# 8.6. Arbori binari optimi

## 8.6.1. Definirea arborilor binari optimi

- Până în acest moment, organizarea arborilor binari ordonaţi s-a bazat pe presupunerea că frecvenţele de acces sunt **egale** pentru toate nodurile structurii, cu alte cuvinte toate cheile au **aceeaşi** probabilitate de apariţie.

- Această presupunere este motivată de faptul că de regulă **lipsesc** informaţii referitoare la **distribuţia acceselor** la cheile structurii.

- Există însă situaţii în care se pot cunoaşte **probabilităţile de acces ale diferitelor chei**.

- Un exemplu în acest sens îl constituie activitatea prin care un **compilator** stabileşte dacă un anumit **identificator** este sau nu cuvânt **cheie** (rezervat).

    - **Măsurători statistice** efectuate pe loturi semnificative de programe permit obţinerea de informaţii relativ exacte referitoare la **frecvenţele de apariţie** şi în consecinţă la **frecvenţele de acces** la **cuvintele cheie** ale unui limbaj.

- Structura **arbore binar ordonat** corespunzătoare unor astfel de **cuvinte cheie** este **constantă** (fixă) întrucât cuvintele cheie **nu** se modifică (**nu** se inserează şi **nu** se suprimă).

- Se notează cu `pi` **probabilitatea de acces** la nodul i care are cheia `ki` şi aparţine unei structuri arbore binar. `pi` se defineşte conform formulei [8.6.1.a]:

------------------------------------------------------------

$$P\{x = k_i\} = p_i; \qquad \sum_{i=1}^{n} p_i = 1 \qquad [8.6.1.a]$$

------------------------------------------------------------

- **Problema** care se pune se referă la **organizarea** de o asemenea manieră a **arborelui binar ordonat**, încât **numărul total al paşilor de căutare** contorizat pentru un număr suficient de încercări, să devină **minim**.

- În acest scop, **lungimea drumului de căutare**, definit în &8.1.1 se modifică prin atribuirea unei **ponderi** (greutăţi) fiecărui nod.

    - Nodurile la care accesul se face mai frecvent devin noduri cu **pondere mai mare**, cele vizitate mai rar, noduri cu **pondere mai mică**.

    - **Ponderea** unui nod se asimilează cu **probabilitatea de acces** la acel nod.

- Se defineşte astfel noţiunea de **drum ponderat** asociat unui **arbore binar ordonat**.

* **Lungimea** $P_I$ a **drumului ponderat** asociat unui **arbore binar** este **suma lungimilor** tuturor drumurilor de la rădăcină la fiecare nod al arborelui, fiecare drum fiind **ponderat** cu probabilitatea de acces la nodul respectiv [8.6.1.b].

------------------------------------------------------------

$$P_I = \sum_{i=1}^{n} p_i * h_i \qquad [8.6.1.b]$$

------------------------------------------------------------

* Se precizează că $h_i$ este **nivelul** nodului $i$.

* În continuare se urmăreşte **minimizarea lungimii drumului ponderat** pentru o **distribuţie de probabilităţi dată**.

  * Spre **exemplu** se consideră setul de chei 3,4,7 având probabilităţile de acces $p_1=1/7$, $p_2=2/7$ şi $p_3=4/7$.

  * Aceste chei pot fi aranjate ca şi arbori binari ordonaţi în 6 moduri dintre care 5 sunt diferite (fig.8.6.1.a).

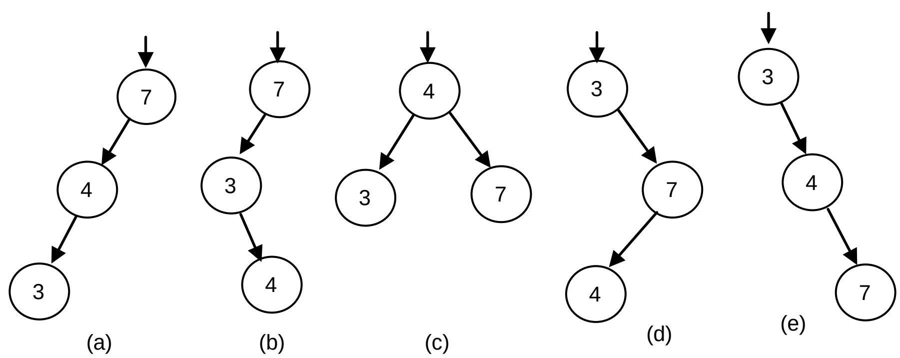


**Fig.8.6.1.a.** Arbori binari cu trei noduri

* Lungimile drumurilor ponderate calculate conform relaţiei [8.6.1.b] sunt [8.6.1.c]:

------------------------------------------------------------

$$\begin{aligned}
P_I^{(a)} &= \frac{1}{7}(1*3+2*2+4*1) = \frac{11}{7} \\
P_I^{(b)} &= \frac{1}{7}(1*2+2*3+4*1) = \frac{12}{7} \\
P_I^{(c)} &= \frac{1}{7}(1*2+2*1+4*2) = \frac{12}{7} \\
P_I^{(d)} &= \frac{1}{7}(1*1+2*3+4*2) = \frac{15}{7} \\
P_I^{(e)} &= \frac{1}{7}(1*1+2*2+4*3) = \frac{17}{7}
\end{aligned} \qquad [8.6.1.c]$$

------------------------------------------------------------

* După cum se observă, în acest exemplu **aranjamentul optim nu** este cel corespunzător arborelui binar perfect echilibrat, ci arborelui degenerat în listă liniară (a).

* Exemplul anterior referitor la activitatea **compilatorului** poate fi reluat într-un **caz mai general** şi anume:

    * Cuvintele preluate din textul sursă **nu** sunt întotdeauna **cuvinte cheie**.

