Capitolul 3
STRUCTURI DE DATE

3.1. Liste

Pentru a îmbunătăți utilizarea memoriei sunt folosite structuri de date înlănțuite. Acestea poartă numele de liste, fiind compuse dintr-o mulțime de noduri între care sunt definite anumite legături. Se întâlnesc liste simplu înlănțuite, liste dublu înlănțuite, liste circulare, structuri de liste şi liste speciale care sunt formate din atomi şi din alte liste. În continuare se prezintă listele simplu înlănțuite şi dublu înlănțuite.

3.1.1. Liste simplu înlănțuite

Lista simplu înlănțuită este o structură liniară de date, în care fiecare nod este format dintr-o parte de informație ce reține diverse valori caracteristice elementului respectiv şi o parte de legătură ce reține adresa următorului element al listei. Există valoarea NIL ce semnifică adresa către nicăieri folosită în marcarea legăturii ultimului element al listei. Pentru a crea un nou nod, folosim o procedură predefinită `new` iar pentru a elibera spațiul de memorie în momentul în care nu mai este necesară reținerea unui anumit nod, există procedura predefinită `dispose`.

Declararea unei liste simplu înlănțuite se realizează astfel:

```
lista=Anod;
nod=record
              inf:integer;
              leg:lista;
       end;
L:lista;
```

În pseudocod, crearea unei liste simplu înlănțuite poate fi realizată astfel:

```
algoritm creare (L:lista);
{      L:=NIL;
       read (A);
       while (A<>0)  do
              {      new(Pl);
                            P->inf:=A;
                     P->leg:=L;
                     L:=P;
                     readln (A);
              }
}
```

Procesul se încheie la citirea numărului 0. Prezentam în continuare o

variantă recursivă de creare a listei:

```
algoritm creare:lista;
{      read (A);
       if (A=O) then creare:=NIL;
                else
                     {      new(P);
                            P->inf:=A;
                            P->leg:=creare;
                            creare:=P;
                     }
}
```

3.1.2. Parcurgerea unei liste simplu înlănțuite

După ce a fost creată, se doreşte în cele mai multe cazuri prelucrarea listei şi afişarea rezultatelor. Deoarece există numai legătura către următorul element al listei, toate prelucrările vor fi făcute începând cu primul element până la ultimul, în ordinea dată de legături. Cel mai simplu exemplu de prelucrare îl constituie afişarea şirului valorilor memorate în listă. Parcurgem pas cu pas elementele listei şi afişăm informația elementului curent:

```
algoritm listare(L:lista);
{      P:=L;
       while (P<>NIL) do
              {      write (P->inf,' ');
                     P:=P->leg;
              }
)
```

3.1.3. Liste dublu înlănțuite

Într-o listă dublu înlănțuită fiecare nod are o parte de informație şi două legături: legătura `succ` către următorul element şi legătura `pred` către elementul precedent. Putem memora poziția primului element L al listei. De cele mai multe ori se vor memora doua poziții în listă: poziția P a primului element şi poziția U a ultimului element.

Declararea listei dublu înlănțuite se realizează în felul următor:

```
lista=^nod;
nod=record
              inf:integer;
              succ,pred:lista;
       end;
L:lista;
```

Inserarea unui nou nod în interiorul unei liste dublu înlănțuite se execută prin introducerea informației ataşate şi refacerea a patru legături, astfel:

```
new (Q);
Q^.inf:=A;
Q^.succ:=P^.succ;
Q^.pred:=P;
Q^.succ^.pred:=Q;
P^.succ:=Q;
```

Pentru eliminarea nodului desemnat de pointerul P, se vor executa instrucțiunile:

```
Q:=P;
P^.succ^.pred:=Q^.pred;
P^.pred^.succ:=Q^.succ;
dispose (Q) ;
```

3.1.4. Parcurgerea unei liste dublu înlănțuite

Fie o listă dublu înlănțuită, având referințele la capete P, respectiv Q.

a) Să se parcurgă lista de la stânga la dreapta şi de la dreapta la stânga.

b) Să se calculeze referința elementului aflat aproximativ la mijlocul acestei liste.

