18. FIŞIERE

18.1 Structuri de date externe

Masivele, listele şi arborii binari, sunt structuri de date interne, prin faptul că se află în memoria internă a unui sistem de calcul. În momentul în care aceste structuri sunt stocate în memoria externă – pe discuri, benzi magnetice, dischete, compact-discuri acestea sunt numite *structuri de date externe*.

Memoria internă a unui calculator este imaginată ca un şir ordonat de baiţi. Fiecare bait se defineşte prin adresă şi prin conţinut. Memoria externă este imaginată, de asemenea, ca un şir de baiţi, cu deosebirea că pentru calculul adresei sunt luate în considerare elementele specifice modului de structurare a suportului.

Astfel, atunci când suportul este organizat în piste, cilindri şi problematica accesului este rezolvată prin poziţionarea pe aceste unităţi de acces, adresarea efectuându-se luând în calcul elemente structurale.

În plus, particularităţile de desfăşurare în plan fizic a operaţiilor de intrare/ieşire determină analiza distinctă a raportului dintre semnificaţia instrucţiunilor I/O din programe şi operaţiile efective de citire/scriere de pe suportul extern.

Presupunând că pentru un sistem de calcul modulele care realizează citiri/scrieri operează cu conţinutul unor buffere de lungime *L*, tot timpul aceste funcţii furnizează informaţii asupra începutului zonei ce este explorată, lungimea acesteia fiind rezultatul logicii de organizare a datelor.

Programatorul are acces direct sau indirect la zona de memorie tampon prin intermediul adresei sale de început sau prin intermediul unei alte zone de memorie definită în programul său, zonă în care este copiat conţinutul bufferului.

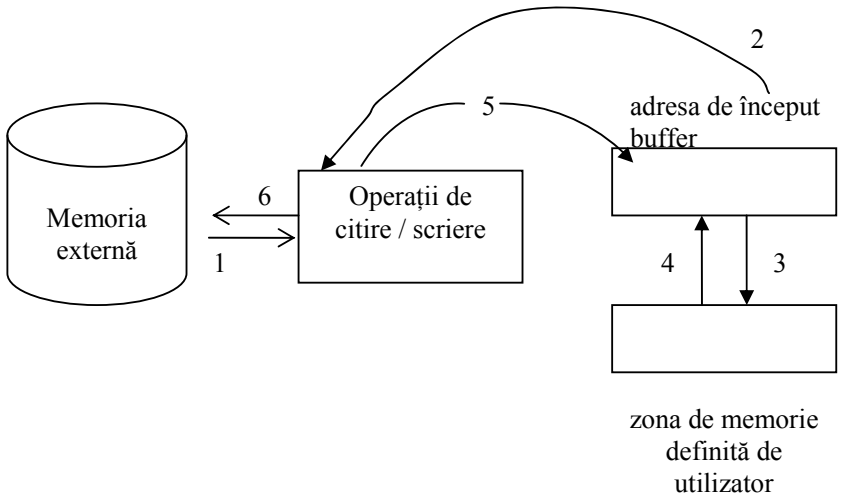


Figura 18.1 *Operaţii de intrare/ieşire*

Dinamica operaţiilor de intrare/ieşire determină actualizarea variabilei pointer care delimitează partea de început a subzonei din buffer ce este copiată în zona de memorie definită de utilizator.

În cazul funcţiilor de citire/scriere a unui caracter, variabila pointer se modifică cu o unitate, marcând exact schimbul de informaţii în dialogul om – calculator .

În cazul citirilor/scrierilor cu format delimitatorul acceptat este analizat, iar algoritmul de punere în corespondenţă este astfel proiectat încât acesta nu este integrat în setul informaţiilor.

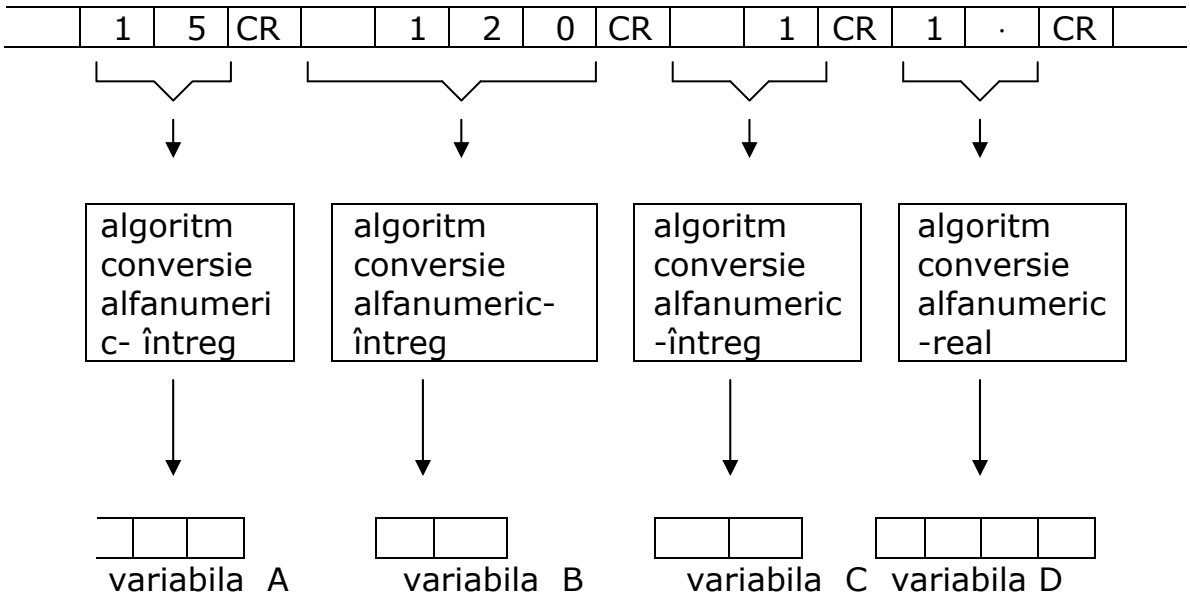


Figura 18.2 *Punerea în corespondenţă a valorilor citite*

Şirurile de caractere sunt delimitate prin <CR>, iar algoritmul de parcurgere a bufferului determină un astfel de conţinut al variabilei pointer încât este transmis ca parametru funcţiilor de conversie începutul fiecărei subzone de buffer care începe după <CR>.

Modul cum este concretizat <CR> ca delimitator de sfârşit de şir şi modul cum este definit descriptorul de format imprimă structurii de parametri ai funcţiilor de conversie anumite particularităţi.

Se observă că dinamica variabilei pointer este influenţată de tipul operaţiei de intrare/ieşire. Acest aspect explică de ce este necesară efectuarea avansului acestuia, când se alternează citiri cu format cu citiri fără format. La operaţii neomogene există modalităţi neomogene de definire şi de tratare a delimitatorilor de sfârşit de şir ca rezultat al activării tastei <CR>.

Ceea ce pare simplu în cazul structurilor de date interne, nu devine mai complicat în cazul structurilor de date externe, atât timp cât sunt clarificate chestiunile legate de variabilele pointer asociate bufferelor şi de faptul că o *citire fizică* efectivă nu înseamnă neapărat o *citire logică*, iar la scriere se întâmplă acelaşi lucru. Prin operaţia logică, înţelegem acţiunea ce corespunde unei apelări de funcţie citire/scriere din program.

De exemplu, pentru zona de lungime minimă *L = 256 octeţi*, ce este scrisă/citită la o singură operaţie fizică efectivă pe/de pe suport, la scrierea pe suportul extern a trei variabile de tip articol, *A, B* şi *C* cu:

lg(A) = 120 octeţi
lg(B) = 110 octeţi
lg(C) = 200 octeţi

variabila pointer permite preluarea datelor din structura *A*, se majorează cu 120, preia datele din structura *B* şi realizează o scriere fizică pe suport. Variabila pointer este apoi reiniţializată şi va prelua datele structurii *C* după care efectuează a doua scriere fizică. În acest caz celor trei *scrieri logice* le-au corespuns două *scrieri fizice*.

Există diferite modalităţi de realizare a operaţiilor de citire/scriere după cum se folosesc sau nu factori de blocare, se definesc sau nu elemente de regăsire a informaţiilor.

Structurile de date externe nu diferă mult de structurile de date interne. Apar unele complicaţii care nu sunt majore de altfel, prin aceea că volumul datelor este foarte mare şi elementele repetitive abundă, ceea ce conduce la ideea că structurile de date externe sunt privite ca structuri de structuri de date interne dispuse pe suport de memorie externă.

Pentru a realiza regăsirea într-un volum de date de dimensiuni remarcabile, este necesară organizarea, sistematizarea datelor şi crearea acelor elemente indispensabile localizării, adresării.

Structurile de date externe sunt contigue, formate din elemente dispuse în continuarea celorlalte şi necontigue, distanţa dintre elemente fiind o variabilă aleatoare a cărei lege de repartiţie este identificabilă, dar care necesită memorarea distanţelor, întrucât nu se construieşte un mecanism de generare cu repetare a acestora.

Structurile de date externe se regăsesc în cele mai multe cazuri sub denumirea de *fişiere*. Când ating un nivel de structurare prin adrese suficient de dezvoltat, se formează fişiere interdependente, iar în cazul unor structuri mai complexe se regăsesc sub denumirea de *baze de date*.

În cazul în care conţinutul de lungime *L* este tratat distinct, se ia în discuţie conceptul de *înregistrare fizică*. Se porneşte de la faptul că într-un buffer, de regulă sunt stocate datele ce corespund unei structuri de tip *articol*, ce se recunosc sub denumirea de *înregistrare logică* sau *articol logic*. Scopul este de a face deosebirea între modul în care se dispun informaţiile pe suportul fizic şi modul în care sunt gândite organizările de date în raport cu prelucrările particulare.

