatriceRara::getColoanaElement(int i){ return coloane[i];}
    int MatriceRara::getLinieElement(int i){ return linii[i];}
```

Metodele de prelucrare a unei matrice sunt bazate pe algoritmi în care accesul la elementele matricei se realizează direct prin intermediul sintaxei *matricei[i][j]*. Din punct de vedere al structurii interne, modelul ales în clasa *MatriceRara* pentru implementarea unei matrice rare diferă de abordarea clasică a masivului bidimensional. Pentru a permite programatorilor, într-un mod transparent, accesul direct la elementele matricei se definesc metodele:

```
    double MatriceRara::getValoareElement(int i, int j)
    {
    for(int k=0;k<dim;k++)
          if((linii[k]==i)&&(coloane[k]==j)) return valori[k];
    return 0;
```

```
}

bool MatriceRara::setValoareElement(int i, int j, int valoare)
{
/*    metoda valideaza noua valoare */
if(valoare) return false;

for(int k=0;k<dim;k++)
      if((coloane[k]==i)&&(linii[k]==j))
      {
            valori[k]=valoare;
            return true;
      }
return false;
}

double MatriceRara::operator ()(int i, int j)
{
return this->getValoareElement(i,j);
}
```

În subcapitolele următoare se face o prezentare detaliată a operatorilor care implementează principalele operaţii matriceale: adunarea, scăderea, transpunerea, înmulţirea şi inversarea.

6.5 Adunarea, scăderea şi transpunerea

Prin prisma caracterului dinamic al modului de alocare a memoriei şi a caracteristicilor unice ale matricelor rare*, adunarea* acestor structuri de date presupune parcurgerea unei serii paşi:

- determinarea numărului de elemente nenule ale matricei sumă; din punctul de vedere al operanzilor sunt definite două situaţii de realizare a sumei, cu elemente comune şi cu elemente distincte; în cazul sumei a două elemente comune, se verifică dacă suma acestora este zero, caz în care rezultatul nu este reţinut în matricea rară generată;
- alocarea memoriei corespunzătoare acestui număr pentru cele trei masive unidimensionale;
- parcurgerea celor două matrice pe linii sau pe coloane şi determinarea sumei.

Prin *elemente comune* au fost desemnate valorile caracterizate prin indici de linie şi de coloană care sunt indentice în ambele matrice.

Pentru implementare s-a folosit suprascrierea operatorilor, tehnică ce oferă o mai mare putere de sugestie operaţiilor matriceale implementate. Este prezentat în continuare operatorul care implementează operaţia de adunare, structurată conform paşilor prezentaţi mai sus.

```
MatriceRara MatriceRara::operator +(MatriceRara & MR)
{
      /*    se determina dimensiunea matricei rezultat      */
```

```
//    se simuleaza suma si se contorizeaza numarul
//    de sume zero si numarul de sume nonzero

MatriceRara rezMR;

if((this->m!=MR.m)||(this->n!=MR.n))
      return rezMR;

int nrsz = 0, nrsnz = 0;
int i = 0, j = 0;
while((i<this->dim)&&(j<MR.dim))
{
      if(this->linii[i]<MR.linii[j])
            i++;
      else
            if(this->linii[i]>MR.linii[j])
                  j++;
            else
                  if(this->coloane[i]<MR.coloane[j])
                        i++;
                  else
                        if(this->coloane[i]>MR.coloane[j])
                              j++;
                        else
                              if(this->valori[i]+MR.valori[j])
                              {
                                    nrsnz++;
                                    i++;
                                    j++;
                              }
                              else
                              {
                                    nrsz++;
                                    i++;
                                    j++;
                              }
}

int rezdim = this->dim+MR.dim-nrsnz-2*nrsz;
rezMR.dim = rezdim;
rezMR.m = this->m;
rezMR.n = this->n;

rezMR.coloane = new int[rezdim];
rezMR.linii = new int[rezdim];
rezMR.valori = new double[rezdim];

//    se determina suma elementelor

int k=i=j=0;

while((i<this->dim)&&(j<MR.dim))
{
      if(this->linii[i]<MR.linii[j])
      {
            rezMR.linii[k] = this->linii[i];
```

```
                rezMR.coloane[k] = this->coloane[i];
                rezMR.valori[k] = this->valori[i];
                i++;
                k++;
          }

