Tabăra de pregătire a lotului național de informatică

Ploieşti 2006

REPREZENTAREA, GENERAREA ŞI RANGUL ARBORILOR BINARI STRICȚI

1. Secvențe de nivel

Definiție: Pentru un arbore binar strict A, definim funcția care ataşează fiecărui nod numărul întreg corespunzător nivelului pe care se găseşte nodul respectiv în arbore : A = (N , M), *niv* : N $\rightarrow$ **N** .

Definiție: Definim recursiv funcția *niv* :

- nodul rădăcină are nivelul 0, *niv*(r) = 0;

- daca nodul n are nivelul x, atunci toți descendenții săi au nivelul x+1, *niv*(n)=x şi n_1, n_2 descendenții direcți ai lui n $\rightarrow$ *niv*(n_1) = x+1, niv(n_2) = x+1.

Pentru un nod n, vom numi valoarea *niv*(n), "numărul de nivel ataşat nodului n".

Definiție: Secvența formată din numerele de nivel ataşate nodurilor terminale considerate în succesiunea dată de parcuregerea de la stânga la dreapta a arborelui se numeşte secvență de nivel asociată arborelui A.

Observație: Parcurgerea de la stânga la dreapta se referă la oricare dintre cele trei parcurgeri clasice (în preordine, inordine sau postordine), parcurgere în care se ignoră nodurile interne.

Exemplu: Pentru arborele binar strict de pondere 5 din figura 5.1 în care sunt specificate numerele de nivel ale tuturor nodurilor, secvența de nivel asociată este: (2, 2, 2, 3, 3).

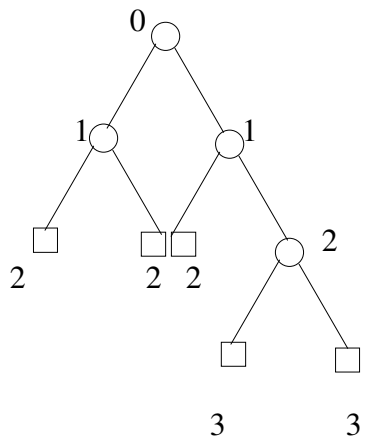

fig. 5.1.

Propoziție: Dacă (a_1, a_2, ... , a_n) este o secvență de nivel, atunci există un indice k$\in$ 1, .., n-1 astfel încât $a_k = a_{k+1}$.

Propoziție: Dacă $(a_1, a_2, ..., a_n)$ este o secvență de nivel, atunci $\sum_{i=1}^{n} 2^{-a_i} = 1$.

Definiție: Fie $(a_1, a_2, ..., a_n)$ este o secvență de numere naturale pentru care există un cel mai mic indice $k \in 1, .., n-1$ astfel încât $a_k = a_{k+1} = s$. Numim *reducere la stânga* înlocuirea în secvența dată a perechii a_k, a_{k+1} prin valoarea s-1, secvența obținută numindu-se *redusa ei stângă*.

Observație: Analog se definește *reducerea la dreapta* ca fiind înlocuirea în secvența dată a perechii a_k, a_{k+1}, pentru cel mai mare k posibil, prin valoarea s-1, secvența obținută numindu-se *redusa ei dreaptă*. Deoarece în raționamentele şi algoritmii următori nu există decât diferențe de formă, vom înțelege prin reducerea unei secvențe - reducerea la stânga - , iar prin redusa unei secvențe - redusa la stânga.

Exemplu: Secvenței (2, 2, 2, 3, 3) din exemplul anterior i se pot aplica succesiv reducerile (2, 2, 2, 3, 3) → (1, 2, 3, 3) → (1, 2, 2) → (1, 1) → (0).

Propoziție: O secvență $(a_1, a_2, ..., a_n)$ este o secvență de nivel dacă şi numai dacă, prin efectuarea tuturor reducerilor stângi posibile, se ajunge în final la secvența (0).

Observație: Procedeul descris mai sus reprezintă un algoritm consistent, fiind şi o demonstrație constructivă a bijectivității funcției ce atașează arborelui binar strict secvența de nivel corespunzătoare.

Lema 1 : Şirul de numere naturale $(a_1, a_2, .. , a_n)$ cu $a_1 < a_2 < .. < a_{k-1}$ şi $a_k > a_{k+1} > .. > a_n$ este o secvență de nivel dacă şi numai dacă $a_k = a_{k+1} = n - 1$ şi $a_1, a_2, .. , a_{k-1}, a_k , a_{k+1}, .. , a_n$ sunt distincte două câte două.

Demonstrație:

(→) Datorită faptului că, în afara perechii a_k, a_{k+1}, oricare două elemente succesive ale şirului sunt distincte obținem că $a_k = a_{k+1}$. Din relația $\sum_{i=1}^{n} 2^{-a_i} = 1$ scriem $\sum_{i=1}^{k} 2^{-a_i} + \sum_{i=k+1}^{n} 2^{-a_i} = 0.\,0..010..010..01_{(2)} + 0.\,0..010..010..01_{(2)}$ (cifrele 1 apar pe pozițiile $a_1, a_2, .. , a_k$ şi respectiv $a_n, a_{n-1}, .. , a_{k+1}$).

Suma se calculează adunând cifrele de acelaşi ordin între ele şi nu poate conduce la rezultatul 1 decât dacă se realizează pe fiecare poziție adunări cu trecere peste ordin, deci pe fiecare poziție există o cifră nenulă, fie în primul număr, fie în al doilea (numerele fiind de

aceeaşi lungime, transportul apărut la adunarea cifrelor 1 de pe poziţiile a_k şi respectiv a_{k+1} se va transmite mai departe până la punctul zecimal cu condiţia ca să nu existe o pereche de zerouri pe aceeaşi poziţie; nici două cifre 1 nu pot exista deoarece am obţine un rezultat zecimal). În concluzie toate numerele $a_1, a_2, .. ,a_{k-1}, a_k , a_{k+1}, .. ,a_n$ sunt distincte, iar a_k şi a_{k+1} au valoarea n-1 (obţinută prin numărarea elementelor $a_1, a_2, .. ,a_{k-1}, a_{k+2}, .. ,a_n$).

