Structuri dinamice externe implementate în Pascal

Prof.dr. Valer ROŞCA
Universitatea Sibiu

Structurile dinamice interne, deşi importante în simplificarea şi redarea naturală a algoritmilor, se volatilizează odată cu terminarea programului. În multe probleme se sesizează necesitatea de a construi astfel de structuri pe suport extern, pentru a le stoca în vederea reutilizării. În general, limbajele de programare, inclusiv limbajul Pascal, nu au facilităţi pentru a crea, actualiza şi consulta astfel de structuri. În cele ce urmează se fac câteva sugestii de realizare a structurilor dinamice memorate extern în fişiere.

Cuvinte cheie: *dinamic, structuri de date, stivă, coadă dublu înlănţuită, creare, ştergere, inserare, traversare.*

1. Sugestii de realizare a structurilor externe

1) Structurile dinamice, prin natura lor, reclamă o memorie adresabilă. De aceea, se pot utiliza ca *fişiere suport*, pentru structuri de date externe, fişierele care permit accesul relativ (fişierele binare cu tip sau fără tip).

2) Fişierul suport trebuie creat şi trebuie să posede spaţiul necesar structurii viitoare de date. "Rezervarea" spaţiului se realizează prin scrierea (secvenţială) a unor articole de iniţializare. Alocarea astfel făcută poate fi privită ca o alocare iniţială, urmând ca atunci când este nevoie (depăşire de spaţiu) să se extindă. Alocarea pentru extindere se poate realiza prin adăugarea la sfârşitul fişierului a unui număr de articole de iniţializare. Ultima variantă este recomandată, pentru a evita sustragerea spaţiului de disc de la alte fişiere, spaţiu care ar putea rămâne neutilizat.

3) Spaţiul alocat fişierului suport trebuie împărţit în două zone: *zona dinamică* şi *zona prolog*. Zona dinamică ocupă cea mai mare parte a spaţiului fişierului şi este destinată dezvoltării structurii de date. Prologul cuprinde o serie de informaţii care definesc starea la un moment dat a structurii şi a spaţiului zonei dinamice. Prologul trebuie să cuprindă, cel puţin, valorile pointerilor pentru capetele structurii şi informaţiile necesare gestionării spaţiului. Zona prolog se rezervă, de obicei, la începutul fişierului şi ocupă unul sau mai multe articole, cu o structură convenabilă.

4) Dezvoltarea structurii în zona dinamică presupune implementarea unei metode de gestiune a acestui spaţiu. Cea mai simplă metodă, eficientă în acest caz datorită faptului că articolele au aceeaşi lungime, este *metoda stivei de spaţiu* disponibil. Articolele zonei dinamice se înlănţuie în stivă, de exemplu, în ordinea crescătoare a numerelor de poziţie a articolelor respective, la momentul alocării spaţiului zonei. Se poate utiliza pentru înlănţuire un câmp care, în structura articolului, urmează să fie tratat ca pointer. Valorile câmpului pointer sunt numere naturale şi deci o valoare negativă, (de exemplu -1), poate simula constanta **NIL**. Poziţia articolului cap de stivă va fi memorată în prolog, ca valoare a pointerului stivei.

5) Pentru a facilita lucrul cu structura de date, se recomandă construirea unor proceduri care să realizeze operaţiile de bază pe aceasta. De regulă sunt necesare procedurile pentru inserare, căutare şi ştergere. La aceste proceduri trebuie adăugate proceduri ajutătoare, necesare gestionării spaţiului şi actualizării prologului. Se recomandă proceduri cum sunt:

- *procedură de iniţializare*, care creează fişierul suport, leagă spaţiul în stivă şi iniţializează prologul;
- *procedură de alocare*, destinată stabilirii articolului disponibil pentru fi utilizat de un nod nou al structurii de date;
- *procedura de eliberare*, al cărui rol este de a returna articolul eliberat la rezervorul de

spațiu disponibil ca urmare a ștergerii nodului corespunzător;
- *citire prolog*, pentru aducerea în apelator a informațiilor de stare, necesare în procesul lucrului cu structura de date;
- *procedura de rescriere prolog*, care se apelează pentru a actualiza în fișier informațiile prologului. Se recomandă ca actualizările produse în prolog să fie consemnate dinamic în memoria internă a programului care a creat prologul și numai ultima variantă, cea de la terminarea programului, să fie înscrisă pe fișier;
- *procedura de extindere*, destinată suplimentării spațiului fișierului suport, atunci când este necesar.