          else
                if(this->linii[i]>MR.linii[j])
                {
                      rezMR.linii[k] = MR.linii[j];
                      rezMR.coloane[k] = MR.coloane[j];
                      rezMR.valori[k] = MR.valori[j];
                      k++;
                      j++;
                }
                else
                      if(this->coloane[i]<MR.coloane[j])
                      {
                            rezMR.linii[k] = this->linii[i];
                            rezMR.coloane[k] = this->coloane[i];
                            rezMR.valori[k] = this->valori[i];
                            i++;
                            k++;
                      }
                      else
                            if(this->coloane[i]>MR.coloane[j])
                            {
                                  rezMR.linii[k] = MR.linii[j];
                                  rezMR.coloane[k] = MR.coloane[j];
                                  rezMR.valori[k] = MR.valori[j];
                                  k++;
                                  j++;
                            }
                            else
                                  if(this->valori[i]+MR.valori[j])
                                  {
                                  rezMR.linii[k] = MR.linii[j];
                                  rezMR.coloane[k] = MR.coloane[j];
                                  rezMR.valori[k]        =        this-
>valori[i]+MR.valori[j];
                                        k++;
                                        j++;
                                        i++;
                                  }
                                  else
                                  {
                                        i++;
                                        j++;
                                  }
     }
     if(i<this->dim)
          for(;i<dim;i++,k++)
          {
                rezMR.linii[k] = this->linii[i];
                rezMR.coloane[k] = this->coloane[i];
                rezMR.valori[k] = this->valori[i];
          }
```

```
	if(j<MR.dim)
		for(;j<MR.dim;j++,k++)
		{
			rezMR.linii[k] = MR.linii[j];
			rezMR.coloane[k] = MR.coloane[j];
			rezMR.valori[k] = MR.valori[j];
		}

	return rezMR;
}
```

Pentru a minimiza numărul de parcurgeri ale celor două matrice rare, în acest caz sunt necesare doar două parcurgeri, în metoda prezentată se porneşte de la ipoteza că matricea rară este generată prin parcurgerea pe linii a matricei iniţiale. Acest lucru asigură o ordine între elementele matricei rare şi permite identificare mai eficientă a elementelor comune. De asemenea, este implementată o parcurgere simultană a celor două matrice. Elementele curente din cele două matrice sunt analizate în ordine prin prisma valorii liniei şi a coloanei. În cazul în care elementele se găsesc pe linii diferite, elementul care are valoarea liniei mai mică este adăugat la rezultat şi se trece la următorul element din matricea respectivă. Dacă elementele curente din cele două matrice prezintă aceeaşi valoare pentru linie, atunci se compară valoarea coloanelor. Pentru elementele comune, se analizează rezultatul sumei şi se memorează doar valorile nenule.

Implementarea *operatorului de scădere* este absolut similară celui de adunare, singura diferenţă fiind aceea că în cazul elementelor comune se calculează diferenţa lor, în locul adunării. O altă abordare, este dată de utilizarea sumei, negând anterior valorile matricei ce se scade. Prin utilizarea operatorului MatriceRara MatriceRara::operator *(double valoare) ce permite înmulţirea matricei cu o valoare dată, scăderea se realizează prin supraîncărcarea operatorului -.

```
	MatriceRara MatriceRara::operator *(double valoare)
	{
	MatriceRara rezMR;
	if(valoare)
	{
	rezMR = *this;
	for(int i=0;i<rezMR.dim;i++)
		rezMR.valori[i]*=valoare;
	}
	return rezMR;
	}
	MatriceRara MatriceRara::operator -(MatriceRara &MR)
	{
	return ((*this)+(MR*-1));
	}
```

Transpunerea matricelor rare, prin intermediul operatorului !, este similară celei efectuate pe structură tablou, constând în inversarea indicilor de linie şi coloană între ei.

```
MatriceRara MatriceRara::operator !()
{
MatriceRara rezMR= *this;
/*    metoda transpune matricea     */
for(int i=0;i<dim;i++)
{
      int temp = rezMR.linii[i];
      rezMR.linii[i] = rezMR.coloane[i];
      rezMR.coloane[i] = temp;
}
return rezMR;
}
```

O altă metodă de a realiza transpunerea este dată de inversarea pointerilor pentru masivele de întregi reprezentând liniile, respectiv coloanele elementelor nenule

```
MatriceRara MatriceRara::operator !()
{
      MatriceRara rezMR= *this;
      int * temp = rezMR.coloane;
      rezMR.coloane = rezMR.linii;
      rezMR.linii = temp;
      return rezMR;
}
```

6.6 Înmulţirea şi inversarea matricelor rare

Pentru *înmulţirea* matricei rare *A*, *(m, l)* dimensională, cu matricea rară *B*, *(l, n)* dimensională, se utilizează procedura standard, având în vedere că metoda *getValoareElement* şi operatorul (i,j) permit accesul direct la elementele matricei rare.

Pentru a genera matricea rezultat, ca şi în cazul operaţiilor de adunare şi scădere, este nevoie să se determine, anterior efectuării produsului, numărul de elemente ale rezultatului. Acest lucru se realizează prin simularea produsului şi contorizarea numărului de valori nenule.