(←) Este suficient să observăm că secvenţei de forma $(a_1, a_2, .. ,a_{k-1}, n-1, n-1, .. ,a_n)$ i se vor aplica exact n-1 reduceri, deoarece la fiecare reducere, elementul apărut în urma reducerii devine egal fie cu elementul din stânga, fie cu cel in dreapta sa. Astfel, la prima reducere apare un nou element egal cu n-2, după a doua reducere un nou element egal cu n-3, etc. După ultima reducere secvenţa va avea lungime 1 şi va conţine numai numărul 0.

Definiţie: Secvenţa $(a_1, a_2, .. , a_j)$ se numeşte prefixul unei secvenţe de nivel dacă există numerele $a_{j+1}, a_{j+2}, .. , a_n$ astfel încât $(a_1, a_2, .. , a_j, a_{j+1}, a_{j+2}, .. , a_n)$ să fie o secvenţă de nivel.

Lema: O secvenţă $(a_1, a_2, .. , a_j)$ este prefixul unei secvenţe de nivel dacă şi numai dacă, efectuând toate reducerile stângi posibile asupra ei obţinem secvenţa strict crescătoare $(r_1, r_2, ... ,r_p)$, cu $r_p \le n - j + p - 1$.

Reprezentare

Se porneşte de la un tablou de adrese ale nodurilor terminale (secvenţa de nivel dată), la fiecare adresa memorându-se nivelul nodului respectiv şi adresele celor doi descendenţi (nil deoarece nodul curent este terminal). Tabloul iniţial reprezintă frunzele arborelui ca pădure de subarbori degeneraţi.

Se unesc (într-o implementare recursivă) subarborii reducând chiar în tablou poziţiile corespunzătoare perechii ce se înlocuieşte cu părintele său.

Dacă nu se mai găseşte o pereche de noduri succesive situate pe acelaşi nivel, deşi nu s-a ajuns cu reducerea tabloului la un singur element sau s-a redus tabloul la un element, dar acesta nu este de nivel 0 (rădăcina), algoritmul generează mesajul de eroare @ Nu este secvenţă de nivel*. În situaţia validă, programul generează recursiv arborele afişat în format TREE.

```
program construire_arbore;
type adresa=^nod;
     nod=record
               niv:byte;
               st,dr:adresa
         end;
var a:array[0..99]of adresa;
    intors:array[1..100]of boolean;
    n,i:shortint; p:adresa;

procedure afisare(q:adresa;ind:shortint);
begin
     writeln(q^.niv);
     if q^.st<>nil then begin
        for i:=0 to ind-1 do
        if not intors[i] then write('│ ')else write('  ');
        write('├─');afisare(q^.dr,ind+1);
        intors[ind]:=true;
        for i:=0 to ind-1 do
            if not intors[i] then write('│ ')else write('  ');
        write('└─');afisare(q^.st,ind+1)
     end
end;

procedure construire;
var k:byte;
begin
     repeat
          k:=1;
          while (k<n)and(a[k]^.niv<>a[k+1]^.niv) do inc(k);
          if k=n then begin writeln('Nu e secv.de nivel');halt end
                 else begin
                           new(p);p^.niv:=a[k]^.niv-1;
                           p^.st:=a[k];p^.dr:=a[k+1];
                           a[k]:=p;
                           for i:=k+1 to n-1 do a[i]:=a[i+1]
                      end;
          dec(n)
     until n=1;
end;

procedure citire;
begin
     write('n=');readln(n);
     for i:=1 to n do begin
         new(a[i]);read(a[i]^.niv);
         a[i]^.st:=nil;a[i]^.dr:=nil
     end;
     readln
end;
begin
     citire;
     construire;
     if p^.niv=0 then afisare(p,0)
                 else writeln('Nu e secv. de nivel')
end.
```

Algoritmul invers presupune citirea structurii arborescente şi construirea secvenței de nivel corespunzătoare. Arborele se citeşte ca succesiune de muchii ale unui graf, (realizând implementarea cu matrice de adiacență - structură fundamental neeconomică pentru arbori- pentru diversitate şi în scopul determinării grafurilor care nu sunt arbori). Algoritmul are la bază parcurgerea în adâncime a grafului, verificându-se simultan proprietăților arborescenței (graf aciclic, conex, arbore binar, strict):

```
program construire_secventa_de_nivel;
uses crt;
type MatAdiacenta=array[1..100,1..100]of boolean;
     VectMarcaj=array[1..100]of boolean;
var a:MatAdiacenta;
    b:VectMarcaj;
    n,m,r:byte;

procedure eroare(a:string);
begin
     delline;
     writeln(a);
     readln;halt
end;

procedure df(i,nivel:byte);
var j,desc:byte;
begin
     desc:=0;b[i]:=true;
     for j:=i+1 to n do
         if a[i,j] then
            if b[j] then eroare('Structura ciclica')
            else if desc=2 then eroare ('Arborele nu este binar')
                 else begin inc(desc);
                      df(j,nivel+1)
                 end;
     if desc=1 then eroare ('Arborele nu este strict')
     else if desc=0 then write(nivel,' ')
end;

procedure citire;
var i,j:byte;
begin
     write('Numar de noduri:');readln(n);
     write('Numar de muchii:');readln(m);
     for m:=1 to m do begin
           readln(i,j);
           a[i,j]:=true;a[j,i]:=true
     end;
     write('Radacina:');readln(r)
end;

begin
     citire;
     df(r,0);
     if m<>n-1 then eroare('Graf neconex')
end.
```

Generare

Generarea tuturor arborilor binari stricți de pondere n se poate reduce deci la generarea secvențelor de nivel de lungime n. Se caută o modalitate ordonată de generare, de exemplu generarea în ordinea lexicografică a secvențelor. Secvențele (1, 2 , ... , n-1, n-1) şi (n-1, n-1, ... , 2 , 1) (fig. 5.3 şi 5.4) sunt prima şi respectiv ultima în ordine lexicografică.