        * De fapt acest lucru este mai degrabă o excepţie.

    * Faptul că un **cuvânt oarecare** `k` **nu** este un **cuvânt rezervat**, respectiv cheia sa **nu** se găseşte în **arborele binar al cuvintelor cheie**, poate fi considerat drept un acces **ipotetic** la un nod "**special**" inserat între **cheia** imediat mai **mică** şi cea imediat mai **mare**.

    * Acestui **nod special** `i` se asociază o **lungime a drumului extern**.

    * Se presupune că se cunosc **probabilităţile de apariţie** $q_i$ ale unor astfel de cuvinte `x` care se situează cu valoarea între două chei $k_i$ şi $k_{i+1}$.

        * Structura **arborelui binar ordonat** poate fi considerabil **modificată**, prin luarea în considerare şi a **căutărilor nereuşite**, respectiv a căutărilor care **nu** se referă la **cuvintele cheie** ale limbajului.

    * În acest caz, **lungimea drumului ponderat total** `P` pentru un astfel de arbore se defineşte cu ajutorul formulei [8.6.1.d].

------------------------------------------------------------

$$P = \sum_{i=1}^{n} p_i * h_i + \sum_{j=0}^{m} q_j * h_j'$$

$$unde \qquad [8.6.1.d]$$

$$\sum_{i=1}^{n} p_i + \sum_{j=0}^{m} q_j = 1$$

------------------------------------------------------------

* Se face precizarea că:

    * $h_i$ este nivelul (adâncimea) **nodului intern** `i`.

    * $h_j'$ nivelul (adâncimea) **nodului extern** `j`.

    * `n` este numărul de **noduri interne**.

    * `m` este numărul de **noduri externe**.

* Lungimea **drumului ponderat total** se mai numeşte şi **cost al arborelul binar ordonat**, întrucât ea reprezintă o măsură a **efortului aşteptat** a fi depus în procesul de căutare.

* Se consideră **toate** structurile **arbore binar ordonat** care pot fi construite pornind de la setul de chei $k_i$, având probabilităţile asociate $p_i$ şi $q_j$.

* Se numeşte **arbore binar optim**, arborele a cărui structură conduce la un **cost minim**.

* În activitatea practică, **probabilităţile** pot fi substituite cu **contoare de frecvenţă**, care memorează **numărul de accese** la un nod.

  * Astfel, se notează cu:

    * $a_i$ = numărul care precizează de câte ori $x$ este egal cu cheia $k_i$.

    * $b_j$ = numărul care precizează de câte ori $x$ este cuprins între cheile $k_j$ şi $k_{j+1}$.

* Prin **convenţie** se consideră că:

    * $b_0$ este numărul care precizează de câte ori $x$ este găsit ca fiind mai mic decât $k_1$, adică cea mai mică cheie.

    * $b_n$ precizează de câte ori $x$ a fost găsit ca fiind mai mare decât $k_n$ , adică cea mai mare cheie (fig.8.6.1.b).

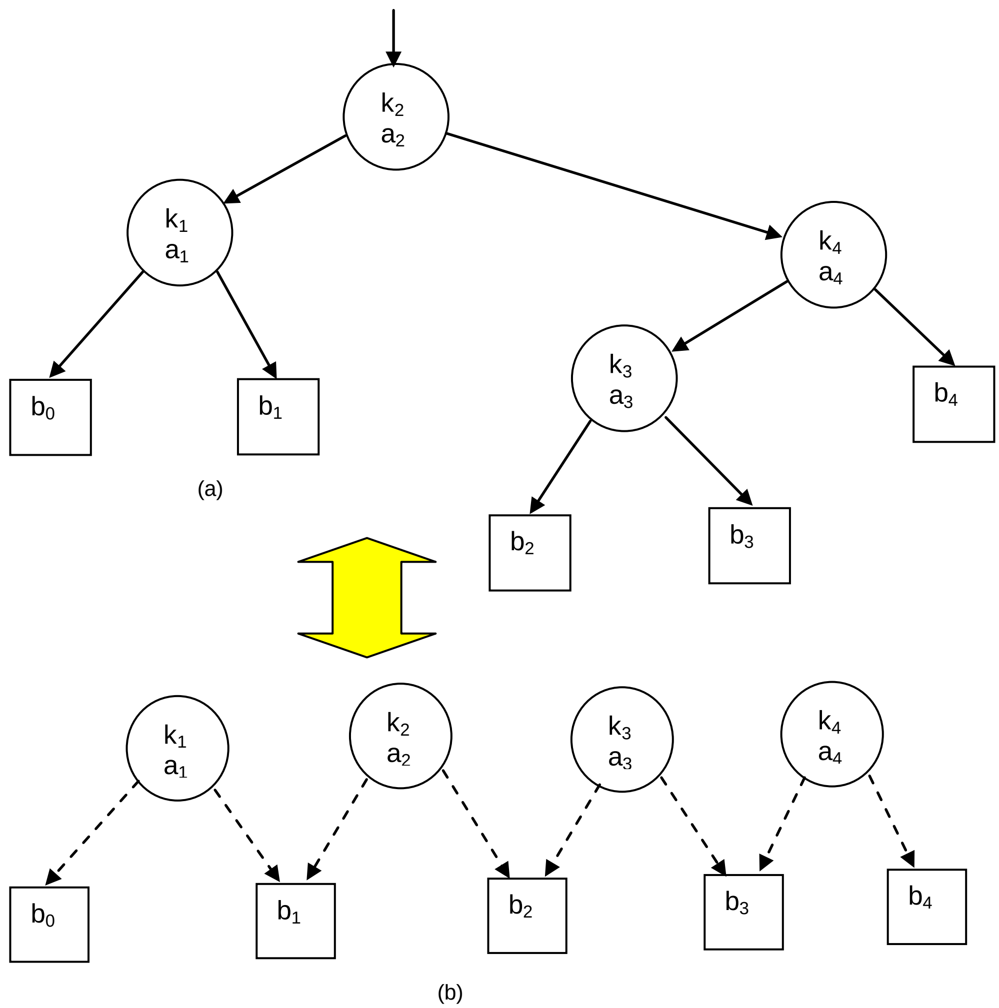


**Fig.8.6.1.b.** Arbore binar optim cu patru chei cu frecvenţe de acces asociate (a), reprezentarea schematică a arborelui $A_{04}$ (b)