```
MatriceRara MatriceRara::operator *(MatriceRara &MR)
{
      MatriceRara rezMR;

      if(this->n!=MR.m)
            return rezMR;

      /*    se determina numarul de elemente ale rezultatului     */
      int rezdim=0;

      for(int i=0;i<this->m;i++)
            for(int j=0;j<MR.n;j++)
            {
                  double val = 0;
                  for(int k=0;k<this->n;k++)
```

```
                {
            val+=this->getValoareElement(i,k)*MR.getValoareElement(k,j);
                }
                if(val) rezdim++;
            }

      rezMR.dim = rezdim;
      rezMR.m = this->m;
      rezMR.n = MR.n;

      rezMR.coloane = new int[rezdim];
      rezMR.linii = new int[rezdim];
      rezMR.valori = new double[rezdim];

      int l = 0;

      for(i=0;i<this->m;i++)
            for(int j=0;j<MR.n;j++)
            {
                  double val = 0;
                  for(int k=0;k<this->n;k++)
                  {
            val+=this->getValoareElement(i,k)*MR.getValoareElement(k,j);
                  }
                  if(val)
                  {
                        rezMR.linii[l] = i;
                        rezMR.coloane[l] = j;
                        rezMR.valori[l] = val;
                        l++;
                  }
            }

      return rezMR;
}
```

Prin analiza acestui operator se constată că matricea rezultată păstrează structura de matrice rară.

Pentru implementarea *operatorului de inversare* s-a folosit algoritmul lui Krâlov. Acesta constă în parcurgerea unui număr de paşi egal cu dimensiunile matricei:

1: $A_1 = A$ $\quad p_1 = \frac{1}{1}\mathrm{tr}(A_1)$ $\quad B_1 = I - p_1 * A_1$

2: $A_2 = A * B_1$ $\quad p_2 = \frac{1}{2}\mathrm{tr}(A_2)$ $\quad B_2 = I - p_2 * A_2$

. . .

n-1: $A_{n-1} = A * B_{n-2}$ $\quad p_{n-1} = \frac{1}{n-1}\mathrm{tr}(A_{n-1})$ $\quad B_{n-1} = I - p_{n-1} * A_{n-1}$

n: $A_n = A * B_{n-1}$ $\quad p_n = \frac{1}{n}\mathrm{tr}(A)$ $\quad B_n = I - p_n * A_n$

Prin *tr(A)* se înţelege urma matricei *A*, suma elementelor diagonale, iar *I* reprezintă matricea unitate de aceeaşi dimensiune cu matricea *A*. Aceste elemente sunt implementate în clasa *MatriceRară* prin intermediul metodelor *Unitate(int)* şi *Urma()*.

```
MatriceRara MatriceRara::Unitate(int n)
{
MatriceRara rezMR;
if(n>0)
      {
      rezMR.n=rezMR.m=rezMR.dim = n;
      rezMR.coloane = new int[n];
      rezMR.linii = new int[n];
      rezMR.valori = new double[n];

      for(int i=0;i<n;i++)
      {
            rezMR.coloane[i] = i;
            rezMR.linii[i] = i;
            rezMR.valori[i] = 1;
      }

      }
return rezMR;
}

double MatriceRara::Urma()
{
double rez = 0;
if(this->m==this->n)
{
      for(int i=0;i<this->m;i++)
            rez+=this->getValoareElement(i,i);
}
return rez;
}
```

Krâlov a demonstrat că după parcurgerea celor *n* paşi, B_n este o matrice identic nulă. De aici rezultă inversa matricei *A*:

$$A^{-1} = p_n * B_{n-1} \qquad (6.36)$$

Se prezintă în continuare *operatorul de inversare* a matricelor rare care implementează algoritmul prezent.

```
MatriceRara MatriceRara::Inversa()
{
      MatriceRara tempMR, rezMR;

      MatriceRara unitateMR = MatriceRara::Unitate(this->m);

      MatriceRara initialaMR = *this;
```

```
      if(initialaMR.m==initialaMR.n)
      {
            double p = initialaMR.Urma() ;
            rezMR = initialaMR - (unitateMR*p);

            for(int k=2;k<initialaMR.m;k++)
            {
                  tempMR = initialaMR*rezMR;
                  p = (1.0/(double)k)*tempMR.Urma();
                  rezMR = tempMR - (unitateMR * p);
            }

            tempMR = initialaMR*rezMR;
            p = (1.0/(double)k)*tempMR.Urma();
            rezMR = rezMR*(1.0/p);