Introducerea factorilor de blocare vine să complice lucrurile, dar să îmbunătăţească indicele de folosire a zonelor de memorie.

Dacă:

$$L' = lg\ (structura\ de\ date\ de\ tip\ articol) < L \quad (18.1)$$

şi dacă există:

$$k = \left[\frac{L}{L'} \right] \qquad (18.2)$$

unde parantezele drepte înseamnă *partea întreagă* a expresiei, raportul *k* reprezintă o expresie mai simplificată a factorului de blocare.

De exemplu, dacă definim o structură de tip articol, ce conţine câmpuri ce conduc la o lungime de 80 octeţi şi *L = 256 octeţi*:

$$k = \left\lceil \frac{256}{80} \right\rceil = \lceil 3{,}2 \rceil = 3 \qquad (18.3)$$

În cazul în care *k' = 1*, pe cei 256 baiţi ai bufferului este încărcat un articol, ce este în fişier. Deci fişierul conţine în final *n* articole fizice şi tot *n* articole logice, gradul de încărcare cu informaţie utilă fiind:

$$g_1 = \frac{n*80}{n*256}*100 = \frac{10}{32}*100 \approx 30\% \qquad (18.4)$$

În cazul în care *k' = 2*:

$$g_2 = \frac{[[n/2]+m]*160}{n*256}*100 \qquad (18.5)$$

unde:

$$m = \begin{cases} 0, \text{ daca n e numar par} \\ 1, \text{ daca n e numar impar} \end{cases}$$

Pentru *k' = 3*:

$$g_3 = \frac{[[n/3]+m]*240}{n*256}*100 \qquad (18.6)$$

Se observă că:

$$g_3 \rangle = g_2 \rangle = g_1 \qquad (18.7)$$

Se vorbeşte de factorul de blocare optim care se stabileşte pentru fiecare tip de memorie externă şi lungime de structură de date de tip articol, ce urmează a fi memorată în fişier.

În continuare, luând în considerare numai aspectele care ţin de modul de stocare a informaţiei, strict dependentă de aplicaţia programatorului, se fac următoarele specificaţii:

- se consideră fişierul ca structură de date contiguă dacă informaţiile utile sunt dispuse unele în continuarea celorlalte, fără baiţi care să le separe;
- se consideră fişierul ca structură de date regulat necontiguă dacă între toate articolele sau între grupuri de articole, având număr fix, există baiţi nefolosiţi, în acelaşi număr; există posibilitatea de a construi o formulă de calcul a adresei articolului *k*, pornind de la adresa altui articol *j*;
- se consideră structuri de date necontigue fişierele ale căror articole sunt dispuse unele faţă de celelalte, la distanţe care sunt variabile aleatoare.

Trecerea de la memoria internă la memoria externă, ia în considerare modul de organizare al fiecărui suport. Dacă la nivelul memoriei interne aceasta este privită ca un vector, în cazul suporturilor externe de informaţie organizate pe piste baiţii sunt priviţi ca având dispunerea asemeni elementelor unei matrice. Linia indică pista pe care se află baitul, iar coloana indică poziţia baitului pe pistă.

Organizarea pe sectoare determină luarea în considerare a unei matrice tridimensionale. Pentru fiecare suport realizatorii pun la dispoziţie formulele de calcul ale adreselor cu luarea în considerare a elementelor de structură a suportului fizic.

Problema fragmentării informaţiilor determină stocarea de date necesare localizării părţii ce se continuă într-o altă zonă a suportului.

Pentru simplificarea prezentării se consideră un suport extern *S*, căruia i se asociază un model matriceal de dispunere a baiţilor. Baitul b_{ij} reprezintă baitul al *j*-lea, aflat pe pista *i* a suportului. Suportul are *n* piste, iar pe o pistă se află *m* baiţi.

Pentru început se presupune că baiţii au aceeaşi destinaţie, respectiv de a memora informaţii utile. Nu există baiţi care stochează informaţii privind structura suportului şi modul de ocupare a acestuia.

18.2 Criterii de clasificare a fişierelor

Şi în cazul clasificării fişierelor, asemenea datelor interne, există o multitudine de criterii, fiecare fişier fiind clasificat cu unul sau mai multe atribute ce corespund criteriilor de clasificare.

a) Criteriul lungimii articolelor ce alcătuiesc fişierul le împarte în:
   - fişiere cu articole de lungime fixă – sunt formate din elemente având aceeaşi lungime; fişierele sunt asemănătoare vectorilor ca structuri de date interne; elementele sunt omogene şi sunt dispuse unele în continuarea celorlalte;
   - fişierele cu articole de lungimi diferite, dar cunoscute – se consideră *m* tipuri de articole, având fiecare lungimea $l_1, l_2, \ldots, l_m$; fişierul conţine aceste elemente dispuse într-o anumită ordine sau în ordine oarecare; în ultima situaţie este necesară memorarea de informaţii care să permită identificarea tipurilor de articole şi lungimea acestora;
   - fişiere cu articole de lungime diferită, dar necunoscută – ceea ce se cunoaşte este legat de faptul că lungimile articolelor se află cuprinse între două limite: lungimile articolelor sunt variabile aleatoare, aparţinând unui interval definit; în mod obligatoriu, primul câmp conţine lungimea articolului.

b) Criteriul informaţiilor ce definesc regulile referitoare la dispunerea elementelor, împarte fişierele în:
   - fişiere având ca singur mod de dispunere poziţia articolelor;
   - fişiere cu elemente sortate după un câmp numit cheie a articolului, în funcţie de care se face reperarea în fişier.

c) Criteriul informaţiilor de localizare a articolelor care alcătuiesc fişierul;
   - fişiere care nu conţin informaţii asupra poziţiei articolelor – singura modalitate de a selecta un articol este parcurgerea tuturor articolelor care îl preced;

- fişiere care au definite zone ce conţin informaţii referitoare la adresele unor grupe şi subgrupe de articole – pentru a identifica un anumit element se localizează grupul şi apoi subgrupul de articole; o dată identificat subgrupul, selectarea elementului căutat este rezultatul parcurgerii articol de articol până la găsirea respectivului element;
- fişiere în care elementele conţin informaţii ce permit conturarea de liste înlănţuite sau arbori pe suporţi de memorie externă – complexitatea legăturilor dintre articolele fişierului determină un volum de informaţie privind adresele articolelor cu care un element intră într-o anumită relaţie.

d) Criteriul operaţiilor ce sunt efectuate de fişiere determină gruparea acestora în:
   - fişiere destinate scrierii datelor;
   - fişiere destinate citirii datelor;
   - fişiere destinate efectuării operaţiilor de actualizare.

   Oricare dintre fişierele unei grupe, în raport cu scopurile prelucrării îşi modifică atributele. De exemplu, un fişier care se creează este destinat scrierii datelor. La consultare acelaşi fişier aparţine grupei a doua, iar dacă înaintea consultării se efectuează actualizări, acelaşi fişier aparţine celei de a treia grupe.

e) Criteriul gestiunii fişierelor ia în considerare existenţa unui sistem de funcţii de prelucrare care asigură efectuarea operaţiilor specifice lucrului cu fişiere, numit sistem de gestiunea fişierelor. Acest criteriu împarte mulţimea fişierelor în:
   - fişiere care sunt prelucrate prin apelarea de funcţii ale unui sistem de gestiune – rezolvă întreaga problematică legată de organizarea şi accesul la date; funcţiilor sistemului de gestiune a fişierelor le corespund instrucţiuni; parametrii instrucţiunilor devin parametrii reali ai funcţiilor sistemului de gestiune; programatorul abordează întreaga problematică numai din punct de vedere logic al rezolvării problemei sale;
   - fişiere pentru care sunt definite funcţii primitive de realizare a operaţiilor elementare cu datele destinate suporturilor externe – programatorului îi revine sarcina să gestioneze totalitatea informaţiilor necesare regăsirii elementelor ce alcătuiesc fişierul; pentru programator, fişierul este o masă amorfă de baiţi, având conţinut şi poziţii; el efectuează transferuri de baiţi din zone de memorie spre suportul extern, indicând adrese acestor zone, lungimea zonei şi un mod de reparare a fişierului; există situaţii în care se efectuează lucru direct în bufferul asociat fişierului cu care se lucrează; în acest caz fişierul apare ca o resursă iar gestiunea acestei resurse este lăsată integral la dispoziţia programatorului;
   - fişiere care se construiesc şi se utilizează folosind instrucţiunile limbajului de asamblare – programatorul defineşte totalitatea condiţiilor pentru efectuarea operaţiilor cu fişiere; sunt definite şi încărcate etichetele care sunt scrise; se definesc bufferele şi se gestionează; se calculează adresele de pe suportul extern unde vor fi scrise datele; programatorul preia în programele sale tot ceea ce efectuează funcţiile primitive sau sistemul de gestiune a fişierelor; programul în care apar fişierele gestionate

de programator nu apelează alte funcţii decât cele scrise de acesta şi foloseşte numai instrucţiuni ale limbajului de asamblare, cel mult seturi de macroinstrucţiuni.