* Pentru calculul **lungimii drumului ponderat total** $P$ se va utiliza formula [8.6.1.e]:

---

$$P = \sum_{i=1}^{n} a_i * h_i + \sum_{j=0}^{n} b_j * h_j' \qquad [8.6.1.e]$$

- Astfel pe lângă faptul că se utilizează **frecvenţe** în loc de **probabilităţi**, apare şi avantajul exprimării lui $P$ prin **valori întregi**.

## 8.6.2. Construcţia arborilor binari optimi

- Construcţia arborilor binari optimi **nu** este o treabă simplă.

- Luând în considerare faptul că numărul **configuraţiilor posibile** de arbori cu $n$ noduri creşte **exponenţial** cu valoarea lui $n$, aflarea **arborelui binar optim** pentru valori mari ale lui $n$ pare foarte complicată.

- **Arborii binari optimi** se bucură însă de **proprietatea** foarte importantă că:

    - **Toţi subarborii** unui **arbore binar optim** sunt de asemenea **optimi**.

- Această proprietate sugerează un **algoritm** care **construieşte** în mod **sistematic** arbori binari optimi din ce în ce mai mari, pornind de la nodurile individuale care sunt considerate drept cei mai mici subarbori.

- Arborii cresc astfel de la frunze spre rădăcină, conform metodei "**bottom-up**" (“**de jos în sus**”).

- **Ecuaţia** care este cheia acestui algoritm este [8.6.1.f].

    - Se notează cu $P$ lungimea drumului ponderat total al arborelui.

    - Se notează cu $P_s$ respectiv $P_d$ lungimile drumurilor ponderate corespunzătoare subarborilor stâng respectiv drept ai rădăcinii (fig.8.6.1.c).

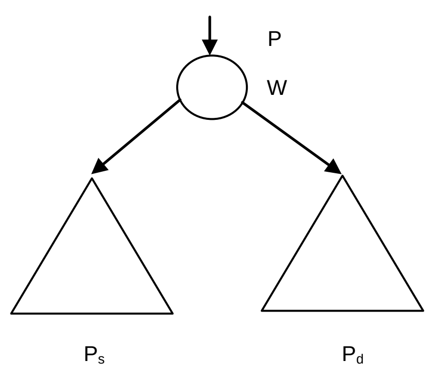


**Fig. 8.6.1.c.** Arbore binar optim

- $P$ poate fi calculat ca fiind egal cu suma dintre $P_s$, $P_d$ şi numărul $W$ care precizează de câte ori se trece prin ramul iniţial al rădăcinii.

- $W$ se numeşte **ponderea arborelui binar optim** şi are valoarea egală cu **numărul total de căutări** efectuate în arbore, adică **suma** frecvenţelor $a_i$ şi $b_j$ pentru toate nodurile cuprinse în arbore [8.6.1.f], [8.6.1.g].

---

$$P = P_s + W + P_d \qquad [8.6.1.f]$$

$$W = \sum_{i=1}^{n} a_i + \sum_{j=0}^{n} b_j \qquad [8.6.1.g]$$

---

- **Media lungimii drumului ponderat** corespunzător acestui arbore este `P/W`.

- În continuare se va preciza **modul de notare** al **ponderilor** şi **lungimilor drumurilor** corespunzătoare tuturor subarborilor care constau dintr-un număr de chei adiacente.

  - Fie $A_{ij}$ **arborele binar optim** reprezentat schematic în fig.8.6.1.d.

  - După cum se observă, $A_{ij}$ este definit prin:

    - Nodurile cu cheile adiacente $k_{i+1}, k_{i+2}, .., k_j$

    - Frecvenţele de acces la chei $a_{i+1}, a_{i+2}, ..., a_j$

    - Frecvenţele de acces interchei $b_i, b_{i+1}, ..., b_j$

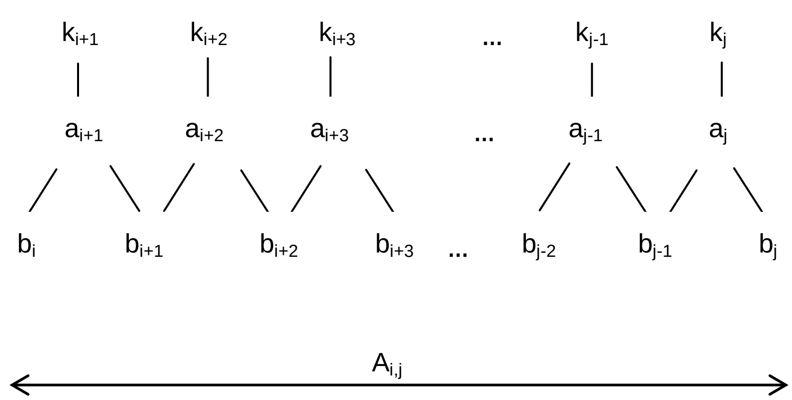


**Fig.8.6.1.d.** Reprezentarea schematică a arborelui binar optim $A_{ij}$

- Se face precizarea că în cadrul structurii arborelui, **proiecţia** cheilor pe axa absciselor produce **secvenţa ordonată crescător** a acestora, indiferent de aranjamentul lor.