      }
      return rezMR;
}
```

Avantajele acestui algoritm constau în simplitatea implementării şi precizia rezultatelor, datorată folosirii unui număr redus de operaţii de împărţire.

6.7 Tipuri particulare de matrice rare

Există tipuri de matrice rare ce prezintă o serie de caracteristici prin prisma cărora se pot defini noi metode de a stoca valorile matricei.

O astfel de matrice rară este matricea bandă, în care valorile nenule sunt poziţionate în mijlocul liniei. În cazul matricelor rare bandă ce sunt pătratice, elemente utilizabile se grupează în jurul diagonalei principale sau secundare. De exemplu, matricea de dimensiune (5,8) din figura 6.10 este o matrice rară în care elementele nenule sunt grupate în jurul diagonalei.

9	**10**	0	0	0	0	0	0
0	**7**	**0**	**9**	0	0	0	0
0	0	**12**	**3**	0	0	0	0
0	0	0	**3**	**3**	0	0	0
0	0	0	0	0	**5**	**0**	**10**

Figura 6.10 *Matrice rară bandă*

Pe baza ipotezei că elementele nenule sunt grupate pe linii în zone de dimensiune redusă, se defineşte o nouă metodă de memorare a matricei bandă. Spre deosebire de abordarea compactă bazată pe cei trei vectori, în această situaţie minimizarea memoriei ocupate se realizează prin reducerea informaţiilor necesare localizării elementelor. Pentru fiecare linie se reţine indexul primei şi ultimei valori din grupul de valori nenule. Figura 6.11 descrie structura asociată matricei bandă

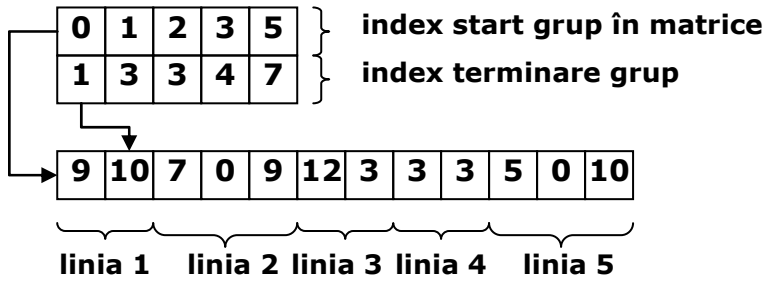

Figura 6.11 *Model de stocare a matricei bandă*

Se observă că pentru fiecare grup de valori nenule se reţine prin intermediul a doi vectori coloana primei valori nenule şi coloana ultimei valori nenule. Din acest motiv, vectorul de valori stochează şi valorile nule cuprinse în grup, fapt care conduce la pierderea eficienţei metodei pe măsură ce matricea bandă creşte în lăţime.

Pentru exemplul considerat, această metodă de stocare este mai eficientă decât modelul compact. Dacă se consideră valorile ca fiind întregi, nivelul de memorie necesar pentru datele matricei este egal cu

$$DMR_{banda} = DM_{index\_start} + DM_{index\_term} + DM_{valori} = (5 + 5 + 12)*4 = 88 \text{ octeţi} \quad (6.37)$$

unde:
DM_{index_start} – dimensiunea zonei de memorie asociată indexului de start;
DM_{index_term} – dimensiunea zonei de memorie asociată indexului de terminare;
DM_{valori} – dimensiunea zonei de memorie asociată valorilor;

Aceeaşi matrice stocată în forma compactă, necesită $DMR_{compact} = 3 * 12 * 4 = 144$ octeţi.

Din punctul de vedere al programatorului, această abordare conduce la definirea clasei *MatriceBanda*

```
class MatriceBanda
{
private:
int m,n;                //dimensiunea matricei
int dim;                //numarul de elemente nenule
int *index_start;       //vector pentru index start
int *index_term;        //vector pentru index terminare
double *valori;         //vector pentru valori

public:
MatriceBanda();
MatriceBanda(double **matrice, int m, int n);
MatriceBanda(const MatriceBanda&);
~MatriceBanda();

double getValoare(int i, int j);}
```

Se observă că pe lângă informaţiile descrise în figura 6.11 este necesar să se memoreze dimensiunea matricei bandă şi numărul de elemente nenule. Pentru a parcurge vectorii *index_start* şi *index_term* nu este nevoie de informaţii suplimentare, deoarece masivele au un număr de elemente egal cu numărul de linii ale matricei.

Pentru a asigura programatorilor un nivel de transparenţă la accesarea directă a valorilor din matricea bandă şi pentru a trece peste bariera dată de structura internă a obiectului *MatriceBanda* se defineşte metoda *double getValoare(int i, int j)* ce permite afişarea valorii elementului de pe linia *i* şi coloana *j*.

```
double MatriceBanda::getValoare(int i, int j)
{
      if((i<0 || i>=m) || (j<0||j>=n))
      {
            cerr<<"Index gresit !";
            return 0;
      }
      if((j<this->index_start[i])||(j>this->index_term[i]))
            return 0;
      else
      {
            int index_linie=0;
            for(int k =0;k<i;k++)
                  index_linie+=(index_term[i]-index_start[i]+1);
            return this->valori[index_linie+(j-index_start[i])];
      }
}
```

Un alt caz de matrice rară particulară este matricea diagonală. Acest masiv bidimensional este pătratic şi are elemente nenule doar pe diagonala principală. Dacă se consideră matricea din figura 6.12

9	0	0	0
0	**7**	0	0
0	0	**10**	0
0	0	0	**3**

Figura 6.12 *Matrice rară triunghiulară.*

se defineşte o metodă de reprezentare particulară ce se bazează pe memorarea dimensiunii matricei şi a valorilor de pe diagonala principală, clasa *MatriceDiagonala*.

```
class MatriceDiagonala
{
private:
      int n;
      int *valori;

public:
```

```
        MatriceDiagonala();
        MatriceDiagonala(MatriceDiagonala&);
        ~MatriceDiagonala();
        …
        double getValoare(int, int);
};
```

Pentru a accesa elementele matricei se implementează metoda *double getValoare(int, int)*.