f) Criteriul organizării fişierelor, ia în considerare existenţa sau inexistenţa unor informaţii de regăsire a datelor, după cum urmează:
   - fişierele cu organizarea secvenţială - succesiune contiguă de articole fără existenţa unor elemente de informare, altele decât delimitatorii de început şi de sfârşit; dacă se ia în considerare similitudinea acestui mod de organizare cu înregistrarea pe o casetă a şlagărelor unei formaţii rock, avem imaginea clară a tuturor posibilităţilor de lucru cu fişierele secvenţiale; accesul la un şlagăr presupune audierea celor care-l preced; ştergerea unui şlagăr, altul decât cel de la sfârşit presupune existenţa a două casete; înregistrarea unui nou şlagăr, se face numai după ultimul şlagăr anterior;
   - fişierele însoţite de informaţii de tip index – presupun sortarea articolelor după chei şi împărţirea acestora în submulţimi; punerea în corespondenţă a elementelor din submulţimi cu adresele fizice determină realizarea într-un fişier a unei mulţimi de subfişiere secvenţiale; cu ajutorul indecşilor se identifică subfişierul secvenţial şi articolul căutat este preluat după ce a fost localizat prin parcurgerea articol după articol a subfişierului; operaţiile de actualizare, vizează ştergerea de articole, adăugarea de articole la sfârşitul fişierului sau subfişierelor, modificarea de câmpuri, rescrierea de articole şi inserarea de articole; subfişierele sunt masive unidimensionale contigue, formate din articole; ştergerea este o operaţie de dezactivare a elementelor, fără realizarea deplasării celorlalte elemente cu o poziţie spre stânga, deci fizic ştergerea unui articol nu se produce, operaţia fiind numai la nivel logic, în sensul că accesul la date este precedat de un test asupra caracterului activ sau neactiv al articolului; inserarea de articole, pentru a menţine criteriul ordonării elementelor care a determinat împărţirea mulţimii în submulţimi de articole, presupune realizarea de liste înlănţuite; articolul inserat este dispus într-o zonă disponibilă numită folcloric, zona de depăşire.
   - fişiere ale căror articole sunt dispuse aleator pe suport – cunoscându-se dimensiunile fişierului se alocă o zonă pe suportul extern printr-o operaţie numită preformare; datele sunt puse în corespondenţă printr-un procedeu oarecare cu adresele unde vor fi scrise; rezultă că procedeul de punere în corespondenţă este cu atât mai performant cu cât el realizează o dispunere mai uniformă a articolelor în zona rezervată; există posibilitatea ca folosind algoritmi de randomizare să se obţină elemente ce intră în calculul adresei fizice a începutului zonei din suportul extern în care se scrie un articol; pornind de la imposibilitatea să se genereze numere diferite care să îndeplinească condiţia de apartenenţă la subintervale, pe măsură ce se scriu articole în fişiere, are loc fărâmiţarea zonei rezervate; se construiesc funcţii pentru gestionarea articolelor care au aceeaşi adresă fizică prin calcul; aceste fişiere cu

organizarea aleatoare, permit efectuarea întregii game de operaţii, cu condiţia ca mecanismul de obţinere a adresei fizice să se menţină neschimbat; fişierul organizat aleator apare ca o structură necontiguă în care fiecare element este localizat pe baza unei formule, având ca parametru valoarea unui câmp ce intră în structura articolului, câmp numit cheia articolului; numeroasele lucrări care prezintă limbajul COBOL, redau imagini sugestive ale modului în care se implementează filozofia fiecărui mod de organizare a fişierelor, deci realitatea este alta, dacă se are în vedere organizarea matriceală a suportului şi modul static în multe cazuri de alocare a memoriei externe.

g) Criteriul suportului unde este stocat fişierul împarte fişierele în:

- fişiere pe cartele perforate;
- fişiere pe bandă perforată;
- fişiere pe bandă magnetică;
- fişiere pe suport optic;
- fişiere pe disc;
- fişiere pe dischete;
- fişier în memoria internă.

Prelucrările moderne, neîngrădite de restricţiile lipsei de memorie internă, au condus la citirea unui fişier ca un singur articol foarte mare şi prelucrarea acestuia în memorie. Logic apare o singură instrucţiune de citire, deci apelarea o singură dată a funcţiei în realitate au loc:

$$cf = \left[\frac{L_f}{L} \right] + m \qquad (18.8)$$

citiri fizice, unde:

- *cd* - citiri fizice din fişier;
- L_f - lungimea fişierului, dată în baiţi;
- *L* - lungimea unei înregistrări fizice la o citire;
- *m* - variabila booleană ce este 0, dacă Lf este divizibil prin L şi 1 în caz contrar.

h) Criteriul privind modul de efectuare a operaţiilor de intrare/ieşire grupează fişierele în:

- fişiere de tip *stream* în care datele la scriere sau la citire sunt câmpuri elementare sau alte tipuri structuri de date, constituite într-un şir, cu indicarea formatului după care se fac mai întâi convenţiile şi apoi se stabilesc lungimile şirurilor care vor fi scrise pe suport; fişierul apare ca o succesiune de elemente oarecare ale căror poziţii sunt deduse, dacă se iau în calcul informaţiile pe care le oferă descriptorii de format şi locul pe care fiecare element îl ocupă în lista de parametri ai funcţiei apelate la scriere; este de dorit ca aceleaşi liste de variabile de la scriere să fie utilizate şi la apelul funcţiilor de citire, iar descriptorii de format să se menţină aceiaşi pentru a oferi succes prelucrărilor;
- fişiere de tip *record* în care datele sunt caracterizate prin lungime şi adresă de început; articolele au lungime fixă sau variabilitatea lungimilor este în cadrul unei mulţimi formate din câteva elemente; operaţiile de lucru cu fişierele de acest tip nu

sunt în general precedate sau urmate de conversii; operaţiile ce preced calculele sunt lăsate la dispoziţia programatorului;

De exemplu, fişierul stocurilor de materiale este:

- un fişier de tip record;
- are organizare indexată;
- se află pe disc;
- este gestionat cu funcţiile unui sistem de gestiune a fişierelor;
- are articole de lungime fixă;
- se efectuează toate operaţiile de actualizare pe el;
- conţine informaţii asupra poziţiei anumitor articole;
- articolele sunt identificate după codul materialului care joacă rol de cheie de articol;
- fiecare material are o cheie unică;
- fişierul este sortat crescător.

Astfel, un fişier oarecare este caracterizat înscriind oricare dintre informaţiile ce rezultă din criteriile specificate.

18.3 Fişiere secvenţiale

Fie mulţimea de elemente E_1, E_2, …, E_n oarecare ce corespund structurilor de date SD_1, SD_2, …, SD_n definite într-un program *P*.

Elementele E_i, i = 1, 2, …, n, sunt şiruri de baiţi caracterizate printr-un conţinut rezultat din iniţializarea structurilor de date SD_i, i = 1, 2, …, n. Pe un suport extern se alocă elementelor E_1, E_2, …, E_n zone de memorie de lungime:

$$\lg(SD_i)+\lg(\alpha) \qquad (18.9)$$

fişierul având în final lungimea:

$$L_f = n*\lg(\alpha)+\sum_{i=1}^{n}\lg(SD_i) \qquad (18.10)$$

De obicei, $lg(a) = 2$. Cei doi baiţi ataşaţi fiecărui element vor conţine lungimea articolului:

$$cont(\alpha)=\lg(SD_i) \qquad (18.11)$$

Fişierul este caracterizat printr-o adresă a articolelor. Astfel:

$$adr(E_1)=A+\lg(\alpha) \qquad (18.12)$$

unde, *A* reprezintă adresa fizică a primului bait a primului articol căruia i s-a ataşat în faţă $lg(a)$ baiţi pentru a memora lungimea articolului respectiv:

$$adr(E_i)=A+\sum_{j=1}^{i-1}\left(cont(\beta_j)\right)+\lg(\beta_i) \qquad (18.13)$$

sau:

$$adr(E_i) = A + \sum_{j=1}^{i-1} \lg(SD_i) + i * \lg(\alpha) \qquad (18.14)$$

unde, β_k reprezintă grupul de baiţi de lungime *lg(a)* ataşat elementului E_k în fişier. Dacă:

$$\lg(SD_1) = \lg(SD_2) = \ldots = \lg(SD_n) \qquad (18.15)$$

atunci:

$$cont(\beta_1) = cont(\beta_2) = \ldots = cont(\beta_n) \qquad (18.16)$$

caz în care, în eticheta de început a fişierului se specifică tipul articolului *lungime fixă*. De asemenea, se memorează şi lungimea, iar $lg(\beta_k)$ devine zero.

Adresa unui element oarecare E_i, este obţinută prin relaţia:

$$adr(E_i) = A + \sum_{j=1}^{i-1} \lg(SD_j) \qquad (18.17)$$

Cu fişierele secvenţiale se efectuează următoarele proceduri:

- crearea unui fişier secvenţial constă în dispunerea elementelor E_1, E_2, …, E_n pe suport în această ordine;
- consultarea integrală sau parţială a fişierului baleiază din aproape în aproape elementele fişierului şi se prelucrează cele care prezintă interes;
- adăugarea elementului E_m se efectuează la adresa:

$$adr(E_m) = adr(E_n) + \lg(S_n) + \lg(\alpha) \qquad (18.18)$$

  ceea ce pune în evidenţă că adăugarea se face numai la sfârşitul fişierului;
- interclasarea a două fişiere;
- sortarea fişierelor, care constă în obţinerea unui fişier în care articolele au o astfel de dispunere încât pentru un câmp *x* există relaţia:

$$cont\ (E_1.x) > cont\ (E_2.x) > \ldots > cont\ (E_n.x) \qquad (18.19)$$

dacă sortarea s-a făcut descrescător, sau:

$$cont\ (E_1.x) < cont\ (E_2.x) < \ldots < cont\ (E_n.x) \qquad (18.20)$$

dacă sortarea s-a făcut crescător.