  - Această proprietate derivă imediat din observaţia ca **arborii binari optimi** sunt de fapt la origine **arbori binari ordonaţi**.

- Se notează cu $w_{ij}$ **ponderea** şi cu $p_{ij}$ **lungimea drumului total** al subarborelui binar optim $A_{ij}$.

  - Conform celor deja precizate, $w_{ij}$ este suma frecvenţelor $a_i$ şi $b_j$ [8.6.1.g].

  - Atât $w_{ij}$ cât şi $p_{ij}$ sunt definiţi pentru $0 \le i \le j \le n$.

- **Formulele recurente** de pentru calcul valorilor lui $w_{ij}$ respectiv $p_{ij}$ apar în [8.6.1.h] respectiv [8.6.1.i].

------------------------------------------------------------

$$w_{ii} = b_i \quad \text{pentru } 0 \le i \le n \qquad [8.6.1.h]$$
$$w_{ij} = w_{i,j-1} + a_j + b_j \quad \text{pentru } 0 \le i < j \le n$$

------------------------------------------------------------

$$p_{ii} = w_{ii} = b_i \qquad \text{pentru } 0 \le i \le n \qquad [8.6.1.i]$$
$$p_{ij} = w_{ij} + \min(p_{i,k-1} + p_{kj}) \qquad \text{pentru } 0 \le i < j \le n$$

------------------------------------------------------------

- Ultima ecuaţie rezultă imediat din relaţia [8.6.1.f] şi din **definiţia optimalităţii**, adică:

  - **Arborele binar optim** are drept **rădăcină** acel nod având cheia cu indicele $k$, pentru care suma $P_s$ şi $P_d$ este **minimă**.

- De aici apare şi **ideea** care poate conduce la **construcţia arborelui optim** $A_{ij}$ şi anume:

  - Trebuie căutată între toate cheile cuprinse între $k_{i+1}$ şi $k_j$ acea cheie cu indicele $k$ care conduce la un arbore cu $p_{ij}$ minim.

  - Stau la dispoziţie de fapt $j-i$ posibilităţi de a alege rădacina unui astfel de arbore.

  - Deoarece există $(1/2)n^2$ valori ale lui $p_{ij}$ pentru cele $0<j-i\le n$ cazuri unde $i<=j$ (jumătatea superioară a matricei P care memorează lungimile drumurilor), operaţia de minimizare va necesita aproximativ $(1/6)n^3$ operaţii [Kn76].

  - Aceasta înseamnă că un **arbore binar optim** poate fi determinat în $O(n^3)$ unităţi de timp utilizând $O(n^2)$ locaţii de memorie.

- **Knuth** propune **reducerea timpului de execuţie** cu un factor proporţional cu $n$, element care face posibilă utilizarea practică a acestui algoritm.

- Se notează cu R mulţimea indicilor $k$ ai nodurilor pentru care relaţia [8.6.1.i] atinge valoarea minimă.

- Mulţimea R este structurată ca şi o matrice denumită în continuare matricea R.

  - Un element oarecare $r_{ij}$ unde $(i<j)$ al mulţimii R specifică de fapt indicele $k$ al rădăcinii arborelui optim $A_{ij}$.

- Evident în matricea R există câte o locaţie pentru **rădăcina** fiecărui arbore binar optim care poate fi construit plecând de la cheile date $k_i$ şi frecvenţele de acces $a_i$ respectiv $b_j$.

- Căutarea lui $r_{ij}$ care în mod normal trebuie efectuată pe domeniul $j-i$ se poate reduce la un interval mult mai mic.

- **Observaţia** lui Knuth care permite acest lucru este următoarea:

  - Se presupune că $r_{ij}$ este o rădăcină a arborelui optim $A_{ij}$.

  - Se subliniază faptul că $r_{ij}$ este de fapt un indice de cheie a cărui valoare este cuprinsă între $i$ şi $j$.

  - Este evident că rădăcina $r_{ij}$ este cuprinsă:

    - Între rădăcina arborelui rezultat prin suprimarea celui mai din **dreapta** nod al arborelui $A_{ij}$ adică arborele $A_{i,j-1}$.

    - Şi rădăcina arborelui rezultat din suprimarea celui mai din **stânga** nod al său adică $A_{i+1,j}$ aşa cum rezultă din figura 8.6.1.e.

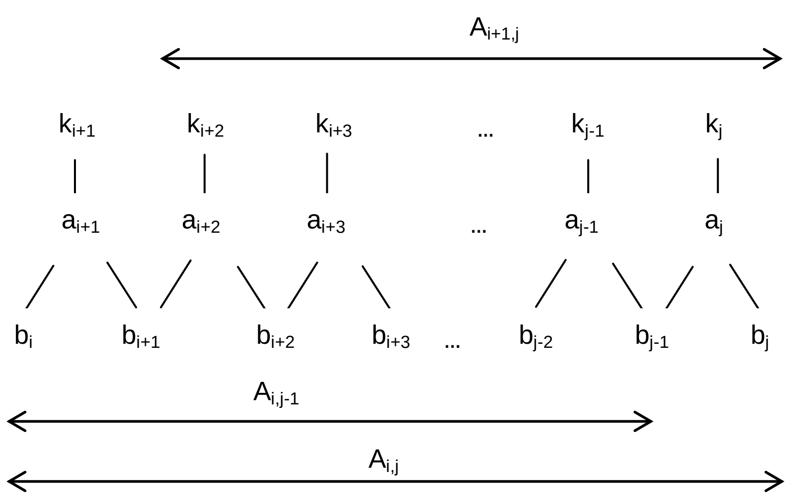


**Fig.8.6.1.e.** Observaţia Knuth. Reprezentarea schematică a arborilor binari optimi $A_{ij}$, $A_{i,j-1}$, $A_{i+1,j}$

  - Această observaţie este o **consecinţă** a faptului că:

    a) Adăugarea unui nod la **dreapta** arborelui produce (eventual) deplasarea spre **dreapta** a rădăcinii arborelui optim.

    b) Suprimarea celui mai din **stânga** nod, produce (eventual) deplasarea tot spre **dreapta** a rădăcinii arborelui optim.