```
double MatriceDiagonala::getValoare(int i, int j)
{
        if((i<0 || i>=n) || (j<0||j>=n))
        {
                cerr<<"Index gresit !";
                return 0;
        }
        if(i!=j)
                return 0;
        else
        {
                return this->valori[i];
        }
}
```

Metoda returnează valoare elementelor pentru care i este egal cu j, celelalte elemente ale matricei având valoarea zero.

Matricea diagonală este o matrice rară, deoarece o matrice pătratică de ordin n ≥ 4, conţine pe diagonala principală mai puţin de 30% din valori.

Prin prisma matricei diagonale se observă că matricea unitate, figura 6.13, reprezintă un caz special al acestui tip de matrice deoarece toate elementele de pe diagonală au valoarea 1. Pentru a memora o matrice unitate este nevoie să se indice doar ordinul matricei, aceasta putând fi generată cu uşurinţă.

Matricea triunghiulara reprezintă o matrice pătratică în care toate valorile aflate sub diagonala principală au valoarea 0. Pentru matricea triunghiulara numărul de elemente nenule este $\frac{(n-1)*n}{2}$, unde *n* reprezintă dimensiunea matricei pătratice, iar raportul acestora în mulţimea de elemente ale matricei, $\frac{(n-1)}{(n+1)}$ ia valori în intervalul [0,33 ; 1) pentru n ≥ 2. Cu toate că ponderea elementelor nenule nu este suficient de mică pentru a fi considerată o matrice rară, acest tip de matrice are în funcţie de rangul său un număr destul de mare de elemente nenule pentru a fi acordată o atenţie specială modului de stocare. De asemenea, proprietăţile algebrice ale matricei triunghiulare contribuie la alegerea acestui tip de matrice în rezolvarea sistemelor de ecuaţii compatibile cu acest tip de matrice:

- suma şi diferenţa dintre două matrice triunghiulare reprezintă tot o matrice triunghiulară;
- rezultatul înmulţirii a două matrice triunghiulare de dimensiune egală reprezintă tot o matrice triunghiulară;
- valoarea determinantului unei matrice triunghiulară este dat de produsul elementelor de pe diagonala principală.

De exemplu, matricea triunghiulară din figura 6.13

2	**9**	**3**	**3**
0	**7**	**1**	**5**
0	0	**10**	**2**
0	0	0	**3**

Figura 6.13 *Matrice triunghiulară*

este stocată, prin memorarea într-un masiv unidimensional a valorilor nenule şi prin indicarea indexului de început a valorilor de pe fiecare linie, figura 6.14.

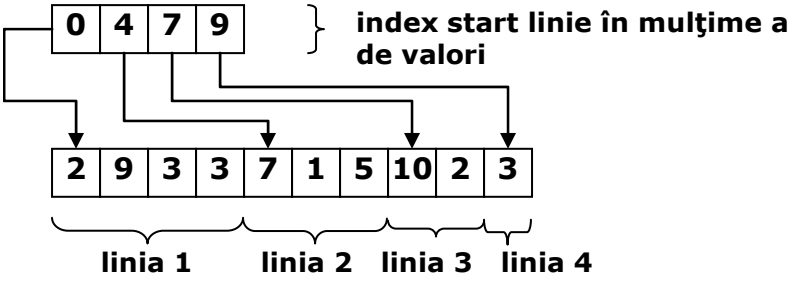

Figura 6.14 *Model de stocare a matricei triunghiulare*

Clasa *MatriceTriunghiulara* implementează această soluţie şi defineşte metode de acces direct la elementele matricei.

```
class MatriceTriunghiulara
{
private:
      int * linii;      //indexul fiecarei linii in lista de valori
      int n;            //dimensiunea matricei patratice
      double * valori;  //valorile nenule din matrice

public:
      MatriceTriunghiulara();
      MatriceTriunghiulara(int);
      virtual ~MatriceTriunghiulara();
      MatriceTriunghiulara(const MatriceTriunghiulara&);

      bool setValoare(int i, int j, double valoare);
      double getValoare(int i, int j);

      double getDeterminant();
      int getDimensiune(){return n;};
```

```
      MatriceTriunghiulara operator+(MatriceTriunghiulara& );
      MatriceTriunghiulara operator=(MatriceTriunghiulara& );

      friend ostream& operator <<(ostream&, MatriceTriunghiulara&);
};
```

Constructorul implicit iniţializează o matrice triunghiulară vidă, iar constructorul cu parametrii primeşte dimensiunea matricei pătratice. Pentru această abordare, valorile nenule sunt introduse de la tastatură de către utilizator parcurgând matricea pe linii.