Prezentam rezolvările în pseudocod:

```
algoritm listare1(P,Q:lista);
{      AUX:=P;
       while (AUX<>NIL) do
              {      write (AUX->inf,' ');
                     AUX:=AUX->succ;
              }
}
algoritm listare2(P,Q:lista);
{      AUX:=Q;
       while (AUX<>NIL) do
              {      write (AUX->inf, ' ') ;
                     AUX:=AUX->pred;
              }
}
algoritm cautare(P,Q:lista):lista;
{      AUXl:=P;
       AUX2:=Q;
       while (AUXl<>AUX2) do
              {      AUXl:=AUXl->succ;
                            if (AUXl=AUX2) then return (AUXl);
                     AUX2:=AUX2->pred;
                            if (AUXl=AUX2) then return (AUXl);
              }
}
```

3.2. Arbori

3.2.1. Arbori liberi

Definiție. Fie o mulțime V de noduri şi o mulțime E de muchii, fiecare muchie legând două noduri distincte.

Se numeşte *lanț* un şir $X_1, X_2, \dots, X_L$ de noduri pentru care oricare două noduri consecutive sunt legate printr-o muchie. Dacă nodurile sunt distincte, lanțul se numeşte elementar.

Se numeşte *ciclu elementar* un lanț $X_1, X_2, \dots, X_L, X_1$ pentru care lanțul $X_1, X_2, \dots, X_L$ este lanț elementar.

Se numeşte *arbore liber* o pereche A = (V,E) cu proprietățile:

l. Oricare două noduri distincte sunt legate printr-un lanț.

2. Nu conține cicluri elementare.

Propoziție. Un arbore liber cu |V| = n noduri are exact n – l muchii. Se poate demonstra prin inducție după numărul n de noduri.

Propoziție. Următoarele afirmații sunt echivalente:

l. A = (V, E) este arbore liber.

2. A = (V,E) cu |V| = n are exact n – 1 muchii şi nu conține cicluri elementare.

3. A = (V,E) cu |V| = n are exact n – 1 muchii şi oricare două noduri sunt legate printr-un lanț.

3.2.2. Arbori cu rădăcină

Definiție. Se numeşte *arbore cu rădăcină* o mulțime de noduri şi muchii în care: există un nod special numit rădăcină, iar celelalte noduri sunt repartizate în k mulțimi disjuncte $A_1, A_2, \dots, A_k$ care sunt la rândul lor arbori cu rădăcină.

Observație.

1. Prin alegerea unui nod drept rădăcină, un arbore liber se poate transforma în arbore cu rădăcină. Totodată, fiecărui nod al arborelui îi va fi asociat un nivel. Nivelul rădăcinii se consideră a fi nivelul 1, iar un nod de pe nivelul i are descendenții direcți pe nivelul i + 1.

2. Pentru fiecare nod, se consideră o ordine a mulțimilor $A_1, A_2, \dots, A_k$.

Spunem atunci că arborele cu rădăcină este *ordonat*.

Definiție. Numim adâncimea unui arbore cu rădăcină nivelul maxim pe care îl are un nod al acestui arbore.

Modalități statice de memorare

1. Putem memora arborele ca o expresie cu paranteze, în care prima poziție este eticheta rădăcinii, urmată, între paranteze, de lista subarborilor respectivi.

2. Putem memora un vector de tați. Vectorul are lungimea egală cu numărul de noduri al arborelui, fiecare poziție i memorând ascendentul direct al nodului i, iar ascendentul rădăcinii (care nu există) se consideră a fi 0.

3.2.3. Arbori binari

Definiție. Un *arbore binar* este un arbore cu rădăcină, în care orice nod are cel mult doi descendenți direcți şi se face distincția între descendentul stâng şi descendentul drept.

Definiție. Un arbore binar se numeşte *arbore binar strict* dacă fiecare nod care are descendenți direcți are exact doi astfel de descendenți.