6) Programele principale de prelucrare a structurii de date pot fi programe independente. Se poate alege soluția ca unele din ele să fie programe multifuncționale. Decizia depinde de complexitatea operațiilor de prelucrare, de frecvența cu care sunt cerute în execuție etc.

2. Exemplu de realizare a unei cozi externe

O secție de prelucrări mecanice posedă un complex de prelucrare a unei familii de repere. Pentru a fi prelucrate, reperele "așteaptă la coadă" în ordinea sosirii lor din alte ateliere și pe măsură ce sunt executate sunt expediate pentru alte prelucrări. Se cere să se realizeze un sistem de programe care să țină evidența reperelor în așteptare și să furnizeze informații despre ele.

Soluție:

1) Deoarece reperele se prelucrează în ordinea sosirii, evidența lor trebuie să fie ținută printr-o structură de date de tip coadă externă. Pentru a facilita operațiile se va construi o coadă dublu înlănțuită.

2) Se emite ipoteza că informațiile despre un reper se prezintă global, ca un text. Astfel, un nod al cozii va avea structura de triplet (**predecesor, info, succesor**).

3) Fișierul suport va aloca spațiu maxim, nemodificabil, de 500 de articole. Prologul va fi format din primele două articole și va cuprinde informații despre spațiul dinamic și starea cozii. Informațiile despre spațiu ocupă primul articol și sunt: valoarea pointerului stivei de spațiu, suprapus peste câmpul **predecesor,** un contor al numărului de noduri (articole) libere care ocupă, în structura articolului, poziția câmpului **succesor**. Valorile pointerilor celor două capete ale cozii ocupă articolul al doilea și se suprapun, de asemenea, peste câmpurile pointerilor:

față ↔ predecesor, **spate ↔ succesor**.

4) Se presupune că a evidenția și a furniza date despre reperele în coadă înseamnă a realiza următoarele operații: înscrierea unui reper în coadă - **inserare**; scoaterea reperului prelucrat din coadă - **ștergere**; afișarea reperului la rând de prelucrat - **afișare**; afișarea reperelor în coadă - **listare**. Pentru realizarea acestor operații se construiește un program multifuncțional, care le execută la alegere, eventual mai multe operații la o lansare în execuție. Acest lucru presupune o operație convențională - **terminare** - pentru oprirea programului.

5) Includerea tuturor operațiilor într-un singur program reduce partea descriptivă a acestuia și, în Pascal, facilitează realizarea procedurilor. Ca structură, programul se realizează dintr-un program principal, cu proceduri pentru fiecare din operațiile enumerate la punctul anterior. Operațiile de alocare și eliberare de spațiu nu se realizează ca proceduri distincte, având în vedere simplitatea și utilizarea lor unică, ci se cuprind în procedurile de bază care le presupun (inserare, respectiv ștergere). Astfel, dintre procedurile suplimentare, se realizează ca atare numai cele referitoare la citirea/scrierea prologului. Pentru comunicarea între apelator și proceduri s-a ales soluția comunicării prin variabile globale, deoarece folosesc aceleași date (fișierul, prologul și structura nodului). Presupunând că fișierul suport este unic, pentru inițializarea lui se construiește un program separat. În cele ce urmează, se prezintă aceste programe pentru a ușura înțelegerea modului lor de realizare.