```
MatriceTriunghiulara::MatriceTriunghiulara()
{
      linii = NULL;
      n = 0;
      valori = NULL;
}

MatriceTriunghiulara::MatriceTriunghiulara(int dim)
{
      if(dim)
      {
            this->n=dim;
            linii = new int[n];
            valori = new double[n*(n+1)/2];
            int indexLinie=0;
            for(int i=0;i<n;i++)
            {
                  linii[i]=indexLinie;
                  for(int j=i;j<n;j++)
                  {
                        cout<<"\n Element ["<<i<<"]["<<j<<"]:";
                        cin>>valori[indexLinie+j-i];
                  }
                  indexLinie+=n-i;
            }
      }
      else
      {
            n = 0;
            linii = NULL;
            valori = NULL;
      }
}
```

Constructorul de copiere al clasei şi metoda ce supraîncarcă operatorul = permit crearea de noi obiecte prin copierea valorilor unor matrice existent. Diferenţa dintre cele două metode este dată de situaţia în care se apelează fiecare metodă. Apelul operatorului = presupune existenţa ambelor obiecte şi programatorul trebuie să dezaloce zona de memorie a obiectului curent înainte de a face iniţializarea.

```
MatriceTriunghiulara::MatriceTriunghiulara (const MatriceTriunghiulara&
MT)
```

```
{
	if(MT.n)
	{
		this->n=MT.n;
		linii = new int[n];
		valori = new double[n*(n+1)/2];
		for(int i=0;i<n;i++)
			linii[i] = MT.linii[i];
		for(int j=0;j<n*(n+1)/2;j++)
			valori[j]=MT.valori[j];
	}
	else
	{
		n=0;
		linii = NULL;
		valori=NULL;
	}
}

MatriceTriunghiulara		MatriceTriunghiulara::operator		=
(MatriceTriunghiulara &MT)
{
	if(n)
	{
		delete[] linii;
		delete[] valori;
	}
	if(MT.n)
	{
		this->n=MT.n;
		linii = new int[n];
		valori = new double[n*(n+1)/2];
		for(int i=0;i<n;i++)
			linii[i] = MT.linii[i];
		for(int j=0;j<n*(n+1)/2;j++)
			valori[j]=MT.valori[j];
	}
	else
	{
		n=0;
		linii = NULL;
		valori=NULL;
	}
	return *this;
}
```

Destructorul clasei gestionează dezalocarea zonei de memorie rezervată de un obiect de tip *MatriceTriunghiulara* prin alocarea dinamică a spaţiului aferent celor două masive *linii* şi *valori*.

```
MatriceTriunghiulara::~MatriceTriunghiulara()
{
	delete[] linii;
	delete[] valori; }
```

Pentru a asigura accesul la elementele matricei, într-un mod transparent şi apropiat cu abordarea directă dată de sintaxa *matrice[i][j]*, clasa implementează metodele *getValoare* şi *setValoare* pentru a returna valoarea elementului de pe linia *i* şi coloana *j*, respectiv, pentru a iniţializa elementul.

```
double MatriceTriunghiulara::getValoare(int i, int j)
{
	if((i<0 || i>=n) || (j<0||j>=n))
	{
		cerr<<"Index gresit !";
		return 0;
	}
	if(j<i)
		return 0;
	else
		return this->valori[linii[i]+j-i];
}

bool MatriceTriunghiulara::setValoare(int i, int j,double valoare)
{
	if((i<0 || i>=n) || (j<0||j>=n))
		return false;
	if(j>=i)
	{
		valori[linii[i]+j-i]=valoare;
		return true;
	}
	else
		return false;
}
```

În cazul metodei *getValoare* se returnează valoarea zero pentru orice element pentru care valoarea *j* este mai mare decât *i* deoarece aceste elemente se găsesc sub diagonala principală.

Metoda *setValoare* validează coordonatele elementului de iniţializat deoarece nu este permisă setarea unui element al matricei ce se găseşte sub diagonala principală, caz în care matricea îşi pierde caracteristica de a fi triunghiulară.

Pornind de la ipoteza că suma a două matrice triunghiulare generează o matrice de acelaşi tip, metoda care implementează această operaţie aritmetică adună elementele de pe poziţii comune fără a lua în considerare rezultatul acestora şi fără a lua în considerare valorile de sub diagonală.

```
MatriceTriunghiulara	MatriceTriunghiulara::operator	+	(
MatriceTriunghiulara& MT)
{
	MatriceTriunghiulara rezMT;
	if(this->n==MT.n)
	{
		rezMT.linii = new int[n];
		rezMT.valori = new double[n];
		rezMT.n=this->n;
		for(int i=0;i<n;i++)
```

```
            {
                  rezMT.linii[i] = this->linii[i];
                  for(int j=0;j<n-i;j++)
                  {
      rezMT.valori[rezMT.linii[i]+j]=valori[linii[i]+j]+MT.valori[MT.lini
i[i]+j];
                  }
            }
      }
      return rezMT;
}
```

Pentru a calcula determinantul matricei triunghiulare, ipoteza de lucru este dată de faptul că elementele de pe diagonala principală reprezintă prima valoare de pe fiecare linie a matricei.