Secvența de nivel imediat următoare unei secvențe date se obține în trei etape:

a) se determină perechea de noduri terminale succesive de pe acelaşi nivel situată cât mai în dreapta (astfel încât să rămână un prefix cât mai lung neschimbat);

b) se mută subarborele terminal astfel determinat în terminalul situat în imediata sa vecinătate spre dreapta (pentru a mări ordinul secvenței generate);

c) dacă sufixul format din următoarele terminale conține o succesiune de noduri situate pe niveluri unitar descrescătoare, subarborele corespunzător va fi răsturnat astfel încât secvența respectivă să apară în ordine crescătoare păstrându-se astfel proprietatea şirului de a fi secvență de nivel.

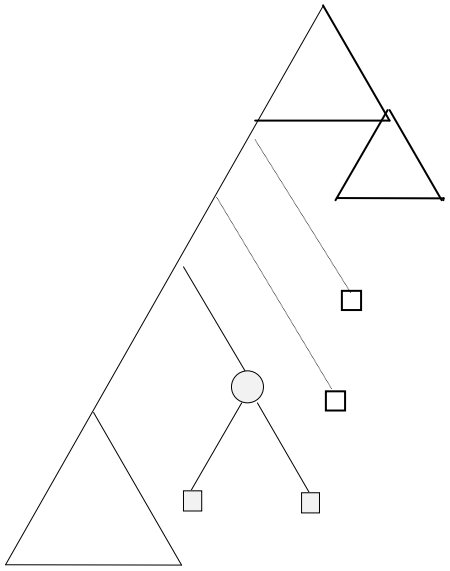

fig. 5.5.

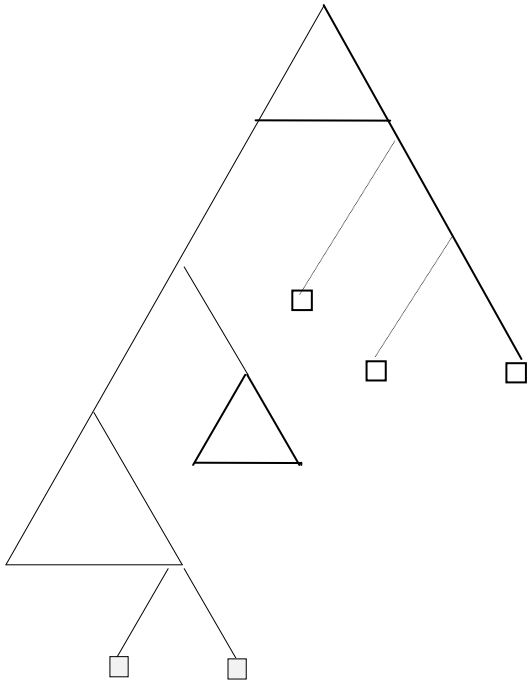

fig. 5.6.

```
program nivel_gen;
var a:array[1..50]of byte;
    i,n:byte;j:integer;

procedure next;
var  k,t,j:byte;
begin
```

```
      k:=n;t:=0;
      while a[k-1]<>a[k] do dec(k);
      while (k+t<n-1)and(a[k+t]=a[k+t+1]+1) do inc(t);
      a[k-2]:=a[k-2]+1; a[k-1]:=a[k-2]+1;
      a[k]:=a[k]-t-1;
      if t<>0 then begin
         for j:=k+1 to n-t-1 do a[j]:=a[j+t];
         for j:=n-t to n-1 do a[j]:=a[n]+j-n+t+1;
         a[n]:=a[n]+t
      end
end;
procedure scrie;
var i:byte;
begin
      write(j:4,':');
      for i:=1 to n do write(a[i]);writeln
end;
begin
      write('n=');readln(n);
      for i:=1 to n-1 do a[i]:=i;
      a[n]:=n-1;j:=1;scrie;
      while a[1]<>n-1 do begin
            inc(j);
            next;
            scrie
      end
end.
```

Algoritmi de rang

Pentru o mulțime M cu n elemente pe care este definită o relație de ordine totală se consideră funcția surjectivă $\varphi : M \rightarrow \{ 1, 2, .. , n \}$ care conservă relația de ordine (dacă x, y $\in$ M cu x < y, atunci $\varphi(x) < \varphi(y)$). Aceasta ataşează elementelor mulțimii M numere între 1 şi n conform ordinii acestora şi poartă numele de funcție de rang. Folosind idempotența celor două mulțimi obținem imediat că φ este o funcție bijectivă. Inversa ei este funcția care selectează din mulțimea M elementul corespunzător unui rang dat şi se numeşte funcție de rang invers.

Observa]ie: Problema numărării arborilor binari stricți cu n frunze este echivalentă cu problema numărării arborilor binari oarecare cu n noduri (terminale şi neterminale) şi a fost rezolvată prin formula lui Catalan (cap.4).

Se ridică o primă problemă care se rezolvă, ca şi formula lui Catalan, printr-o formulă de combinatorică: aflarea numărului de arbori cu n+1 frunze şi care au prima frunză situată pe un nivel dat .

Propoziție: Fie $A_{n,k}$=card{$(a_1, .. , a_n) \mid (a_1, .., a_n)$ este o secvență de nivel şi a_1=k}.

Atunci:

1) $A_{n,k} = \sum_{m_1+m_2+\ldots+m_k=n-k} b_{m_1} b_{m_2} b_{m_3} \ldots b_{m_k}$, unde b_i este numărul arborilor binari stricți cu i+1 frunze.