- Acest lucru se exprimă formal cu ajutorul relaţiei [8.6.1.j]:

------------------------------------------------------------

$$r_{i,j-1} \le r_{ij} \le r_{i+1,j} \qquad [8.6.1.j]$$

------------------------------------------------------------

- Această observaţie permite **limitarea procesului de căutare**:

  - Soluţiile pentru $r_{ij}$ trebuiesc căutate **numai** în domeniul $r_{i,j-1}, r_{i+1,j}$.

- Rezultă astfel un număr $O(n^2)$ de paşi elementari pentru construcţia unui arbore binar optim.

- Cu alte cuvinte, pentru a determina **rădăcina arborelui binar optim** $A_{ij}$:

  - (1) Se verifică **toate** perechile de arbori $A_{i,k-1}$ şi $A_{k,j}$ pentru $k \in [r_{i,j-1}, r_{i+1,j}]$.

  - (2) Se alege acel indice $k$ pentru care suma $p_{i,k-1}+p_{k,j}$ este minimă conform relaţiei [8.6.1.i].

- În continuare se trece la descrierea **algoritmului de construcţie** al arborelui binar optim $A_{0n}$ care conţine $n$ chei.

- Se reamintesc următoarele definiţii care se referă la **arborele binar optim** $A_{ij}$ constând din nodurile având cheile $k_{i+1},...k_j$:

  $a_i$ : frecvenţa căutărilor cheii $k_i$.

  $b_j$ : frecvenţa căutărilor unui argument $x$ cuprins între cheile $k_j$ şi $k_{j+1}$.

  $w_{ij}$ : ponderea arborelui binar optim $A_{ij}$.

  $p_{ij}$ : lungimea drumului ponderat al arborelui binar optim $A_{ij}$.

  $r_{ij}$ : indicele cheii rădăcinii arborelui binar optim $A_{ij}$.

- Se declară următoarele structuri de date [8.6.1.k]:

```
------------------------------------------------------------
{Arbori optimi:  Structuri specifice de date - C}

int a[Numar_Noduri]; /*tablou frecvente chei*/   [8.6.1.k]
int b[Numar_Noduri]; /*tablou frecvente interchei*/
int p[Numar_Noduri, Numar_Noduri]; /*matrice lungimi drumuri
                                     ponderate*/
int w[Numar_Noduri, Numar_Noduri]; /*matrice ponderi arbori
                                     optimi*/
int r[Numar_Noduri, Numar_Noduri]; /*matrice radacini arbori
                                     optimi*/
------------------------------------------------------------
{Arbori optimi:  Structuri specifice de date - PASCAL}

TYPE TipIndex=0..n;
VAR  a: ARRAY[1..n] OF integer;                   [8.6.1.k]
     b: ARRAY[TipIndex] OF integer;
     p,w: ARRAY[TipIndex,TipIndex] OF integer;
     r: ARRAY[TipIndex,TipIndex] OF TipIndex;
------------------------------------------------------------
```

- Se presupune că ponderile $w_{ij}$ au fost calculate în mod direct utilizându-se valorile pentru frecvenţele $a$ şi $b$, pe baza relaţiilor [8.6.1.h] şi memorate în matricea W.

- **Procedura de construcţie**:

- Utilizează matricea W drept argument (dată de intrare).

- Construieşte gradual matricea R, care memorează rădăcinile subarborilor binari optimi.

- Construieşte simultan cu R, matricea P utilizată în procesul de construcţie. Matricea P poate fi considerată un rezultat intermediar.

* Procedeul folosit este următorul:

  - (1) Se porneşte cu cei mai mici subarbori posibili, respectiv cei care **nu** conţin nici un nod, adică arborii $A_{ii}$ care au lăţimea `0`.

  - (2) Se construiesc succesiv subarbori cu lăţimi din ce în ce mai mari: 1,2,3, etc.

* Se notează cu `h` lăţimea `j-i` a subarborelui $A_{ij}$.

* Pentru toţi arborii care au lăţimea `h=0`, $p_{ii}$ poate fi determinat direct din relaţia [8.6.1.i] conform secvenţei [8.6.1.l].

```
------------------------------------------------------------
/*Construcţia arborilor optimi de lăţime h=0*/

pentru i=0 la n
   p[i,i]=w[i,i];  /*=b[i]*/                        [8.6.1.l]
------------------------------------------------------------
```

* În cazul lui `h=1`, avem de-a face cu arbori cu un singur nod, nod care în mod evident este şi rădăcina arborelui.

* `i` precizează limita **stânga** pentru index iar `j` limita **dreapta** în arborele considerat $A_{ij}$ ( secvenţa [8.6.1.m]).

```
------------------------------------------------------------
/*Construcţia arborilor optimi de lăţime h=1*/

pentru i=0 la n-1
  |j=i+1;
  |p[i,j]=p[i,i]+p[j,j]+w[i,j];                   [8.6.1.m]
  |r[i,j]=j;
  |□
------------------------------------------------------------
```

* Pentru cazurile în care lăţimea `h` este mai mare ca `1`, se utilizează o **secvenţă repetitivă** cu `h` luând valori succesive cuprinse între `2` şi `n`.