Întrucât se lucrează în cele mai multe cazuri cu fişierul în totalitatea lui, nu este justificată memorarea de informaţii pentru anumite articole.

Explorarea fişierelor secvenţiale corespunde unei structuri de date contigue, asemeni unui vector de structură sau a unei înşiruiri de diferite structuri, cu posibilitatea calculului adresei unui element oarecare:

$$adr\ (suc\ (E_i)) = adr\ (E_i) + lg\ (SD_i) + lg\ (\alpha) \qquad (18.21)$$

$$adr\ (pred\ (E_i)) = adr\ (E_i) - lg\ (SD_i\text{-}1) - lg\ (\alpha) \qquad (18.22)$$

Se spune că fişierul este închis dacă:

$$\lim_{m\to\infty} suc^m(E_i) = E_n \qquad (18.23)$$

Se spune că fişierul este deschis dacă:

$$\lim_{m\to\infty} pred^m(E_i) = E_1 \qquad (18.24)$$

O astfel de abordare determină continuarea prelucrării chiar dacă există tentative de a închide un fişier deja închis sau de a deschide un fişier deja deschis.

Apare problema privind parametrii funcţiei de deschidere. Dacă sunt luaţi în considerare parametrii primei deschideri, problema este rezolvată, tentativele de deschidere a unor fişiere deja deschise fiind inefective.

Întrucât există formule de calcul pentru adresele fiecărui element din submulţimea $E_1, E_2, \ldots, E_n$ nu se justifică construirea unui vector a adreselor în fişier $a_1, a_2, \ldots, a_n$ pentru aceste elemente.

Sistemele de operare evoluate gestionează închiderea fişierelor la închiderea execuţiei programelor.

18.4 Fişiere secvenţial – indexate

Se consideră o mulţime de elemente $E_1, E_2, \ldots, E_n$ generate după structura *SD* şi un câmp *x* astfel încât:

$$cont\ (E_1.x) < cont\ (E_2.x) < \ldots < cont\ (E_n.x) \qquad (18.25)$$

deci, şirul elementelor este sortat crescător după câmpul *x*, care joacă rol de cheie a articolelor în structura *SD* de generare.

Elementele de sortare vor fi memorate, fiecare ocupând o zonă de lungime:

$$\lg(SD) + \lg(\gamma) \qquad (18.26)$$

unde, γ reprezintă o zonă în care este memorată o adresă fizică de pe suport, ţinând seama de structurarea contiguă a suportului.

Fişierul are un fond informaţional de lungime:

$$L_f = n * [\lg(\gamma) + \lg(SD)] \qquad (18.27)$$

Se construieşte mulţimea perechilor:

$$(cont(E_i.x).a_i) \qquad (18.28)$$

a cheilor şi adreselor articolelor ce intră în componenţa fişierului. Se calculează:

$$\sigma^2 = \frac{\sum_{i=1}^{n}\left(cont(E_i.x) - \bar{x}\right)}{n} \qquad (18.29)$$

şi rezultă o împrăştiere suficient de mare care să reducă şansele găsirii unei formule de calcul a adresei fizice a unui articol, folosind strict ca informaţie cheia acestuia.

Dacă se acceptă ideea utilizării unui arbore binar cu 4 noduri terminale, mulţimea elementelor $E_1, E_2, \ldots, E_n$ este împărţită în 4 subşiruri, având un număr aproape identic de elemente, reţinând adrese şi cheile elementelor ce delimitează fiecare subşir de articole:

$$(cont(E_{kj}.x), a_{kj}), \quad j= 1,\ 2,\ 3,\ 4;\ k = 1,\ 2, \ldots,\ n \qquad (18.30)$$

unde $k_1 < k_2 < k_3 < k_4$.

Perechile de delimitare ale începutului de subşir, se obţin astfel:

- pentru primul subşir:

$$Q_1 = \left(cont(E_1.x), a_1\right) \qquad (18.31)$$

- pentru al doilea subşir:

$$Q_2 = \left(cont(E_{1+m}.x), a_{m+1}\right) \qquad (18.32)$$

- pentru al treilea subşir:

$$Q_3 = \left(cont(E_{1+2m}.x), a_{2m+1}\right) \qquad (18.33)$$

- pentru al patrulea subşir:

$$Q_4 = \left(cont(E_{1+3m}.x), a_{3m+1}\right) \qquad (18.34)$$

Perechile pentru delimitarea sfârşitului pentru fiecare din cele 4 subşiruri sunt:

$$\begin{aligned} P_1 &= \left(cont(E_m.x), a_m\right) \\ P_2 &= \left(cont(E_{2m}.x), a_{2m}\right) \\ P_3 &= \left(cont(E_{3m}.x), a_{3m}\right) \\ P_4 &= \left(cont(E_n.x), a_n\right) \end{aligned} \qquad (18.35)$$

S-a considerat că fiecare subşir are *m* elemente, cu excepţia ultimului şir care are 1, 2 sau 3 elemente mai puţine.

Setului de date *i* se asociază arborele binar:

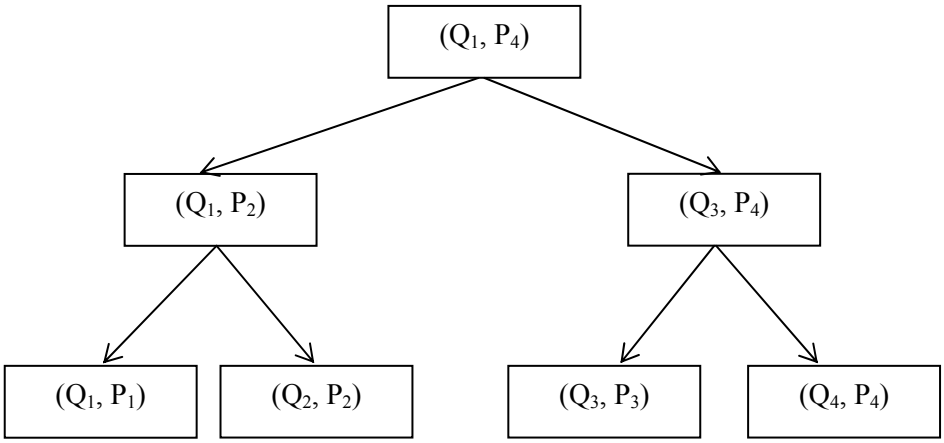


Figura 18.3 *Structura setului de date i*

Dacă, de exemplu, fişierul sortat este organizat secvenţial şi se doreşte citirea articolului penultim, care are cheia *7233*, în mod normal trebuie să fie citite cele *n-2* articole care îl preced. Dacă însă fişierul este înzestrat cu această informaţie pe care o conţine arborele binar cu 4 noduri terminale, inspectarea nodului rădăcină permite vizualizarea faptului că articolul căutat cu cheia *7233* este în fişier, adică:

$$cont\ (E_1.x) \leq 7233 \leq cont\ (E_n.x) \qquad (18.36)$$

Întrucât:

$$7233 > \frac{\text{cont}\,(E_n.x) + \text{cont}\,(E_i.x)}{2} \qquad (18.37)$$

rezultă că se parcurge pe nivelul inferior, nodul din dreapta.

Întrucât:

$$7233 > \frac{\text{cont}\,(E_{2n+1}.x) + \text{cont}\,(E_n.x)}{2} \qquad (18.38)$$

rezultă că se parcurge pe nivelul inferior, nodul din dreapta.

Odată ajungând pe ultimul nivel al arborescenţei, se reţin adresele a_{3m+1} şi a_n şi se baleiază secvenţial subfişierul delimitat astfel. Deci, se vor baleia mai puţin ¼ din totalul elementelor fişierului.

Pentru construirea arborilor binari asociaţi există numeroşi algoritmi de partiţionare a fondului informaţional. Important este ca numărul căutărilor secvenţiale să fie cât mai redus. Trebuie realizat un echilibru între numărul de nivele ale arborelui binar şi numărul subfişierelor, deoarece creşterea numărului de subfişiere conduce la creşterea duratei de acces la acestea din cauza parcurgerii unui număr prea mare de noduri în arborele binar.

În cazul în care se doreşte inserarea unui articol E_j într-un astfel de fişier se identifică poziţia lui, astfel încât:

$$cont\ (E_k.x) < cont\ (E_j.x) < \ldots < cont\ (E_{k+1}.x) \qquad (18.39)$$

În acest caz, articolul ca atare este memorat într-o zonă rezervată a fişierului, legătura cu celelalte articole efectuându-se prin:

$$\begin{aligned} cont(\gamma_k) &= adr(E_j) \\ cont(\gamma_j) &= adr(E_{k+1}) \end{aligned} \qquad (18.40)$$

ce corespunde unei inserări de elemente într-o listă.

La un număr mare de inserări, parcurgerea listelor reduce performanţa programului de exploatare a fişierului secvenţial – indexat. Acestea justifică trecerea la reorganizarea fişierului. Prin reorganizare se înţelege crearea unui nou fişier, în care elementele se dispun într-o aceeaşi zonă fără a mai exista informaţii de legătură, ca în cazul în care ar fi dispersate pe suport.

La reorganizare, se construieşte un nou binar al indecşilor. Utilizarea arborilor binari are aici numai un caracter exemplificativ. Tipul de arbore depinde în principal de împrăştierea cheilor în intervalul pe care sunt definite. Informaţiile privind arborele asociat tabelei de indecşi se stochează pe suport şi face parte din fişier.

Pentru o parcurgere mai rapidă, în cazul în care este posibil, informaţiile aferente structurii arborescente a indecşilor se încarcă în memoria internă şi se lucrează cu ele folosind funcţiile de parcurgere ale unui arbore.