2) $A_{n,k} = A_{n-1,k-1} + A_{n,k+1}$

3) $A_{n,k} = \frac{k}{2n-k} C_{2n-k}^{n-k}$.

Programul Pascal următor conține algoritmii de rang corespunzători reprezentării arborilor binari stricți prin secvențe de nivel:

a) Se calculează rangul unei secvențe date adunând numărul de secvențe de nivel de lungime n care au prefixul k (cu k între 1 şi a_1-1) cu numărul de secvențe care au prefixul (a_1,k) (cu k între 1 şi a_2-1) . . . cu numărul de secvențe de nivel care au prefixul (a_1, a_2, ... , a_{j-1}, k) (cu k între 1 şi a_j-1), etc. Numărul de secvențe de nivel de lungime n care au prefixul (a_1, a_2, ... , a_{j-1}, k) (pentru k fixat) este $A_{n-j,k}$.

b) Se determină secvența de rang dat cu ajutorul valorilor $A_{n,k}$, construind pe fiecare poziție a secvenței valoarea r_p corespunzatoare din secvența redusă şi, folosind o consecință a lemei 2 (privind redusa prefixului unei secvențe de nivel), se alege a_{j+1} astfel încât fie $a_{j+1} > p+1$ dacă $r_p = p$, fie $a_{j+1} > r_p$ dacă $r_p > p$. ?n oricare dintre situații se diminuează la fiecare pas valoarea rangului cu numărul secvențelor de nivel care au prefixul (a_1, a_2, ... , a_j) construit.

```
program rang_nivel;
uses crt;
type suita=array[0..50] of byte;
     rang_nk= array[1..50,0..49]of longint;
var ank:rang_nk;
    r,a:suita;
    n,i,j,opt:byte;
    x:longint;

function calc(n,k:byte):longint;
var tt:real;i:byte;
begin
     tt:=k/(2*n-k);
     for i:=1 to n-k do tt:=tt/i*(n+i);
     calc:=round(tt)
end;

function verif(a:suita):boolean;
begin
     verif:=false;
     for i:=n-1 downto 1 do begin
```

```
           j:=1;
           while (a[j]<>a[j+1])and(j<i) do inc(j);
           if a[j]<>a[j+1] then exit;
           a[j]:=a[j]-1;
           for j:=j+1 to i do a[j]:=a[j+1]
     end;
     if a[1]=0 then verif:=true
end;
function rang :longint;
var rr:longint;m,i,j,p:integer;
begin
     rr:=1;m:=1;p:=0;r[0]:=0;
     for j:=1 to n-1 do begin
         for i:=0 to a[j]-m-1 do rr:=rr+ank[n-j,m+i-p];
         p:=p+1;r[p]:=a[j];
         while r[p-1]=r[p] do begin
               p:=p-1;r[p]:=r[p]-1
         end;
         if r[p]=p then m:=p+1 else m:=r[p]
     end;
     rang:=rr
end;

function catalan(i:byte):longint;
var tt:real;var j:byte;
begin
     tt:=1/(i+1);
     for j:=1 to i do tt:=tt/j*(i+j);
     catalan:=round(tt)
end;

procedure ranginv;
var m,i,j,p:integer;
    tsum:longint;
begin
     x:=x-1; {conversia de la rang 1...x la 0...x-1}
     m:=1;p:=0;r[0]:=0;
     for j:=1 to n-1 do begin
         i:=0;tsum:=0;
         repeat
               tsum:=tsum+ank[n-j,m+i-p];
               i:=i+1
         until tsum>x;
         x:=x-tsum+ank[n-j,m+i-p-1];
         a[j]:=m+i-1;
         p:=p+1;r[p]:=a[j];
         while r[p]=r[p-1] do begin
               p:=p-1;r[p]:=r[p]-1
         end;
         if r[p]=p then m:=p+1
                   else m:=r[p]
     end;
     a[n]:=r[p];
     for i:=1 to n do write(a[i]:3)
end;

begin
     write('n=');
     readln(n);
     for i:=1 to n do
         for j:=0 to i do
```

```
            ank[i,j]:=calc(i,j);
      repeat
            clrscr;
            writeln('1.FUNCTIA RANG ATASATA SECVENTEI DE LUNGIME ',N);
            writeln('2.SECVENTA DE LUNGIME ',N,' DE RANG DAT');
            readln(opt);
            case opt of
            1: begin
                    write('Dati secventa: ');
                    for i:=1 to n do read(a[i]);readln;
                    if not verif(a) then writeln('nu e secv.de nivel')
                                    else writeln(rang);
                    readln
            end;
            2:begin
                   write('Dati rangul:');readln(x);
                   if x>catalan(n-1) then writeln('nu exista')
                                     else ranginv;
                   readln
            end
            end
      until opt >2
end.
```

2. Secvențe Pal

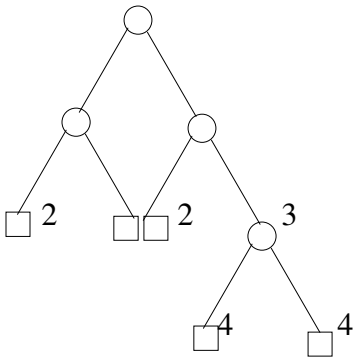

fig. 5. 10.

Prezentare

Pentru un arbore binar strict de pondere n se defineşte secvența (p_1, p_2, .. , p_{n-1}), unde p_i este un întreg reprezentând numărul de noduri interne aflate înaintea nodului i în succesiunea de noduri reprezentând parcurgerea arborelui în preordine.

Exemplu: Pentru arborele din figura 5.10. secvența Pal corespunzătoare este (2, 2, 3, 4, 4).

Observaţie: numărul ataşat ultimei frunze este întotdeauna n-1 şi deoarece nu este semnificativ nu a mai fost luat în considerare.