```
double MatriceTriunghiulara::getDeterminant(){
      double determinant = 1;
      for(int i=0;i<n;i++)
            determinant*=valori[linii[i]];
      return determinant;
}
```

Matricea de permutare este un masiv bidimensional în care fiecare linie sau coloană conţine o singură valoare unu, în rest elementele fiind nule. Matricea are această denumire deoarece este utilizată în operaţii algebrice pentru a permuta elementele unui vector conform unui model stabilit anterior. Dacă se consideră matricea MP din figura 6.15 şi vectorul X = 1,2,3,4,5

0	0	0	1	0
0	0	1	0	0
1	0	0	0	0
0	1	0	0	0
0	0	0	0	1

Figura 6.15 *Matrice de permutare de ordin egal cu cinci*

prin înmulţirea X * MP se obţine vectorul XP = 3,4,2,1,5, fapt ce evidenţiază că elementele vectorului au fost rearanjate conform poziţionării valorilor egale cu unu în matrice de permutare.

Pentru a stoca o astfel de matrice, în care *n* elemente sunt egale cu unu, restul fiind zero, se defineşte clasa *MatricePermutare*.

```
class MatricePermutare
{
private:
      int n;                        //ordinul matricei
      int* index_coloane;     //indexul coloanei
public:
      MatricePermutare();
      ~MatricePermutare();
```