Programul de inițializare (*Initializare*) scrie secvențial articolele prologului. Inițial, topul stivei se fixează la doi (s-a stabilit baza stivei la articolul 499), iar contorul de articole disponibile la 488. Pointerii celor

două capete ale cozii sunt inițializați astfel încât să arate situația de "coadă vidă": **fata=spate=1**. În continuare, se realizează înlănțuirea în stivă a spațiului, făcând ca "pointerul" **succesor** al fiecărui nod să aibă ca valoare numărul relativ al nodului următor, ultimul având valoarea 1 (valoare **NIL** convențională). Se remarcă în program descrierea tipurilor de date: tipul **nod**, pentru a descrie structura nodurilor şi tipul **fisier-suport** pentru a descrie fişierul suport. Pe baza acestor tipuri se definesc variabilele cu care se operează : **fissup** ca variabilă fişier şi **reper** ca variabilă de tipul **nod**. Descrierile de tip sunt menținute ca atare şi în programul de tratare a cozii.

```
PROGRAM Initializare;
CONST  nrart = 500;  maxinfo = 20;
TYPE
 datereper = STRING[maxinfo];
 nod = RECORD
         predecesor: INTEGER;
         info: datereper;
         succesor: INTEGER
       END;
 fisiersuport = FILE OF nod;
VAR
 i: INTEGER;  reper: nod;  fissup: fisiersuport;
BEGIN
 Assign(fissup, 'repere.dat' );
 Rewrite(fissup);
 WITH reper DO
  BEGIN   {pregatire nod 0 prolog stiva}
   predecesor:= 2;
   info:= 'STARE SPATIU';
   succesor:= nrart - 2
  END;
 Write(fissup, reper);
 WITH reper DO
  BEGIN {pregatire nod 1 prolog coada}
   predecesor:= 1;
   info:= 'STARE COADA';
   succesor := 1;
 END;
 Write(fissup, reper);
 { Inlantuire noduri IN stiva}
 reper.info:= ' ';
 FOR i:= 3 TO nrart-1 DO
  BEGIN
   reper.succesor:= i;
   Write(fissup, reper)
  END;
 reper.succesor:= 1;
 Write(fissup, reper);
 Close(fissup)
END.
```