Propoziție: Un şir de numere naturale (p_1, p_2, .. , p_{n-1}) este o secvență Pal dacă şi numai dacă $p_i \le p_{i+1}$, $\forall i \in \{1, .., n-1\}$, $p_n = n$ şi $p_i \ge i$, $\forall i \in \{1, .., n\}$.

Reprezentare

Algoritmul de reprezentare consta din reconstituirea recursivă a legăturilor de descendență pe baza secvenței Pal corespunzătoare; variabila globală j memorează numărul nodurilor interne parcurse în secvența în inordine până la nodul curent:

```
program reprezentare_secv_Pal;
var p:array[1..200]of byte; {secventa Pal}
    st,dr: array[1..200]of byte; {descendenti stangi, drepti}
    inchis:array[0..200]of boolean; {inchide nivelul de afisare pe
                                      fiecare nivel de indentare}
    n,i,j,jj,k,term:integer;

procedure afisare(j:byte;ind:integer);
begin
     if j=0 then begin writeln ('*');exit end;
     writeln(j);
     intchis[ind]:=false;
     for i:=0 to ind-1 do
         if not inchis[i] then write('│ ')
                          else write('  ');
     write('├-');afisare(st[j],ind+1);
     inchis[ind]:=true;
     for i:=0 to ind-1 do
         if not intors[i] then write('│ ')
                          else write('  ');
     write('├-');afisare(dr[j],ind+1)
end;

procedure construieste(nod:byte);
begin
      if nod=n then exit;
      if nod=p[term] then st[nod]:=0
                     else begin
                           st[nod]:=nod+1;j:=j+1;
                           construieste(st[nod])
                     end;
      term:=term+1;
      if j=p[term] then dr[nod]:=0
                   else begin
                           dr[nod]:=j+1;j:=j+1;
                           construieste(dr[nod])
                     end
end;
function valid:boolean;
begin
     valid:=false;
     if (p[1]<1)or (p[1]>n) then exit;
     for i:=2 to n do if (p[i]<i) or (p[i]<p[i-1]) then exit;
     if p[n+1]<>n then exit;
     valid:=true
end;
begin
     readln(n);
     for i:=1 to n do read(p[i]);
     n:=n-1;
     if not valid then writeln('IMPOSIBIL')
     else begin
```

```
          term:=1;j:=1;construieste(1);
          afisare(1,0)
     end
end.
```

Algoritmul invers citeşte structura arborescenta şi construieşte secvența Pal corespunzătoare prin numărarea propriu-zisă a nodurilor interne pe măsură ce se parcurge în preordine arborele şi prin afişarea numărului determinat în momentul întîlnirii unui nod terminal (cu descendenții st, dr =0). Programul propus are la bază un algoritm simplificat care nu memorează secvența în vederea altor prelucrări şi nu face decât validări minimale:

```
program construire_secventa_Pal;
var st,dr:array[1..100]of byte;
    n,i,r,nr:byte;
procedure eroare(a:string);
begin
     writeln(a);
     readln;halt
end;
procedure parcurg_pal(i:byte);
begin
     if st[i]*dr[i]=0 then write(nr,' ')
     else begin
          nr:=nr+1;
          parcurg_pal(st[i]);
          parcurg_pal(dr[i])
     end
end;
begin
     write('n=');readln(n);
     for i:=1 to 2*n-1 do begin
         write('st[',i,']=');readln(st[i]);
         write('dr[',i,']=');readln(dr[i]);
         if (st[i]=0)xor(dr[i]=0) then eroare('Arbore nestrict')
     end;
     nr:=0;
     write('radacina=');readln(r);
     parcurg_pal(r);
end.
```

Generare

Algoritmul de generare a tuturor secvențelor Pal presupune păstrarea ordinii lexicografice a secvențelor (ca şi în cazul secvențelor de nivel), prima dintre secvențe fiind (1, 2, ... , n), iar ultima (n, n, ... ,n) corespunzând arborilor din figura 5.12.

fig. 5.12.

Algoritmul de generare speculează proprietățile relevate de propoziția de caracterizare prezentată la începutul subcapitolului : se generează toate secvențele care au proprietatea că pe fiecare poziție *i* pot exista numere între *i* şi *n*, iar pozițiile următoare trebuie să satisfacă simultan proprietățile $p_i \le p_{i+1}$ şi $p_i \ge i$:

```
program pal_gen;
var p:array[0..50]of byte; {secventa Pal generata}
    gata:boolean;
    i,n,k:byte;
    j:integer;
procedure next;
begin
     k:=n;
     while p[k]=n do dec(k);
     if k=0 then gata:=true
     else begin
        inc(p[k]);
        for i:=k+1 to n do
            if i>p[k] then p[i]:=i else p[i]:=p[k]
     end
end;

procedure scrie;
var i:byte;
begin
     write(j:4,':');
     for i:=1 to n do write(p[i]);writeln
end;

begin
     write('n=');readln(n);
```

```
     for i:=1 to n do p[i]:=i;
     p[0]:=0;j:=1;
     gata:=false;
     repeat
           scrie;
           next;
           inc(j)
     until gata;
end.
```

5.2.4. Algoritmi de rang

Propoziție: Fie $P_{n,k}$ cardinalul mulțimii $\{(a_1, a_2, .. , a_n) \mid (a_1, .. , a_n)$ este o secvență Pal şi $a_1 = k\}$. Atunci:

1) $P_{n,n} = 1$

2) $P_{n,1} = b_{n-1}$

3) $P_{n,k} = \sum_{j=k-1}^{n-1} P_{n-1,j}$ pentru $k \in [2, n-1]$

4) $P_{n,k} = \frac{k}{2n-k} C_{2n-k}^{n-k}$, $\forall k \in [1, n]$

Propoziție: Algoritmul pentru determinarea rangului unei secvențe Pal date se reduce de fapt la o formulă de calcul: rang $(p_1 , p_2, ... , p_n) = 1 + \sum_{k=1}^{n} \sum_{j=2}^{p_k - k + 1} P_{n-k+1, j}$.

```
program rang_secv_Pal;
var p:array[1..100]of byte; {Secventa Pal}
    n,i,j,k:byte; x:real;
    pnk:array[1..100,1..100]of longint;
    r,rg:longint;

procedure calc;
begin
     for i:=1 to n do
         for j:=1 to i do begin
             x:=j/(2*i-j);
             for k:=1 to i-j do x:=x*(2*i-j-k+1)/k;
             pnk[i,j]:=trunc(x)
         end
end;

procedure rang;
begin
     rg:=1;
     for k:=1 to n do
         for j:=2 to p[k]-k+1 do rg:=rg+pnk[n-k+1,j];
     writeln('rang=',rg)
end;
```

```
procedure ranginv;
begin
     for k:=1 to n-1 do begin
         r:=0;j:=0;
         repeat
               j:=j+1;
               r:=r+pnk[n-k+1,j];
         until r>=rg;
         p[k]:=j+k-1;
         rg:=rg-r+pnk[n-k+1,1]
     end;
     p[n]:=n;
     for i:=1 to n do write(p[i]:4);
     readln
end;

begin
     write('n=');readln(n);
     fillchar(pnk,sizeof(pnk),0);
     calc;
     repeat
           write('rangul secventei dorite=');readln(rg);
           ranginv;
           rang;readln
     end
end.
```

3. Secvenţe Zaks

Prezentare

Secvenţele Zaks corespunzătoare arborilor binari stricţi reproduc în cod binar forma polneză de parcurgere în preordine a arborilor binari.

Se consideră un arbore binar strict cu n noduri interne. Se notează nodurile interne cu 1 şi nodurile terminale cu 0. Parcurgerea arborelui în preordine va genera o secvenţă de 0 şi 1 având lungimea 2n+1 în care ultimul elemnt este obligatoriu 0, deci se omite, obţinându-se o codificare de maximă eficienţă din punctul de vedere al memoriei (pe 2n biţi, faţă de n octeţi în cazul secvenţelor anterior prezentate).

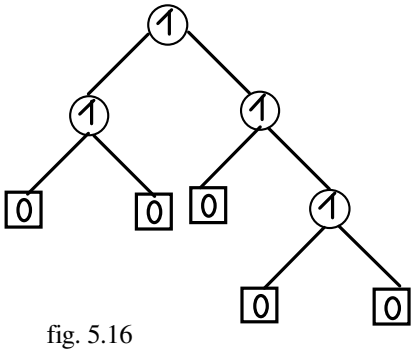

fig. 5.16

Exemplu: Arborelui de pondere 5 din figura 5.16 îi corespunde secvenţa Zaks @de 0 şi 1* : (11001010).

Observaţie: Dacă în secvenţa Zaks notăm numai numărul de cifre 1 aflate în faţa fiecărui 0 obţinem secvenţa Pal corespunzătoare aceluiaşi arbore.

Altă formă de reprezentare a secvenţelor Zaks este aceea în care se notează poziţia fiecărei cifre 1 în secvenţa Zack clasică; se obţine astfel secvenţa @de 1-uri* corespunzătoare. Analog pentru poziţiile zerourilor din secvenţa Zaks clasică se obţine secvenţa @de 0-uri*.

Exemplu: Pentru arborele din figura 5.16 secvenţa Pal este (2, 2, 3, 4), secvenţa @de 1* este (1, 2, 5, 7), iar secvenţa @de 0* este (3, 4, 6, 8).

Definiţie: Spunem că o secvenţă oarecare de 1 şi 0 satisface proprietatea de dominaţie dacă orice prefix al secvenţei conţine tot atâtea sau mai multe cifre de 1 decât cifre de 0. Prin prefixul unei secvenţe de lungime *n* înţelegem subsecvenţa formată din primele *i* cifre ale secvenţei, $i \in \{1,2,3,\ldots,n\}$.

Propoziţie: O secvenţă de 0 şi 1 este o secvenţă Zaks ataşată unui arbore binar strict dacă este formată din tot atâtea cifre de 0 şi 1 şi satisface proprietatea de dominaţie.

Observaţie: Propoziţia precedentă demonstrează relaţia de corespundenţă biunivocă ce se poate stabili între arborii binari stricţi de pondere *n*+1 şi secvenţele de *n* de 0 şi *n* de 1

care satisfac proprietatea de dominație. Din demonstrație deducem mai multe modalități de reducere a unei secvențe @de 0 şi 1* :

* se completează secvența cu un 0 final, se înlocuieşte cea mai din stânga secvență 100 prin 0 de câte ori este posibil şi la sfârşit trebuie să obținem secvența 0;

* se înlocuieşte cea mai din stânga secvență 100 prin 0 de câte ori este posibil şi în final se obține o secvență formată prin concatenarea unui număr oarecare de secvențe 10;

* se elimină orice secvență 10 din secvență de câte ori este posibil şi în final se obține secvența vidă.

Propoziție:

1. Secvențele de întregi $z = (z_1, z_2, \ldots, z_n)$ cu proprietatea că $1<z_1< z_2< \ldots < z_n = 2n$ şi $z_i \geq 2i,\ \forall i \in [1,n]$ şi arborii binari stricți de pondere n+1 se află în corespondență biunivocă (secvențe de 0).

2. Secvențele de întregi $u = (u_1, u_2, \ldots, u_n)$ cu proprietatea că $1 = u_1< u_2< \ldots < u_n < 2n$ şi $u_i \leq 2i-1,\ \forall i \in [1,n]$ şi arborii binari stricți de pondere n+1 se află în corespondență biunivocă (secvențe de 1).

*Observa]ie:*Deci reprezentarea unui arbore atât prin secvența Zaks corespunzătoare, cât şi prin secvența de 1 sau prin secvența de 0 corespunzătoare este consistentă.