* Cazul `h=n` presupune construcţia arborelui $A_{on}$.

* În fiecare caz, lungimea drumului minim $p_{ij}$ şi indexul $r_{ij}$`=k` asociat rădăcinii se caută printr-o instrucţie de ciclare simplă cu indicele `k` luând valori în intervalul $[r_{i,j-1}, r_{i+1,j}]$ furnizat de relaţia [8.6.1.j].

* Secvenţa de construcţie a arborilor binari optimi cu lăţimea mai mare ca `1` apare în [8.6.1.n].

```
------------------------------------------------------------
{Construcţia arborilor optimi de lăţime h>1}

pentru (h=2 la n)
  pentru (i=0 la n-h)                              [8.6.1.n]
    j=i+h;
     *găseşte acel indice m care  conduce la o valoare
      minimă al lui min. min se determină cu relaţia
         min = minim(p[i,m-1]+p[m,j]), dintre toţi indicii
      m care satisfac relaţia r[i,j-1]<=m<=r[i+1,j];
    p[i,j]=min+w[i,j];
    r[i,j]=m;
    □
------------------------------------------------------------
```

- Detaliile rafinării acestei secvenţe apar în programul **`ReprezentareArbore`** procedura **`ArbOpt`**, secvenţa [8.8.c].

- **Lungimea medie a drumului** pentru arborele $A_{on}$ este furnizată de raportul $p_{on}/w_{on}$ iar rădăcina sa este nodul având indicele $r_{on}$.

- În forma din secvenţa [8.6.1.n], algoritmul de construcţie al arborelui binar optim $A_{on}$ necesită un **efort de calcul** de ordinul O*(n^2)* şi un **volum de memorie** de acelaşi ordin.

    - Aceste valori **nu** sunt în general acceptabile pentru valori foarte mari ale lui `n`.

- **Hu** şi **Tacker** au dezvoltat un alt algoritm care necesită numai O*(n)* locaţii de memorie şi O*(n* log n)* efort de calcul.

    - Acest algoritm ia însă în considerare **numai** cazurile în care frecvenţele de acces la chei sunt nule $(a_i=0)$, adică sunt înregistrate numai accesele interchei [Kn76].

- Un alt algoritm cu performanţe similare a fost descris de **Walker** şi **Gotlieb**.

    - Acest algoritm care construieşte un arbore **aproape optim**, poate fi implementat utilizînd **principii euristice**.

    - **Ideea de bază** este următoarea:

        - (1) Se consideră toate nodurile (adevărate şi speciale) ale structurii arbore, ponderate de frecvenţele (probabilităţile) lor de acces, ca fiind distribuite pe o scară liniară.

        - (2) Se caută nodul care este cel mai apropiat de "centrul de greutate".

            - Acest nod se numeşte "**centroid**" şi **indexul** său se calculează cu formula [8.6.1.o], valoarea obţinută rotunjindu-se la cel mai apropiat întreg.

------------------------------------------------------------

$$\frac{1}{w}\left(\sum_{i=1}^{n} i * a_i + \sum_{j=0}^{n} j * b_j\right) \qquad [8.6.1.o]$$

--------------------------------------------------------------------

    - Dacă toate nodurile au aceeaşi pondere, atunci în mod evident rădăcina arborelui optim coincide cu **centroidul**.

    - În marea majoritate a restului cazurilor, rădăcina se găseşte în vecinătatea apropiată a centroidului, utilizându-se în acest sens o căutare limitată a unui optim local.

  - (3) În continuare procedura este aplicată celor 2 arbori care au rezultat, apoi celor 4 ş.a.m.d.

  - (4) După ce s-a ajuns la arbori suficienţi de mici se poate aplica metoda exactă descrisă anterior.

- Metoda care rezultă, conduce la arbori destul de avantajoşi (în 2-3% din cazuri chiar la soluţia optimă) necesitând O*(n)* unităţi de spaţiu de memorare şi O*(n*log n)* unităţi de timp pentru execuţie.

# 8.7. Arbori Huffman

## 8.7.1. Coduri prefix. Algoritmul lui Huffmann.

- Un exemplu de utilizare al **arborilor binari cu ponderi** ca structuri de date îl reprezintă **codurile Huffmann**.

- Se presupune că se prelucrează **mesaje** care constau din **secvenţe de caractere**.

- În fiecare mesaj, caracterele sunt **independente** şi pot apare în **orice poziţie** a mesajului.

- Se presupune, de asemenea, că se cunoaşte **probabilitatea de apariţie** a fiecărui caracter, probabilitate care **nu** depinde de poziţia caracterului în cadrul mesajului.

    - Spre **exemplu**, se consideră mesaje formate din 5 caractere `a,b,c,d,e` care apar respectiv cu probabilităţile: `0.12, 0.4, 0.15, 0.08, 0.25`.

- Se doreşte **a se codifica fiecare caracter** printr-o **secvenţă de cifre binare** `"0"` şi `"1"`, astfel încât codul unui caracter, să **nu** fie **prefix** pentru codul nici unui alt caracter.

- Această proprietate numită şi "**proprietate de prefix**" permite **decodificarea** unui mesaj format din caractere `"0"` şi `"1"` prin ştergerea repetată a prefixelor şirului care sunt coduri de caractere.

    - Spre exemplu, în fig.8.7.1.a se prezintă două codificări posibile ale simbolurilor anterior precizate.

    - Pentru codul 1, algoritmul de decodificare este simplu: se separă grupe de câte 3 biţi care se decodifică conform tabelei (evident codurile `101`, `110` şi `111` nu există).