În limbajele precum C şi C++, fiecare programator implementează algoritmi proprii pentru organizarea secvenţial – indexată, iar prin comparaţia comportamentului lor statistic cu alţi algoritmi existenţi, sunt dezvoltaţi sau abandonaţi.

18.5 Fişiere aleatoare

Sunt acele fişiere care ocupă o anumită zonă din memoria externă şi ale căror elemente nu sunt reperate decât prin adresa fizică pe care o au pe suportul extern.

Întrucât abordarea fişierelor *random*, depăşeşte prin amploarea fundamentării acest cadru, se consideră oportună prezentarea unui caz particular de fişiere ale căror adrese ale elementelor se calculează în funcţie de poziţia pe care acestea o au, tot astfel cum se procedează în cazul masivelor unidimensionale.

Se consideră funcţia *poz*(E_m) care indică poziţia elementului E_m în şirul de articole.

Astfel, dacă mulţimea de elemente E_1, E_2, …, E_n are exact *n* elemente:

$$1 \leq poz\ (E_j) \leq n \qquad (18.41)$$

$$poz\ (succ\ (E_j)) = poz\ (E_j) + 1 \qquad (18.42)$$

$$poz\ (pred\ (E_j)) = poz\ (E_j) - 1 \qquad (18.43)$$

Din cele două relaţii rezultă că:

$$poz\ (succ\ (E_j)) - poz\ (pred\ (E_j)) = 2 \qquad (18.44)$$

sau:

$$poz(E_j) = \frac{poz(succ(E_j) + poz(pred(E_j)))}{2} \qquad (18.45)$$

Cu aceste definiţii:

$$adr(E_j) = A + (poz(E_j) - 1) * lg(E_j) \qquad (18.46)$$

se construieşte şirul de perechi:

$$(cont(E_j.x), poz(E_j)) \qquad (18.47)$$

care permite, în cazul în care se stabileşte o relaţie de forma:

$$poz(E_j) = \rho\ (cont\ (E_j.x)) \qquad (18.48)$$

regăsirea elementului după poziţie pe care o are în fişier.

În cazul în care nu se identifică funcţia $\rho(\ )$, elementele $cont(E_j.x)$, j = 1, 2, ..., n, vor fi memorate într-o structură omogenă unidimensională şi poziţia elementului în care este memorată această informaţie, corespunde poziţiei articolului în fişier.

18.6 Fişiere interdependente

Interdependenţa fişierelor, este privită sub două aspecte şi anume: interdependenţa la nivelul articolelor unui fişier şi, respectiv, interdependenţa la nivelul a două sau mai multe fişiere.

Se consideră două fişiere:

- fişierul *PRODUSE*, ale cărui elemente sunt generate după structura *PROD*, cu câmpurile *cod produs, denumire produs, stoc, unitate de măsură, număr materii prime* care intră în componenţa sa şi materiile prime, gama de operaţii;
- fişierul *MATERIALE*, ale cărui elemente sunt generate după structura *MAT*, cu câmpurile *cod material, denumire material, unitate de măsură, cantitate existentă în stoc, preţ unitar*.

Dacă dorim să aflăm costul materiilor prime necesare realizării unui produs, citim din fişierul *PRODUSE* articolul corespunzător respectivului produs şi rând pe rând preluăm codurile materialelor ce intră în componenţa sa. Pentru fiecare material ce intră în componenţa produsului, citim câte un articol din fişierul *MATERIALE*.

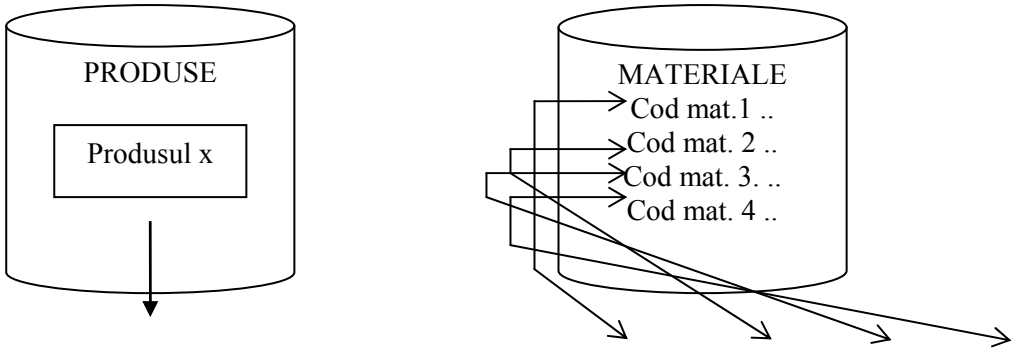
Să ne imaginăm că fişierele sunt secvenţiale. Rezolvarea este extrem de greoaie, pentru că accesul la materiale este obţinut numai prin baleierea de la început a fişierului *MATERIALE*, dacă materialele nu au fost memorate

în ordine crescătoare după codurile acestora în articolul din fişierul *PRODUSE* şi dacă fişierul *MATERIALE* nu este sortat.

Problema devine eficientă dacă, în locul codurilor materialelor, dispunem de adresele articolelor în fişierul *MATERIALE*, dar reorganizarea fişierului de materiale, ar atrage după sine actualizarea adreselor pentru materiale din componenţa fişierului *PRODUSE*.

Dacă se lucrează cu fişiere indexat – secvenţiale, rezultatul este bun, iar dacă se folosesc poziţiile materialelor dintr-o listă, performanţa devine şi mai bună.

Acest tip de dependenţă este unidirecţională între două fişiere. Numărul de legături dintre un articol din fişierul *PRODUSE* şi fişierul *MATERIALE* este variabil, strict dependent de numărul de materiale ce intră în componenţa produsului cu care articolul din fişierul *PRODUSE* este pus în corespondenţă.



Cod produs	Denumire produs	UM	Preţ	Nr. mat. 4	Cod mat. 1	Cod mat. 2	Cod mat. 3	Cod mat. 4

Figura 18.4 *Legătura între fişierele PRODUSE şi MATERIALE*

Este de dorit ca fişierele *PRODUSE* şi *MATERIALE* să se realizeze într-o primă etapă cu codurile materialelor şi lângă ele să fie rezervate câmpuri pentru memorarea adreselor fizice ale articolelor ce corespund în fişierul *MATERIALE* respectivelor coduri.

În a doua fază, un sistem de programe identifică poziţia fiecărui articol şi încarcă în fişierul *PRODUSE* în zonele disponibile de adresă chiar adresa articolului al cărui cod se află în imediata sa vecinătate.

Reorganizarea fişierelor pe fondul definirii codurilor nu implică decât reîncărcarea de adrese, lucru ce se efectuează automat.

Fişierele interdependente privesc legături într-un singur sens, de la fişierul de produse către fişierul de materiale. Se construiesc şi legături în sensul opus, pentru fiecare articol al fişierului de materiale, indicându-se în care dintre produse, este folosit respectivul material. Este o informaţie necesară, dar care lungeşte mult articolul *MAT* şi de aceea, în acest caz este puţin utilizată o astfel de abordare.

Dacă totuşi se doreşte şi o legătură dinspre fişierul *MATERIALE* către fişierul *PRODUSE*, realizarea se efectuează în acelaşi fel, parcurgându-se tot doi paşi.

În final se vor obţine legăturile dintre articole, ilustrate în figura 18.5.

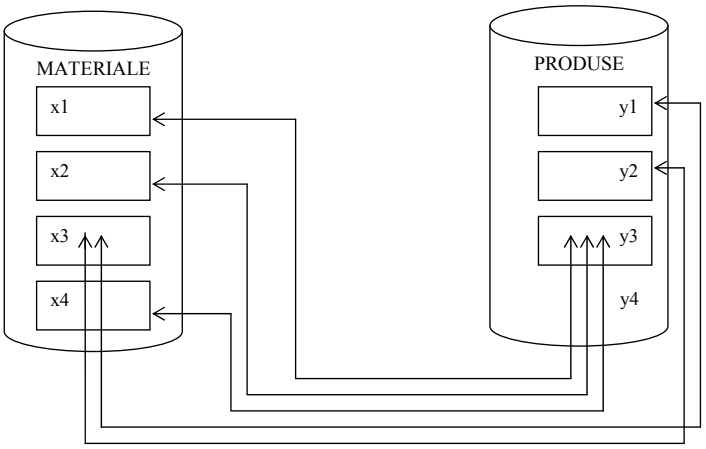


Figura 18.5 *Legăturile dintre articolele fişierelor MATERIALE şi PRODUSE*

Rezultă că produsul y_3, utilizează materialele x_1, x_2 şi x_4, iar materialul x_3 este prezent în compoziţia produselor y_1 şi y_2.

Articolele fiecărui fişier rămân în continuare independente unele de celelalte. Ele sunt prelucrate separat, singurele combinaţii fiind cele legate de posibilitatea de a permite calculul necesarului de materiale, pentru realizarea de *k* produse de tipul y_3 şi de a stabili dacă stocurile materialelor x_1, x_2, x_4 sunt suficiente.

La serviciul aprovizionare, prin parcurgerea fişierului *MATERIALE*, se observă care sunt viitoarele cereri din materialul x_3.

Se observă că o astfel de construcţie este limitativă, dar cu o serie de artificii de consultare a celor două fişiere se obţin şi alte regrupări de date.

O privire de ansamblu pune în evidenţă că păstrarea articolelor din fiecare fişier ca entităţi independente reduce diversitatea combinaţiilor de date, care pentru un manager competent, reprezintă o sursă sigură de fundamentare a deciziilor.