Reprezentare

Pentru reprezentarea unui arbore binar strict a cărui secvență Zaks este cunoscută se folosesc reducerile de tipul 100→0 stabilind @identitatea şi legăturile* fiecărui 0 din secvență. Cu alte cuvinte transformăm operația de contragere a subarborilor prezentată la capitolul 5.1.1 în operație de construire şi conectare de subarbori (capitolul 5.1.2).

Deoarece notația corespunzătoare fiecărui nod nu are nici o importanță, au fost stabiliți identificatorii de nod ca numere naturale @de jos în sus* (primele numere naturale - terminalele şi, pe măsură ce se determină, următoarele numere naturale nodurile interne). S-a folosit secvența @de 0* ca date de intrare şi pe baza ei a fost construită secvența Zaks de 0 şi 1 pentru a realiza o procedură mai comodă de validare a datelor de intrare:

```
program construire_arbore_Zaks;
type secv=array[1..100]of byte;
     adr=^nod;
     nod=record
               id:byte;
               st,dr:adr
         end;
var x:secv;
    a:array[1..100]of adr;
    intors:array[1..100]of boolean;
    z:string;
    n,i,nn:byte;
procedure afiseaza(q:adr;ind:shortint);
begin
     writeln(q^.id);
     if q^.st<>nil then begin
        for i:=1 to ind-1 do
        if not intors[i] then write('| ')else write('  ');
        write('├-');afiseaza(q^.st,ind+1);
        intors[ind]:=true;
        for i:=1 to ind-1 do
            if not intors[i] then write('³ ')else write('  ');
        write('└-');afiseaza(q^.dr,ind+1)
     end
end;

procedure uneste;
var i,j,nz:byte;
    temp:adr;
begin
     i:=pos('100',z);
     nz:=0;
     for j:=1 to i do if z[j]='0' then inc(nz);
     delete(z,i,2);
     inc(nn);new(temp);temp^.id:=nn;
     temp^.st:=a[nz+1];temp^.dr:=a[nz+2];
     a[nz+1]:=temp;
     for j:=nz+2 to n do a[j]:=a[j+1]
end;

function valid:boolean;
var i:byte;
begin
     valid:=false;
     if x[1]=0 then exit;
     for i:=2 to n do
         if (x[i]<=x[i-1])or(x[i]<2*i) then exit;
     valid:=true
end;

begin
     write('Ponderea arborelui:');readln(n);
     n:=n-1;
     fillchar(z,2*n,'1');z[0]:=chr(2*n);
     write('Secventa Zacks data de pozitiile cifrelor 0:');
     for i:=1 to n do begin
         read(x[i]);z[x[i]]:='0'
     end;
     z:=z+'0';
     for i:=1 to n+1 do begin
         new(a[i]);a[i]^.id:=i;
```

```
          a[i]^.st:=nil;a[i]^.dr:=nil
      end;
      nn:=n+2;
      if valid then begin
         while z<>'0' do uneste;
         afiseaza(a[1],1)
      end
      else writeln('Secventa incorecta');
      readln
end.
```

Invers, construirea secvenței Zaks pe baza arborelui dat ca structură dinamică sau ca structură statică prin vectorii de descendenți stângi respectiv drepți ai fiecărui nod este o versiune a procedurii de parcurgere în inordine:

```
program construire_secv_Zaks;
type arbore=^varf;
     varf=record
                nod:char;
                st,dr:arbore;

          end;
var rad:arbore;

procedure citire(var elem:arbore);
var x:string;
begin
     readln(x);
     if x<>'' then begin
        new(elem);elem^.nod:=x[1];
        write('introduceti desc.stang pentru ',elem^.nod,':');
        citire(elem^.st);
        write('introduceti desc.drept pentru ',elem^.nod,':');
        citire(elem^.dr)
     end
     else elem:=nil
end;

procedure preordine(elem:arbore);
begin
     if elem^.st=nil then write('0')
     else begin
               write('1');
               preordine(elem^.st);
               preordine(elem^.dr)
     end
end;

begin
    write('introduceti radacina :');
    citire(rad);
    if rad<>nil then preordine(rad);
    writeln(#8' ')
end.
```

Generare

Generarea tuturor secvențelor Zaks în ordine lexicografică se poate realiza generând toate secvențele de 0 în ordine lexicografică sau secvențele de 1 în ordine antilexicografică. Pentru varietatea materialului vom lucra de această dată cu secvențele de 1. Construirea secvenței de 0 şi 1 pe baza secvenței de 1 foloseşte o funcție inversă corespunzătoare: $f_i^{-1}(u_i) = 1\ \underbrace{0 \ldots 0}_{u_{i+1}-u_i-1}$, unde s-a definit artificial $u_{n+1} = 2n$.

```
program generare_zaks;
var a:array[1..100]of byte;
    n,i,j,k:byte;
procedure scrie;
var i:byte;
begin
     for i:=1 to n do write(a[i]:4);
     write('    1');
     for i:=2 to n do begin
         for j:=a[i-1]+1 to a[i]-1 do write('0');
         write('1')
     end;
     for j:=a[n] to 2*n-1 do write('0');
end;

procedure gen;
begin
     for i:=1 to n do a[i]:=i;
     repeat
           scrie;
           i:=n;
           while (a[i]=2*i-1)and(i>1) do dec(i);
           a[i]:=a[i]+1;
           for i:=i+1 to n do a[i]:=a[i-1]+1
     until a[1]=2
end;

begin
     write('n=');readln(n);
     gen
end.
```

5.3.4. Algoritmi de rang

Algoritmul de determinare a rangului unei secvențe Zaks se face ținând cont de următoarele:

* rangul secvenței Zaks este egal cu al secvenței de 0 corespunzătoare;
* rangul secvenței Zaks este complementar în raport cu numărul arborilor binari stricți față de rangul secvenței de 1 corespunzătoare : rang(x) = b_n – rang(u) – 1.