    - Astfel, mesajul `001010011` reprezintă şirul original `bcd`.

| Simbol | Probabilitate | Cod 1 | Cod 2 |
|---|---|---|---|
| a | 0.12 | 000 | 000 |
| b | 0.40 | 001 | 11 |
| c | 0.15 | 010 | 01 |
| d | 0.08 | 011 | 001 |
| e | 0.25 | 100 | 10 |

**Fig.8.7.1.a.** Codificare binară

- Se poate uşor verifica faptul că şi codul 2 are **proprietatea de prefix**.

- Diferenţa faţă de cazul precedent este aceea că separarea grupelor de cifre nu se poate face dintr-o dată deoarece lungimea grupelor este variabilă (2 sau 3).

- Astfel secvenţa `1101001` reprezintă tot secvenţa `bcd`.

- **Problema** care se pune, se **formulează** în felul următor:

* Se dă un **set de caractere** precum şi **probabilităţile lor de apariţie**.

* Se cere să se construiască un **cod** cu **proprietatea de prefix**, astfel încât lungimea medie a codului pentru un caracter să fie **minimă**.

* Este evident faptul că un astfel de cod va conduce la **mesaje de lungime minimă**.

- Una din tehnicile de aflare a **codului de prefix optim** este **algoritmul lui Huffmann**.

- Conform algoritmului lui **Huffmann**:
  - (1) Se selectează două caractere `a` şi `b` care au probabilitatea cea mai scăzută de apariţie.
  - (2) Se înlocuiesc cele două caractere cu un singur caracter imaginar (spre exemplu `x`), a cărui probabilitate de apariţie este suma probabilităţilor lui `a` şi `b`.
  - (3) Aplicând iterativ această procedură, se obţine **codul prefix optimal** pentru un set de caractere.
  - (4) Codul pentru setul original de caractere se obţine utilizând codul lui `x` la întâlnirea caracterului `a` sau `b`, căruia i se adaugă un `"0"` pentru `a`, respectiv un `"1"` pentru `b`.

- În această manieră, un **cod prefix** poate fi reprezentat printr-un **arbore binar**, dacă **nodurilor terminale** ale arborelui li se asociază caracterele originale ale alfabetului, iar ramurilor arborelui ponderile `0` sau `1`.

- **Codul unui caracter** poate fi asimilat cu un drum în **arborele binar** respectiv.

- Toate drumurile pornesc de la rădăcină şi se finalizează cu un nod terminal.

- Se consideră că trecerea la **fiul stâng** al unui nod adaugă un `"0"` codului, iar trecerea la **fiul drept**, se adaugă un `"1"`.

- **Secvenţa de cifre binare** rezultată, reprezintă **codul** caracterului respectiv.

- În figura 8.7.1.b apare reprezentarea în forma de **arbore binar** a codului 1 (a) respectiv a codului 2 (b) din figura 8.7.1.a.

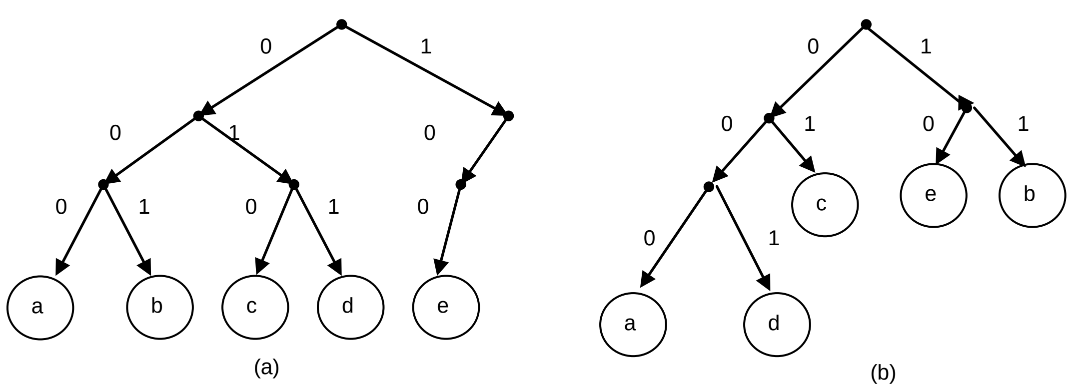


**Fig.8.7.1.b.** Coduri prefix în reprezentare de arbore binar

### 8.7.2. Arbori Huffmann. Implementarea algoritmului lui Huffmann

- În implementarea algoritmului lui **Huffmann** se utilizează o colecţie de **arbori binari speciali** care se bucură de următoarele caracteristici:

    - **Nodurile terminale** sunt caractere.

    - **Rădăcina** oricărui arbore are asociată o valoare care reprezintă suma probabilităţilor de apariţie a caracterelor corespunzătoare tuturor nodurilor terminale ale arborelui respectiv.

        - Această sumă se numeşte **greutate** sau **pondere** a **arborelui**.

- Iniţial **fiecare caracter** este un arbore format dintr-un singur nod.

- La terminarea algoritmului rezultă **un singur arbore** ale cărui noduri terminale conţin toate caracterele alfabetului.

- În acest arbore, **drumul** de la rădăcină la orice nod terminal reprezintă codul caracterului asociat nodului, conform schemei stânga egal `"0"`, dreapta egal `"1"`.

    - Astfel de arbori se numesc **arbori Huffmann**.

- **Construcţia unui arbore Hufmann** se realizează după cum urmează:

    - (1) În fiecare pas, algoritmul lui Hufmann selectează din colecţie doi arbori cu cea mai mică greutate.