18.7 Fişiere înlănţuite

Apar întrebările: structurile dinamice cu multe elemente sunt memorate pe suport extern? Se implementează mecanisme de înlănţuire în memoria externă? Cum se generează legăturile exprimate prin adrese dintre elementele deja alocate dacă se specifică numai cheile de identificare a acestora?

Răspunsurile nu sunt simple, dar au fost date cu mulţi ani în urmă, obţinându-se funcţii de creare şi prelucrare a fişierelor înlănţuite.

Se consideră că pentru realizarea unui produs, este necesară parcurgerea unor etape, efectuarea unor operaţii de prelucrare într-o anumită succesiune, numită gama de operaţii.

Descrierea unei operaţii, conţine informaţii precum:

- codul operaţiei;
- denumirea operaţiei;
- durata de execuţie;
- calificarea cerută;
- tipul de meseriaş care o execută;
- utilajul necesar;
- scule necesare;

- plata pentru efectuarea operaţiei;
- operaţia precedentă;
- operaţia următoare;

La execuţia programului de încărcare a fişierului gamei de operaţii, se introduc rând pe rând, datele ce urmăresc şablonul descris mai sus. Pentru primul articol al unei game, operaţia precedentă este *NULL*, iar pentru ultimul articol din gamă, operaţia următoare este de asemenea *NULL*.

Sistemul de creare a fişierului cu acest tip de înlănţuire adaugă două câmpuri ce vor fi iniţializate cu adresele ce corespund poziţiei articolelor pentru operaţia precedentă şi pentru operaţia următoare din gama de operaţii.

Câte game de operaţii sunt definite într-un proces de producţie, atâtea liste dublu înlănţuite vor fi realizate.

Prin parcurgerea unui fişier cu articole înlănţuite, se determină care este necesarul de specializări şi care sunt salariile ce trebuie plătite, dacă sunt realizate *k* produse de tipul *x*, pentru care este necesară efectuarea operaţiilor din gama de operaţii cu un cod specificat.

Fişierul gamei de operaţii, va avea articole ce corespund gamelor G_1, G_2, ..., G_n, care la rândul lor conţin operaţiile O_{11}, O_{12}, ..., O_{k1}, ..., O_{21}, O_{22}, ..., O_{k2}, ..., O_{n1}, O_{n2}, ..., O_{kn}.

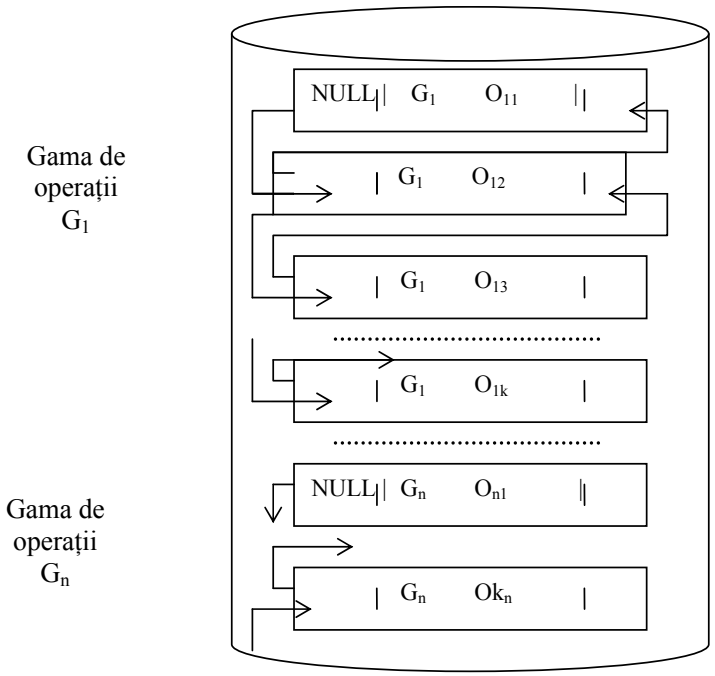


Figura 18.6 *Structura fişierului gamei de operaţii*

O altă situaţie, este aceea corespunzătoare memorării pe suport extern a structurilor arborescente.

Se consideră că la o întreprindere se realizează *N* produse P_1, P_2, ..., P_N. Fiecărui produs i se asociază o structură arborescentă, deci vor exista *N* noduri rădăcină şi *N* noduri terminale.

$$M = \sum_{i=1}^{N} K_i \qquad (18.49)$$

unde K_i reprezintă numărul de materii prime, repere sau subansamble nerealizate în întreprindere şi care intră în componenţa produsului P_i.

Se pune problema memorării celor *N* arborescente, cu condiţia realizării cel puţin a unui nivel de redundanţă controlat, dacă minimizarea este dificil de realizat sau chiar nu este dorită.

Nodurile reprezintă produse, subansamble, repere sau materii prime, despre care trebuie cunoscut:

- cod de recunoaştere, respectiv cheia;
- denumire;
- unitate de măsură;
- cantitate necesară, respectiv greutate specifică;
- preţ unitar;
- gama de operaţii;
- cod reţetă de fabricaţie;
- caracteristici de performanţă standard;

De asemenea, se impune stabilirea poziţiei în arborescenţă, indicând nodul părinte, nodul descendent şi nodul vecin din dreapta.

Această multitudine de date, se grupează în două categorii:

- date pentru descrierea componentei ce corespunde unui nod;
- date pentru descrierea poziţiei nodului în arborescenţă.

Celor două categorii de date le vor corespunde fişiere descriptive şi fişiere de legături.

Fişierele descriptive, au articole formate din două tipuri de date:

- date ce se completează de către utilizatori cu acele elemente ce caracterizează un produs, un ansamblu, subansamblu, reper sau materie primă:
- date ce se completează de un sistem de gestiune a fişierelor înlănţuite şi care conţin adrese de articole sau identificatori de marcare a finalului înlănţuirii.

De exemplu, pentru produsul *A* care este format prin asamblarea reperelor *B, C, D* şi *E* în ordinea menţionată în figura 18.7 fişierul descriptiv, conţine 3 articole ce corespund elementelor *A, B, C, D* şi *E*, iar fişierul de structură descrie legăturile dintre elementele *A, B, C, D* şi *E*, conform arcelor ce definesc gradul de tip arbore. Deci, acest fişier are articolele: (A; B), (A; C), (C; D), (C; E).

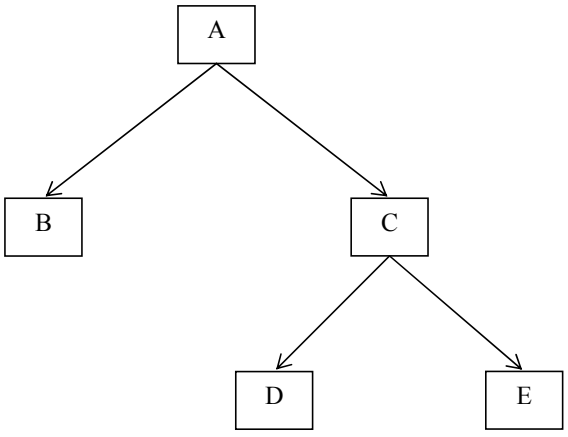


Figura 18.7 *Structura componentelor produsului A*

Indicarea acestor articole nu necesită o anumită ordine. Analiza perechilor este aceea care permite identificarea tuturor elementelor necesare completării câmpurilor de adresă din fişierul descriptiv.

Astfel, într-o pereche *(x; y)* rezultă că pentru nodul *x*, nodul *y* este descendent, iar pentru nodul *y* nodul *x* este părinte.

Nodul *A* este părinte pentru nodurile *B* şi *C*, dar el nu apare în nici una dintre perechi ca descendent, deci *A* este nod rădăcină. În articolul din fişierul descriptiv ce corespunde produsului *A*, un câmp este completat cu *NULL*.

Nodul *B* nu apare ca părinte în nici o altă pereche, deci nu are descendenţi. Se completează şi un câmp din articolul fişierului descriptiv ce corespunde reperului *B*.

Pentru regăsirea rapidă a informaţiilor, înainte de a completa câmpurile din fişierul descriptiv, vor fi completate în fişierul de legătură, câmpurile ce corespund adreselor fizice pe care le ocupă articolele fişierului descriptiv.

Astfel, legăturii corespunzătoare perechii *(A; B)* i se asociază în fişierul de legătură, articolul:

cod A	cod B	adr (articol A)	adr (articol B)

Figura 18.8 *Structura articolului în fişierul de legătură*

Dacă este analizat un produs, problema nu diferă prea mult cu modul de reprezentare a structurilor dinamice în memoria internă. Dacă însă fişierul descriptiv conţine totalitatea nodurilor ce descriu cele *N* produse care se realizează, iar fişierul de legături are atâtea elemente câte arce au cele *N* arborescenţe după care se descompun produsele P_1, P_2, ..., P_n, avem o imagine mai exactă a ceea ce înseamnă reprezentarea în memoria externă a datelor ce formează articole interdependente şi care în final, dau fişierele înlănţuite.

Obţinerea controlului asupra redundanţei revine la a deplasa informaţii privind adresele din articolele fişierului descriptiv în fişierul de legătură. Dacă se minimizează redundanţa, în sensul că fişierul de produse conţine repere şi materii prime descrise o singură dată, reconstituirea arborescenţei se obţine prin construirea tuturor adreselor de regăsire nu în articolele din fişierul descriptiv, ci în articolele de legătură.

Problema traversării unui arbore se menţine din punct de vedere al semnificaţiei, dar în cazul fişierelor înlănţuite revine la a utiliza, alternativ, informaţii din fişierul de legături şi din fişierul descriptiv.