Se consideră arborii binari stricți care au primul terminal pe nivelul k. Acest lucru se traduce pentru secvențele de 1 prin aceea că cel mai mare indice *i* pentru care $u_i = i$ este egal cu *k*. Ne interesează să numărăm arborii care au primul nod pe un nivel mai mare decât *k*.

Propoziție: Fie $U_{n,k} = \text{card}\{(u_1, u_2, .. , u_n) \mid u_j > k,\ j = \max\{ i \mid u_i = i\}\}$. Atunci:

1) $U_{n,k} = U_{n-1,k-1} + U_{n,k+1}$

2) $U_{n,k} = \frac{k+2}{2n-k} C_{2n-k}^{n-k-1}$.

Observa]ie: Din definiția mulțimii $U_{n,k}$ avem : $U_{i,i-1} = 1,$ iar $U_{i,0} = b_i,\ \forall i \in \{1,...,n\}$.

Rangul unei secvențe de 1 (considerat în ordinea lexicografică a generării acestor secvențe) se poate calcula recursiv considerând toți arborii care au primul terminal pe nivel mai mare decât cel pe care se afla primul terminal al arborelui curent, plus arborii care au primul terminal pe acelaşi nivel ca şi primul terminal al arborelui curent, dar inferiori din punct de vedere lexicografic arborelui curent. Aceştia din urmă sunt de forma prezentată în

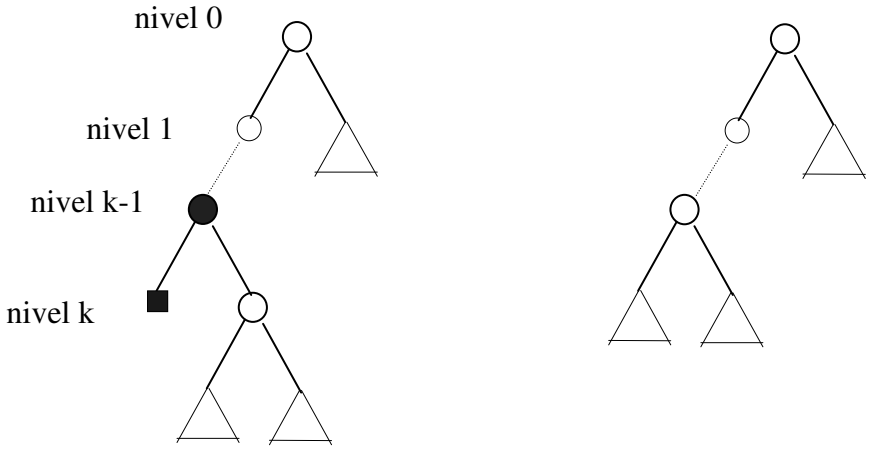

fig. 5.17.

figura 5.17 (stânga) şi sunt echivalenți cu arborii obținuți prin desființarea primului terminal şi a printelui acestuia, transferând întreg subarborele drept al neterminalului desființat în locul acestuia. Deci obținem pentru calculul rangului secvenței $(u_1, u_2, ... , u_n)$ următoarea relație de recurența :

$$\text{rang}(u_1, u_2, ... , u_n) = \begin{cases} 1 & \text{daca } p = n \\ U_{n,p} + \text{rang}(u_1, \ldots, u_{p-1}, u_{p+1} - 2, \ldots, u_n - 2) & \text{daca } p < n \end{cases},$$

unde $p = \max\{j \mid u_j = j\}$. Se determină secvența de 1 când se cunoaşte rangul acesteia calculând pentru fiecare i = 1, 2, ... numărul $U_{i,k}$ maxim care se cuprinde în numărul reprezentând rangul şi diminuând corespunzător rangul, până ce acesta devine 1:

```
program zaks;
uses crt;
var a:array[0..100]of byte;
    t:array[1..100,0..100]of integer;
    n,i,j,k:byte;r:real;c:char;

function rang(n:byte):real;
var k,j:byte;
begin
     k:=n;
     while (a[k]<>k)and(k>1)do dec(k);
     if k<n then begin
        for j:=k to i-1 do a[j]:=a[j+1]-2;
        rang:=t[n,k]+rang(n-1)
     end
     else rang:=1;
end;

procedure ranginv;
    var i,j,niv,ii:byte;
    begin
         niv:=1;i:=1;ii:=1;
         while r>1 do begin
            j:=1;
            while t[n-ii+1,j]>r do inc(j);
            r:=r-t[n-i+1,j];
            for j:=1 to j do a[i+j-1]:=niv+j-1;
            niv:=niv+2;i:=i+j;
         end;
         for j:=i to n do a[j]:=niv+j-1;
         for i:=1 to n do write(a[i]:4);
         readln
    end;

function valid:boolean;
var i:byte;
begin
     valid:=false;{a[0]:=0}
     for i:=1 to n do
         if (a[i]<=a[i-1])or(a[i]>2*i-1) then exit;
     valid:=true
end;

procedure calc;
var x:real;
begin
     for i:=1 to n do
         for j:=0 to i-1 do begin
             x:=(j+2)/(2*i-j);
             for k:=1 to i-j-1 do x:=x*(2*i-j-k+1)/k;
             t[i,j]:=trunc(x)
         end
end;

begin
     write('n=');readln(n);
     calc;
```

```
     repeat
           clrscr;
           writeln('1. RANG'#13#10'2. RANG INVERS'#13#10'3. STOP');
           c:=readkey;
           case c of
                '1':begin
                         write('Secventa de 1:');
                         for i:=1 to n do read(a[i]);readln;
                         if valid then writeln(rang(n))
                                  else writeln('Eroare');
                         readln
                end;
                '2':begin
                         write('Rangul secventei:');readln(r);
                         if r<=t[n,0]then ranginv
                                    else writeln('Eroare');
                         readln
                end
           end
     until not(c in ['1','2'])
end.
```