Programul pentru tratarea cozii (*Coadaexterna*) conține un modul principal, care apelează o serie de proceduri descrise în conttinuare. El debutează cu apelul procedurii *citireprolog* care aduce în program informațiile de stare. Partea sa centrală este o structură repetitivă, construită prin variabila auxiliară booleană **term**, care face ca execuția să fie continuă atâta timp cât utilizatorul selectează o operație diferită de operația terminare. În interiorul structurii repetitive, alegerea operației este facilitată prin afişarea unui meniu, iar selecția se face prin structura CASE OF. Pentru fiecare caz, se apelează pro-cedura specifică, operația de terminare conducând la apelul procedurii de rescriere prolog. În acest program, tipurile declarate (**datereper,nod**), precum şi variabilele **fissup, fata, spate, top, maxdisp** au caracter global.

```
{programul principal pentru tratarea cozii}
PROGRAM Coadaexterna;
CONST
  maxinfo = 20;
TYPE
  datereper = STRING[maxinfo];
   nod = RECORD
           predecesor: INTEGER;
           info: datereper;
           succesor: INTEGER
         END;
  fisiersuport = FILE OF nod;
VAR  fissup: fisiersuport;
       opr: char;
       term: BOOLEAN;
       fata, spate, top, maxdisp: INTEGER;
{ *********Proceduri*********}
{procedura de deschidere si citire prolog}
PROCEDURE citireprolog;
VAR  reper: nod;
BEGIN
  Assign(fissup, 'repere.dat');
  Reset(fissup );
  WITH reper DO
    BEGIN
      Seek(fissup, 0);
      Read(fissup, reper);
      top:= predecesor;  maxdisp:= succesor;
      Seek(fissup, 1);  Read(fissup, reper);
      fata:= predecesor;  spate:= succesor
    END
END;

{procedura de rescriere prolog si inchidere}
PROCEDURE rescrieprolog;
VAR
```

```
    reper: nod;
  BEGIN
    WITH reper DO
      BEGIN
        predecesor:=top;  succesor:= maxdisp;
        Seek(fissup, 0);     Write(fissup, reper);
        predecesor:= fata;  succesor:= spate;
        Seek(fissup, 1);   Write(fissup, reper)
      END;
    Close(fissup);
  END;

  {procedura afisare reper de prelucrat}
  PROCEDURE afisare;
  VAR
     nodfata: nod;
  BEGIN
    IF fata =1 THEN    {coada vida}
      WriteLn('Nu mai sunt repere de prelucrat !')
    ELSE BEGIN    {cel putin un reper IN coada}
            Seek(fissup, fata);
            Read(fissup, nodfata);
            Write('Reperul de prelucrat este:');
            WriteLn(nodfata.info)
          END
  END;

  {procedura de listare a reperelor existente IN coada}
  PROCEDURE listare;
  VAR
     p, n: INTEGER; nodcurent: nod;
  BEGIN
     IF fata = 1 THEN    {coada vida}
          WriteLn('Nu mai sunt repere IN coada !')
     ELSE BEGIN        {cel putin un reper IN coada}
          n:= 0;  p:= fata;
          WriteLn('Reperele IN coada sunt:');
          WHILE p<>1 DO
            BEGIN
              Seek(fissup, p);
              Read(fissup, nodcurent);
              WriteLn(nodcurent.info);
              n:= n+1;
              p:= nodcurent.succesor;
            END;
            WriteLn('Total repere IN coada = ', n)
        END;
  END;

  {procedura inserare}
  PROCEDURE inserare;
  VAR
    p, q: INTEGER; inforep: datereper;
    nodnou, nodspate: nod;
  BEGIN
    Write('Reperul:'); ReadLn(inforep);
    IF maxdisp = 0 THEN     {lipsa spatiu}
      BEGIN WriteLn('Reperul nu s-a inserat !');
            WriteLn
      END
    ELSE                {inserarea se poate face}
      BEGIN  {alocare nod nou si pregatire inserare}
        p:= top;
        Seek(fissup, p); Read(fissup, nodnou);
        nodnou.info:= inforep;  top:= nodnou.succesor;
        nodnou.succesor:= 1; maxdisp:= maxdisp -1;
        IF spate = 1 THEN {Inseraer IN coada vida}
          BEGIN
            nodnou.predecesor:= 1; spate:= p; fata:= p;
            Seek(fissup, p); Write(fissup, nodnou)
          END
        ELSE    {inserare la spate}
          BEGIN
            q:= spate; Seek(fissup, q);
            Read(fissup,nodspate);
            nodnou.predecesor:= spate; spate:= p;
            nodspate.succesor:= p;
            Seek(fissup, p); Write(fissup, nodnou);
            Seek(fissup, q);Write(fissup,nodspate);
          END;
      END;
  END;

  {procedura stergere}
  PROCEDURE stergere;
  VAR  p, q: INTEGER;
       nodfata, nodurmator: nod;
  BEGIN
    IF fata =1  THEN     {Coada vida}
         WriteLn ('Nu mai sunt repere IN coada !')
    ELSE   {Se sterge nodul din fata}
      BEGIN
        q:= fata; Seek (fissup, q);
        Read (fissup, nodfata);
        IF fata = spate THEN
              {Un singur nod, coada devine vida}
          BEGIN fata:=1; spate:= 1; END
        ELSE   {cel putin doua noduri IN coada}
          BEGIN
            p:= nodfata.succesor;  Seek (fissup, p);
            Read (fissup, nodurmator);
            nodurmator.predecesor:= 1;
            Seek (fissup, p);
            Write (fissup, nodurmator);
            fata:= p;
          END;
    {nodul eliberat trece IN stiva}
      nodfata.info:=' ';
      nodfata.succesor:= top;
      top:= q;
      maxdisp:= maxdisp + 1;
      Seek (fissup, q); Write (fissup, nodfata)
    END
  END;
  { ========Componenta apelatoare==========}
  BEGIN
    citireprolog;   term:= FALSE;
    WHILE NOT term DO
     BEGIN
        WriteLn;
        WriteLn( 'Operatiile admise sunt:');
        WriteLn( ' - Inserare      (I)' );
        WriteLn( ' - Stergere      (S)' );
        WriteLn( ' - Afisare       (A)' );
```

```
        WriteLn( ' - Listare      (L)' );
        WriteLn( ' - Terminare  (T)' );
        WriteLn;
        WriteLn( 'Alegeti operatia prin litera aferenta.');
        WriteLn; Write(' Operatia: '); ReadLn(opr);
        opr:= upcase(opr);
        CASE opr OF
            'I' : inserare;
            'S' : stergere;
            'A' : afisare;
            'L' : listare;
            'T' : BEGIN
                    rescrieprolog; term:= TRUE
                  END;
      ELSE  WriteLn('Operatie necunoscuta!')
      END;
    END;
END.
```