Definiție. Un arbore binar se numeşte *arbore binar plin* dacă are un număr de n nivele şi pentru toate nodurile de pe nivele 1, 2, ... , n – l există doi descendenți direcți.

Un arbore plin cu n nivele are

$$1 + 2 + ... + 2^{n-1} = 2^n - 1$$

noduri.

Definiție. Un arbore binar se numeşte *arbore binar complet* dacă pe primele n – 1 niveluri are toate nodurile posibile, iar pe ultimul nivel n are o parte din noduri, considerate pe orizontală în ordinea de la stânga la dreapta.

Modalități statice de memorare

1. Putem memora un arbore binar prin memorarea etichetei nodului rădăcină şi folosind doi vectori ST şi DR ce memorează etichetele descendenților direcți stâng respectiv drept. Dacă nu există descendent direct, pe poziția respectiva se va memora valoarea 0.
2. Dacă arborele este arbore binar complet, sau apropiat de un arbore binar complet putem folosi eficient un singur vector, în care legăturile stânga şi dreapta sunt implicite.

3.2.4. Parcurgerea arborilor binari

Preordine

Parcurgerea în preordine constă în vizitarea rădăcinii urmată de vizitarea subarborelui stâng şi apoi a subarborelui drept, acest lucru fiind valabil recursiv, pentru orice subarbore al arborelui considerat. Algoritmul recursiv este următorul:

```
algoritm preordine(A:arbore);
{      if (A<>NIL) then
              {      write (A->INF) ;
                     preordine(A->ST);
                     preordine(A->DR);
              }
}
```

Inordine

Parcurgerea în inordine vizitează, pentru fiecare subarbore, mai întâi subarborele stâng, apoi rădăcina, apoi subarborele drept. Dacă arborele binar respectiv este şi arbore de căutare, atunci parcurgerea în inordine vizitează

vârfurile în ordinea crescătoare a cheilor. Prezentăm algoritmul recursiv:

```
algoritm inordine(A:arbore);
{      if (A<>NIL) then
              {      inordine(A->ST);
                     write(A->INF);
                     inordine(A->DR);
              }
}
```

Postordine

Parcurgerea în postordine vizitează, pentru fiecare subarbore, mai întâi subarborele său stâng, apoi subarborele său drept, apoi vârful rădăcină.

Parcurgerea în postordine se poate realiza recursiv astfel:

```
algoritm postordine(A:arbore);
{      if (A<>NIL) then
              {      postordine(A->ST);
                     postordine(A->DR);
                     write(A->INF);
              }
}
```

Toţi algoritmii recursivi prezentați au şi variante iterative, eliminarea recursivității realizându-se prin folosirea explicită a unor stive. Prezentăm în continuare algoritmul iterativ de parcurgere în preordine:

```
algoritm RSD_iterativ
{      stiva<-vida;
       I :=RAD;
       OK:=true;
       while OK=true do
              {      while (I<>NIL) do
                            {      write( I );
                                   stiva<-I;
                                   I:=ST[I);
                            }
                     if (stiva<>vida) then
                            {      I<-stiva;
                                   I:=DR[I);
                            }
                                   else OK:=false;
              }
}
```

Parcurgerea pe nivele

Dându-se un arbore binar, să se viziteze vârfurile acestuia în ordinea crescătoare a nivelelor. Acest lucru se realizează folosind o coadă auxiliară de noduri vizitate dar neprelucrate. Algoritmul se încheie când coada devine vidă, şi este o particularizare a parcurgerii în lățime a unui graf.

Procedura este iterativă şi poate fi prezentată astfel:

```
algoritm nivele{A:arbore);
ST:stiva; P,U:intregi;
{      P:=O;U:=O;
       U:=U+1;ST[U):=A;write {A->INF);
       while (P<U) do
              { P:=P+1; NOD:=ST[P];
              if {NOD->ST<>NIL) then
                     {      U:=U+1; ST[U):=NOD->ST;
                            write{NOD->ST->INF);}
              if {NOD->DR<>NIL) then
                     {      U:=U+1; ST[U]:=NOD->DR;
                            write{NOD->DR->INF);}
              }
}
```