Operaţiile care sunt specifice structurilor arborescente construite în memoria internă sunt definite şi în cazul arborescenţelor de pe mediile externe. Modificările vizează pointerii, al căror conţinut reflectă adrese ale suportului extern.

Funcţiile specifice pentru lucru cu fişiere înlănţuite au ca parametri toate elementele necesare ştergerii unui nod, modificării unor legături sau a unor câmpuri. Adresele care apar la descrierea legăturilor dintre noduri permit parcurgerea arborelui de la rădăcină spre bază sau de la orice nod către bază sau de la orice nod către rădăcină.

Când se proiectează o anumită structură, nu numai listă sau arbore, trebuie ca programatorul să se familiarizeze cu ideea că oricărui arc îi corespund două adrese. El trebuie să găsească algoritmi pentru încărcarea

în câmpuri, pe care are obligaţia să le definească, a acestor două adrese. Programatorul este obligat să cunoască funcţiile care întorc adresa fizică a începutului zonal de memorie pe care o ocupă un anumit articol.

Dacă sistemele de gestiune a fişierelor înlănţuite realizează aceste operaţii pentru structuri arborescente sau pentru liste, cazuri particulare de arborescenţe, în cazul structurilor oarecare programatorul trebuie să-şi construiască funcţii proprii. El are grijă să rezerve zone pentru adrese, îşi defineşte structura de date adecvată descrierii structurii produsului. De fiecare dată, va avea grijă ca ceea ce realizează să nu genereze ambiguităţi sau să conducă la reprezentări ce nu concordă cu structura pe care doreşte să o implementeze în fişiere.

18.8 Fişierele bazei de date

Din punctul de vedere al modului de structurare a datelor, bazele de date reprezintă o formă mai generală de reprezentare a datelor, care reflectă caracteristici ale elementelor omogene ce definesc mai multe mulţimi.

Fie mulţimile M_1, M_2, ..., M_k, formate fiecare din n_1, n_2, ..., n_k elemente, aşa fel alese încât caracterizarea completă a unui element $x_j \in (M_1)$ este efectivă dacă sunt prezentate datele d_{j1}, d_{j2}, ..., d_{jk}, unde $d_{jk} \in (M_k)$.

În plus, fiecare mulţime are elementele structurare după o anumită regulă. Punerea în corespondenţă a elementelor celor k mulţimi, conduc în numeroase situaţii, ca unui element din mulţimea M_i să-i corespundă mai multe elemente din mulţimea M_j sau mai multor elemente din mulţimea M_i să le corespundă un singur element din mulţimea M_h.

Se observă că necesitatea de a separa informaţiile în fişiere distincte este dată în principal din dorinţa de a diminua redundanţa dintr-un fişier, pe de o parte, şi pentru a permite noi facilităţi de exploatare a structurilor de date, pe de altă parte.

În continuare, se consideră un exemplu clasic, foarte frecvent utilizat în descrierea principiilor şi fundamentelor de realizare a bazelor de date.

Fie mulţimea M_1 a persoanelor dintr-o localitate, care se află într-una din relaţiile:

x este vecin cu y
x este fiul lui y
x este tatăl lui y
x este soţia lui y
x este soţul lui y

Un individ al colectivităţii, este descris prin:
- nume;
- adresă;
- raport cu alţi indivizi, respectiv vecini, rude;
- tip automobil/marca;
- culoare automobil,
- an cumpărare;
- loc de muncă.

Fie mulţimea M_2 mulţimea automobilelor, în care elementele se află în relaţiile:

marca x este produsă de y
marca x are capacitate z
capacitatea z are consumul specific w

Un autoturism este descris prin:
- nume producător;
- model;
- culoare;
- an fabricaţie;
- capacitate;
- tip combustibil;
- caracteristici motor;
- consum la 100 km.

Fie M_3 mulţimea locurilor de muncă, formată din elemente caracterizate prin:
- nume instituţie;
- capital social;
- tip instituţie;
- număr salariaţi;
- nume compartiment;
- nume meserii acceptate la compartimentul respectiv;
- nume lucrători din compartiment.

Fie M_4 mulţimea impozitelor care se aplică mijloacelor de transport, formată din elementele:
- limita inferioară a capacităţii cilindrice;
- limita superioară a capacităţii cilindrice;
- impozit;
- taxa CASCO pentru primii 3 ani de funcţionare;
- taxa CASCO pentru maşinile cu vechime cuprinsă între 4 – 10 ani;
- taxa CASCO pentru maşinile cu vechime mai mare de 10 ani.

Deşi se discută foarte mult, cea mai costisitoare etapă din activitatea de manipulare a bazelor de date o reprezintă încărcarea acestora.

În cazul de faţă, datele nu suferă o uzură morală rapidă pentru că vizează indivizii dintr-un cartier sau bloc. În cazul în care fenomenul are o dinamică accelerată, se observă că la terminarea încărcării 60 – 80% din date sunt uzate moral şi baza de date practic este inoperantă.

În cazul considerat, cele patru mulţimi conduc la date ce alimentează baza de date şi se trage concluzia că în continuare ea este complet încărcată.

Pentru a vedea puterea de prelucrare pe care software-ul bazei de date o are, exemplificăm câteva cereri:
- listarea locurilor de muncă ale membrilor familiei lui *x*;
- listarea proprietarilor de autoturisme cu culoarea *z*;
- listarea tuturor celor care lucrează la locul de muncă *w* şi au autoturisme pentru care plătesc taxa CASCO cuprinsă în intervalul [a, b];
- există o relaţie între capitalul social al firmelor şi uzura morală a autoturismelor pe care le au salariaţii lor?

- listarea proprietarilor care plătesc un impozit mai mare decât *C* lei şi taxa CASCO mai mare decât *D* lei;
- există meseriaşi de tipul *z* care au maşini de tipul u absolut noi şi care au vecini în aceeaşi situaţie?

Dacă toate datele despre un individ, sunt înregistrate într-o structură cuprinzătoare, care conţine elementele de descriere ale celor patru mulţimi, se observă o multitudine de repetări, ceea ce conduce de fapt la regruparea informaţiilor. Cele 4 mulţimi nu sunt un dat, ci sunt rezultatul unei analize a modului în care sunt sistematizate şi concentrate datele.

Pentru elementele din mulţimea M_1 se asociază arborescenţa:

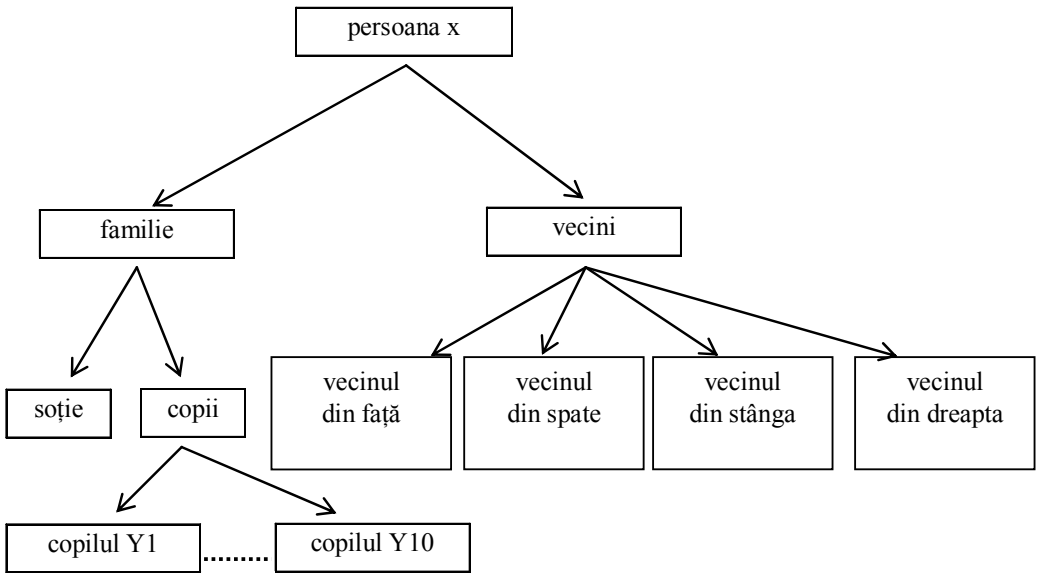


Figura 18.9 *Arborescenţa asociată elementelor mulţimii M_1*

Pentru elementele mulţimii M_2 se asociază arborescenţa din figura 18.10.

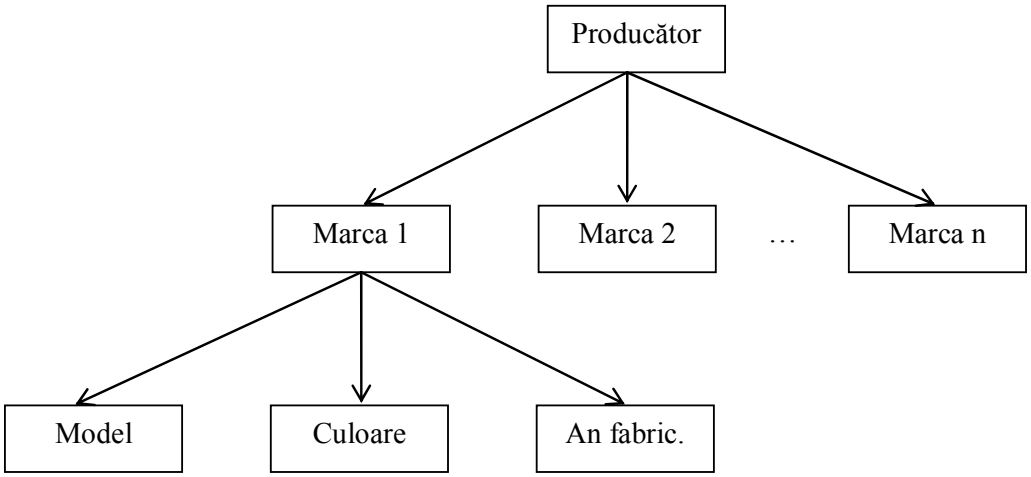


Figura 18.10 *Arborescenţa asociată elementelor mulţimii M_2*

Şi pentru elementele mulţimii M_3 şi M_4 se asociază, de asemenea, arborescenţe.