Procedurile *citireprolog* şi *rescriereprolog* sunt complementare. Procedura *citireprolog* deschide fişerul suport şi citeşte informaţiile din nodurile prologului, depunându-le în variabilele globale corespunzătoare (**top**, **maxdisp**, respectiv **fata**, **spate**). Aceste variabile sunt utilizate pe parcursul programului, suferind eventuale modificări de valoare, iar în final, prin procedura *rescrieprolog*, actualizează corespunzător nodurile de proveniență. Procedura de rescriere prolog închide fişierul suport. De remarcat că aceste proceduri citesc/scriu nodurile implicate în acces direct (**Seek** urmat de **Read /Write**).

Procedurile *afisare* şi *listare* sunt similare, prima afişând informaţia din nodul faţă al cozii, iar a doua din toate nodurile cozii. În ambele proceduri se ţine seama de situaţia de "coadă vidă" (**fata=1**). În procedura *listare* se traversează coada şi se numără nodurile vizitate. Informaţia despre repere se afişează într-o formă simplă. Procedurile utilizează variabile locale de tipul **nod,** pentru a referi nodurile cozii.

Procedura *inserare* este mai complexă, ea preluând şi funcţia de alocare a spaţiului. Inserarea se face la spatele cozii, ceea ce explică utilizarea variabilelor **nodnou** şi **nodspate** şi presupune citirea nodurilor implicate pentru a fi modificate. Dacă spaţiul este complet ocupat, lucru care se verifică uşor prin variabila globală **maxdisp**, se amână inserarea până când survin operaţii de ştergere care să elibereze spaţiul. Dacă inserarea se poate face, atunci nodul nou se alocă, se scoate din stivă şi se pregăteşte pentru inserare. Inserarea propriu-zisă diferă după cum coada este vidă sau nu. În primul caz, verificabil prin testul asupra pointerului **spate**, nodul se pregăteşte cu referinţele egale cu **1** şi ambii pointeri de capete (**fata**, **spate**) se fixează să puncteze spre acest nod. În cazul al doilea, nodul nou se ataşează de nodul spate, modificându-se corespunzător legăturile lor şi se actualizează pointerul **spate** pentru a puncta spre acest nod. În ambele cazuri, nodul sau nodurile modificate sunt rescrise în poziţiile de unde au fost citite. În această procedură, variabilele **p** şi **q** se utilizează ca variabile ajutătoare pentru a nu altera, când nu este cazul, variabilele pointeri **fata** şi **spate**.

Procedura *stergere* afectează totdeauna nodul din faţa cozii, dacă un astfel de nod există ("coadă nevidă"). Nodul faţă este citit şi ştergerea decurge diferit după cum el este unicul nod al cozii sau nu. În primul caz, ştergerea efectivă înseamnă doar modificarea pointerilor cozii (**fata=spate=1**) pentru a ilustra situaţia de "coadă vidă". În al doilea caz, trebuie citit şi nodul următor, căruia i se modifică legăturile, astfel încât să devină nod faţă. Se actualizează pointerul **fata** ca să puncteze spre acest nod şi apoi este rescris. În ambele cazuri, nodul şters se returnează la stiva de spaţiu disponibil, pregătind câmpurile sale corespunzător (**succesor:=top** pentru legare în capul stivei) şi rescriindu-l în fişier. Pointerul **top** se actualizează corespunzător, iar variabila **maxdisp** este incrementată pentru a arăta creşterea spaţiu-lui disponibil cu un nod.

Bibliografie:

1. C. Apostol, B. Ghilic-Micu, I. Gh. Roşca, V. Roşca - *Introducere în programare. Teorie şi practică Pascal*, Ed. Viaţa Românească, Bucureşti, 1993
2. I. Gh. Roşca, C. Apostol, B. Ghilic-Micu, V. Roşca, *Prelucrarea fişierelor în Pascal*, Ed. Tehnică, Bucureşti 1994.
3. I. Gh. Roşca, V. Roşca, B. Ghilic-Micu, C. Apostol - *Programare sistematică în Pascal*, în curs de apariţie