```
        MatricePermutare(const MatricePermutare&);
        …
};
```

Valorile memorate pentru a reprezenta corect matricea de permutare sunt reprezentate de:

- rangul matricei pătratice;
- indexul coloanei pe care se găseşte valoarea unu de pe fiecare linie; deoarece există o singură valoare nenulă pe fiecare linie şi coloană, poziţia valorii în cadrul vectorului *index_coloane* indică linia corespondentă.

Figura 6.16 descrie vectorul index_coloane asociat matricei de permutare din figura 6.15.

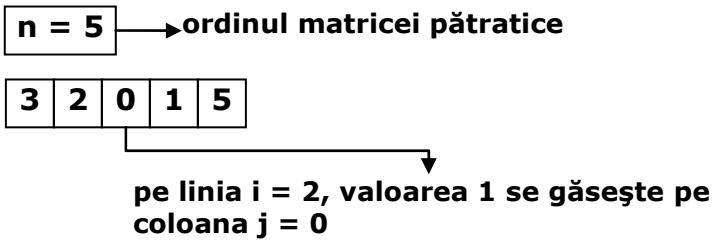

Figura 6.16 *Model de stocare a matricei de permutare.*

Într-o matrice de permutare de ordin *n*, numărul de valori nenule este egal cu *n*, iar în cazul în care această valoare depăşeşte nivelul trei, matricea este validată ca fiind o matrice rară datorită ponderii mici a valorilor nenule.

6.8 Estimarea parametrilor unei regresii statistice folosind clasa MR

Se consideră datele din tabelul 6.7 ce caracterizează cinci întreprinderi din punctul de vedere al productivităţii *y*, al numărului de utilaje de mare performanţă deţinute x_1 şi al veniturilor suplimentare acordate salariaţilor x_2. Se doreşte determinarea unei funcţii care să caracterizeze dependenţa dintre productivitate şi celelalte două variabile considerate independente.

Tabelul nr. 6.7 *Evoluţia indicatorilor y, x_1 şi x_2 într-o întreprindere*

y – productivitatea muncii (creşteri procentuale)	1	2	5	6	7
x_1 – utilaje de mare randament (bucăţi)	2	4	4	4	5
x_2 – venituri suplimentare (mil. lei)	1	1	3	5	5

Pentru a specifica forma funcţiei, se analizează pe cale grafică dependenţa variabilei *efect(y)* în raport cu fiecare dintre variabilele cauzale.

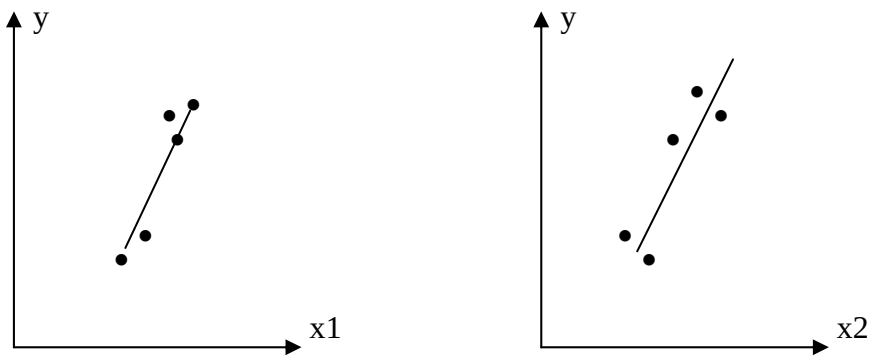

Figura 6.17 *Dependenţa y= f(x₁), y= f(x₂)*

Întrucât norul de puncte din fiecare reprezentare grafică sugerează o linie dreaptă, specificăm modelul astfel:

$$y_i = a_0 + a_1 x1_i + a_2 x2_i + u_i \qquad (6.38)$$

unde u_i reprezintă o variabilă "reziduală" ce caracterizează influenţa altor factori asupra variabilei efect *y*, factori care din diverse motive nu pot fi luaţi în considerare.

Dacă simbolizăm " $\bar{y}$ " valorile "ajustate", rezultate în urma aplicării modelului liniar, specificăm modelul astfel:

$$\bar{y}_i = \bar{a}_0 + \bar{a}_1 x_{1i} + \bar{a}_2 x_{2i} \qquad (6.39)$$

Relaţia anterioară se scrie pentru fiecare set de valori prezentate în tabelul 6.7, rezultând:

$$\begin{pmatrix} y_1 \\ y_2 \\ \Lambda \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x1_1 & x2_1 \\ 1 & x1_2 & x2_2 \\ \Lambda & \Lambda & \Lambda \\ 1 & x1_n & x2_n \end{pmatrix} \cdot \begin{pmatrix} \bar{a}_0 \\ \bar{a}_1 \\ \Lambda \\ \bar{a}_n \end{pmatrix} + \begin{pmatrix} u_1 \\ u_2 \\ \Lambda \\ u_n \end{pmatrix} \qquad (6.40)$$

Aşadar:

$$Y = XA + U \qquad (6.41)$$

iar

$$\bar{Y} = X\bar{A} \qquad (6.42)$$

Determinarea dependenţei dintre variabila efect şi variabilele cauză înseamnă determinarea valorilor numerice ale parametrilor $\bar{a}_0, \bar{a}_1$ şi $\bar{a}_2$. În acest

scop se utilizează metoda celor mai mici pătrate. Această metodă presupune minimizarea expresiei:

$$\sum_t u_t^2 \qquad (6.43)$$

adică, matriceal, U'*U. Dar:

$$U'U = \left(Y - X\overline{A}\right)'\left(Y - X\overline{A}\right) \quad (6.44)$$

unde prin U' s-a notat transpusa matricei U.

În [Peci94] se demonstrează că prin minimizarea relaţiei de mai sus se ajunge la expresia:

$$\overline{A} = \left(X'X\right)^{-1}X'Y \qquad (6.45)$$

O determinare cât mai precisă a matricei parametrilor unei regresii presupune existenţa unui număr foarte mare de date, adică un număr mare de linii în matricea *X*. În multe cazuri practice valorile acestor date sunt nule, fapt ce justifică implementarea relaţiei anterioare pe o structură de matrice rare. Având deja definiţi operatorii de transpunere, înmulţire şi inversare, implementarea relaţiei de mai sus presupune scrierea unei singure linii de cod:

$$\overline{A} = \left(\left(\left(!X\right)*X\right)\right).Inversa()*\left(!X\right)*Y \quad (6.46)$$

Aşadar, pentru aflarea matricei $\overline{A}$ sunt necesare două operaţii de transpunere, trei înmulţiri şi o inversare. Matricele *X* şi *Y* asupra cărora se operează au în multe dintre cazurile practice o densitate foarte mică, astfel că este pe deplin justificată folosirea unor structuri de memorare specifice matricelor rare.

Asistăm în prezent la un fenomen care tinde să atingă tot mai multe domenii de activitate. Necesitatea de a cunoaşte cât mai precis anumiţi factori sau parametri ce caracterizează domeniul respectiv, care până nu de mult erau fie consideraţi aleatori, fie nu se punea problema determinării lor, considerată a fi imposibilă. Cunoaşterea acestor factori oferă o descriere mai detaliată a sistemului în care se lucrează, permiţând în acest fel o mai bună activitate de control şi comandă a acestuia.

În cele mai multe dintre cazuri baza de calcul a acestor factori o constituie statistica matematică şi teoria probabilităţilor, ceea ce conduce la necesitatea rezolvării unor probleme liniare de foarte mari dimensiuni. Caracterul de raritate al structurilor implicate în rezolvarea problemelor, datorat caracteristicilor reale ale sistemelor, la care se adaugă necesitatea unei rezolvări rapide, în concordanţă cu dinamica crescândă a sistemelor actuale, justifică pe deplin introducerea în aplicaţiile informatice asociate a unor structuri de date adaptate acestor particularităţi.

Softul orientat spre lucrul cu matrice rare exploatează caracterul de raritate al structurilor manipulate, oferind un dublu avantaj: memorie şi timp. În ultimii ani memoria nu mai constituie o problemă, însă timpul necesar calculelor, odată cu apariţia sistemelor în timp real, se dovedeşte a fi tot mai mult o resursă critică.

Referitor la lucrul cu matrice în general, în cadrul unui sistem în care timpul reprezintă o resursă critică, există posibilitatea de a realiza un soft care să facă o evaluare anterioară calculelor asupra densităţii matricei, şi, în funcţie de aceasta, să decidă asupra structurilor de date ce vor fi folosite pentru memorare şi efectuarea calculelor, astfel încât timpul afectat calculelor să fie minim.