Deci, dacă fiecare mulţime este concretizată în câte un fişier, articolele în cadrul fiecărui fişier sunt legate între ele, funcţie de structura arborescenţei asociate.

Baza de date presupune atât legături între articolele fiecărui fişier, cât şi legături între fişiere.

Se ridică întrebarea: pentru a răspunde la toate solicitările, este necesară constituirea unor structuri de adrese; toate structurile de adrese sunt create de la început sau acestea se creează pe măsură ce necesităţile de prelucrare impun acest lucru?

În fişierul persoanelor, câmpul corespunzător autoturismului conţine şi adresa articolului ce corespunde mărcii şi culorii autoturismului.

Dacă interesează listarea proprietarilor de maşini de culoare *z* se, procedează astfel:

- se construieşte şirul:

$$S = (s_1, s_2 \dots s_n) \qquad (18.50)$$

unde s_i este adresa articolului pentru descrierea autoturismului al cărui proprietar este persoana a_i din fişierul de persoane.

- din fişierul autoturismelor se extrage subşirul:

$$P = (p_1, p_2, \dots p_n) \qquad (18.51)$$

al adreselor articolelor ce corespund autoturismelor de culoare *z*:

$$P \cap S = (s_{k1}, s_{k2}, \dots s_{km}) \qquad (18.52)$$

ceea ce înseamnă că persoanele ale căror înregistrări ocupă poziţiile k_1, k_2, … k_m cu maşini de culoare *z* şi afişarea:

$$cont\ (ref\ (k_i)\ .\ nume),\ i = 1, 2, \dots, m \qquad (18.53)$$

conduce la tabelarea numelor de persoane care posedă maşini de culoare *z*.

Interogarea unei baze de date revine la constituirea mai multor şiruri, din fiecare mulţime câte unul. Numărul de şiruri maxim este de regulă egal cu numărul mulţimilor de articole definite.

Astfel, dacă se doreşte listarea tuturor celor care au locul de muncă *w* şi au autoturism pentru care plătesc o taxă CASCO cuprinsă în intervalul *[a, b]*, se procedează astfel:

- se construieşte şirul adreselor persoanelor care au locul de muncă *w:*

$$(\gamma_j = const\ (w.adresa\_persoana_j)),\ j = 1, 2, \dots n \qquad (18.54)$$

- se construieşte şirul capacităţilor cilindrice:

$$(C_1, C_2, \dots C_h) \qquad (18.55)$$

al autoturismelor, însoţite de adresele articolelor din fişierul asociat mulţimii M_2, ce corespund autoturismelor pentru capacităţile identificate:

$$m_1\ m_2 \dots m_e \qquad (18.56)$$

De regulă, $e \geq h$, întrucât există mai multe mărci de maşini cu aceeaşi capacitate cilindrică.

Prin analiza conţinutului fişierului M_4, rezultă că autoturismele pentru care se plăteşte o taxă CASCO cuprinsă în intervalul *[a, b]*, sunt cele care au capacităţile cuprinse între:

$$[\alpha_1, \beta_1] \qquad (18.57)$$

$$[\alpha_2, \beta_2] \qquad (18.58)$$

Aceste limite conduc la filtrarea elementelor mulţimii $C_1, C_2, \ldots C_h$, astfel încât rezultă submulţimea adreselor:

$$m' = \{m_1', m_2', \ldots m_p'\} \qquad (18.59)$$

de regulă $p < e$.

Se construieşte şirul de adrese:

$$\delta_u = (\gamma_i / 0 \leq 1 < k; \; cont\ (\gamma.i.adresa\_autoturism) \in m') \qquad (18.60)$$

Scrierea elementelor şirului:

$$(cont\ (\gamma u\text{->}nume\_persoana)), \quad u = 1, 2, \ldots, w \qquad (18.61)$$

rezolvă cererea formulată iniţial.

În exemplul dat, s-a procedat la utilizarea de variabile pointer. Pentru a face deosebire între pointerii folosiţi la referirea elementelor din memoria internă şi pentru a evita confuziile ce apar folosind conceptul de pointer spre fişier, întrucât este deja concentrat tipul de dată *FILE*. În continuare se utilizează conceptul de pointer extern.

Fiind dat un suport extern al căror baiţi au adresa cuprinsă între valorile $[A,B] \cap N$, $A < B$ şi $A, B \in N$, variabila *v* se spune că este pointer extern dacă şi numai dacă:

$$cont\ (v) \in [A, B] \cap N \qquad (18.62)$$

Poziţionarea citirii este de fapt atribuirea sau modificarea conţinutului unei variabile de tip pointer extern, aşa fel încât ea să conţină adresa baitului de unde începe citirea/scrierea.

În enunţul problemei, M_1, M_2, M_3 şi M_4 reprezintă fişiere, adică variabile pointer care sunt iniţializate cu adresele baiţilor de unde începe alocarea memoriei externe pentru fiecare din cele patru fişiere. Dacă se are în vedere că fişierul are şi o etichetă de început de lungime, $cont(M_i) + \alpha$ reprezintă adresa primului articol din fişierul M_i.

Presupunând că fişierului M_i i se alocă o zonă contiguă, cuprinsă între adresele $[A_i, B_i]$, rezultă că:

$$cont\ (M_i) = A_i \qquad (18.63)$$

$$cont\ (\delta_i) = B_i - lg\ (SD_i) \qquad (18.64)$$

unde δ_i reprezintă adresa ultimului element de structură SD_i al fişierului M_i care are o etichetă de sfârşit de fişier de lungime β.

Revenind la fişierele interdependente, fişierele înlănţuite, se observă că structurile de date ce se asociază fişierelor, pe lângă informaţiile utile date de programator, se definesc încă multe câmpuri de tip pointer extern care asigură traversările prin structura externă pe timpul execuţiei.

Folosind definirea pointerilor extern, locul de muncă *w* este identificat prin pointerul extern *pw* ce corespunde adresei articolului respectivului loc de muncă.

Mulţimea $m_1, m_2, \ldots m_e$, reprezintă valori ale variabilei pointer extern *r*, asociat fişierului M_2:

$$m = \{ r_j \mid cont\ (r_j\text{->}capacitate) \in \{C_1, C_2, \ldots C_h\}, i < j < nrr\ (M_2) \qquad (18.65)$$

unde *nrr( )* este o funcţie care defineşte numărul de articole existent într-un fişier:

$$nrr : \{F_1, F_2, \ldots F_k\} \rightarrow N \qquad (18.66)$$

unde *F* este mulţimea fişierelor, date prin valoarea iniţială a pointerilor lor externi:

$$m' = \{ m_i \mid cont\ (m_j\text{->}\ capacitate) \in [\alpha_1, \beta_1] \cup [\alpha_2, \beta_2] \} \qquad (18.67)$$

Selectarea elementelor reprezintă construirea de şiruri de adrese şi apoi prin operaţii de reuniune, intersecţie, se obţin şirurile de elemente care prezintă rezolvarea problemei.

Utilizarea bazelor de date reprezintă un domeniu al informaţiei aplicate. Construirea de sisteme de gestiune a bazelor de date este o preocupare de mare importanţă pentru toţi realizatorii de software, iar elementele prezentate definesc doar o serie de trăsături generale ale filozofiei bazelor de date.

Fiecărui mecanism particular îi corespund reguli precise de definire, iniţializare şi modificare a pointerilor externi ce se asociază fiecărui articol.

De menţionat că în spatele oricărei cereri de informaţie furnizată de utilizatorul bazei de date, se află pointeri externi, care în unele cazuri sunt deja iniţializaţi şi se folosesc ca atare, iar în alte cazuri, sunt numai definiţi şi îşi încarcă conţinutul în funcţie de cerere.

Există situaţii când pentru a rezolva o problemă, se definesc noi pointeri externi şi se activează proceduri de iniţializare, în concordanţă cu problema de rezolvat. În acest caz se creează noi legături între elemente, cu noi posibilităţi de selectare a informaţiilor din baza de date.

Statistic, diversitatea de fişiere face ca utilizarea unora să fie în anumite cazuri mai eficiente decât a altora. Experienţa fiecărui programator este cea care hotărăşte tipul de fişier cu care se lucrează pentru fiecare aplicaţie.

La alegerea tipului de fişier cu care se lucrează, concură o serie de factori din care se enumără:

- numărul de elemente care alcătuiesc fişierul;
- tipul prelucrărilor: integrale, după o cheie, prin selecţie;
- frecvenţa şi tipul operaţiilor de actualizare;
- durata impusă de prelucrare şi necesitatea sortării datelor;

- timpul disponibil pentru elaborarea programelor;
- costuri de elaborare şi utilizare;
- sistemul de calcul disponibil;
- sistemele de gestiune a fişierelor cunoscute şi disponibile.

Alegerea tipului de fişier şi comportamentul algoritmilor implementaţi pentru regăsirea informaţiilor, sunt rezultatul unei analize statistice a datelor, înregistrate prin urmărirea în timp a comportamentului la prelucrare.