    - (2) Cei doi arbori sunt combinaţi într-unul singur, a cărui greutate este egală cu suma greutăţilor celor doi arbori.

        - Combinarea arborilor se realizează, generând o nouă rădăcină care are drept fii rădăcinile arborilor în cauză (ordinea lor **nu** contează).

    - (3) Continuând în aceeaşi manieră, în final se obţine arborele, care pentru probabilităţile date, reprezintă codul cu lungimea medie minimă.

- În fig.8.7.2.a apare reprezentarea grafică a construcţiei unui astfel de arbore Huffmann.

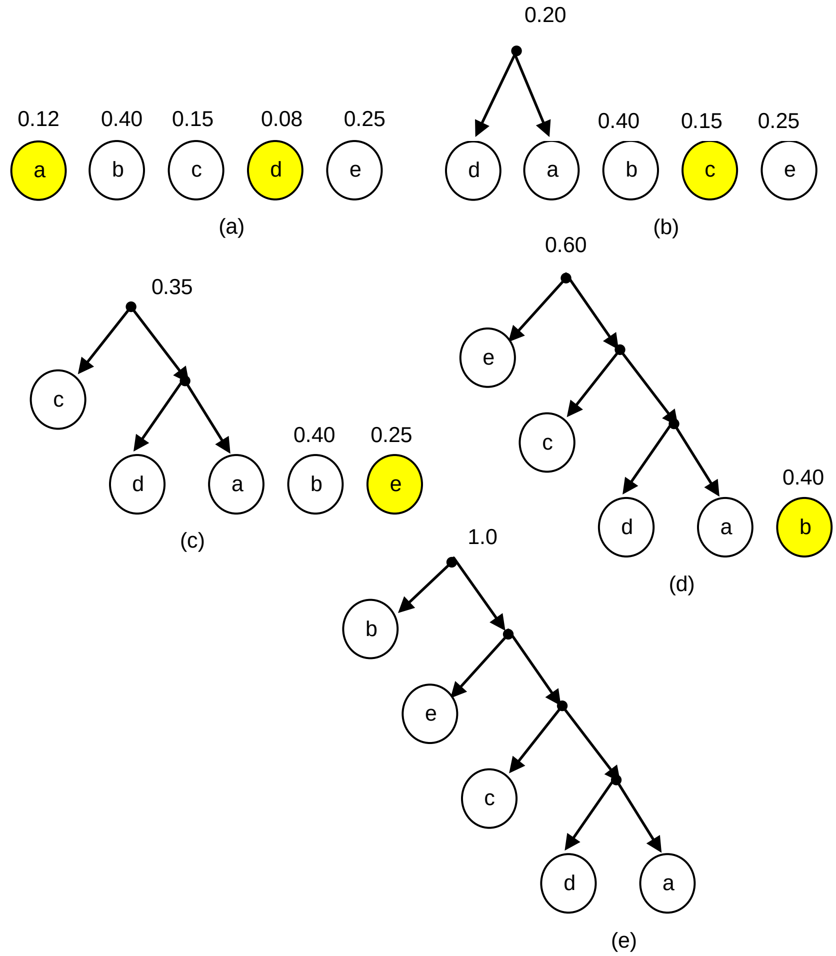


**Fig.8.7.2.a.** Construcţia unui arbore Huffmann

* În continuare se descriu **structurile de date specifice**.

* Pentru reprezentarea arborilor şi implementarea algoritmului lui Huffmann se utilizează trei tablouri: `arbore`,`alfabet` şi `zona` (secvenţa [8.7.2.a]).

```
------------------------------------------------------------
/*Arbori Huffmann  Structuri de date - varianta C*/

typedef struct linie_arbore {
               int fiu_stang;   /*cursor în arbore*/
               int fiu_drept;   /*cursor în arbore*/
               int parinte;     /*cursor în arbore*/
          } linie_arbore;

linie_arbore arbore[MaxNod1];   /*tabloul arbore*/
int ultim_nod;                 /*cursor în arbore*/

typedef struct linie_alfabet {
                char simbol;                    [8.7.2.a]
                float probabilitate;
                int terminal;   /*cursor în arbore*/
          } linie_alfabet;

linie_alfabet alfabet[MaxNod2];  /*tabloul alfabet*/

typedef struct linie_zona {
```

```
            float greutate;
            int radacina;     /*cursor în arbore*/
         } linie_zona;
linie_zona zona[MaxNod3];      /*tabloul zona*/
int ultim_arb;                /*cursor în zona*/
------------------------------------------------------------
```

- Tabloul `arbore`, memorează **structurile arborilor**
  - Fiecare element al tabloului `arbore` se referă la un nod al unui arbore pentru care se păstrează cursorii la fiul său stâng, la cel drept şi la părintele său.
  - Variabila de tip indice (cursor) `ultim_nod`, indică ultimul element ocupat al tabloului.
  - Câmpul `parinte` a fost prevăzut pentru a facilita parcurgerea drumului de la un nod terminal spre rădăcină în vederea determinării codului unui
- **Simbolurile** şi **probabilităţile lor de apariţie** se înregistrează în tabloul `alfabet`.
  - Pentru fiecare simbol se prevede cursorul care indică nodul terminal asociat din tabloul `arbore`.
- Se mai utilizează şi tabloul `zona` pentru precizarea **colecţiei de arbori Huffmann**.
  - Fiecare înregistrare a tabloului `zona` se referă la un arbore şi cuprinde greutatea arborelui şi un cursor la rădăcina sa din tabloul `arbore`.
  - Şi în acest caz se utilizează variabila de tip indice (cursor) `ultimArb` care indică ultimul element ocupat al tabloului `zona`.
- Valorile iniţiale ale câmpurilor celor trei tablouri pentru exemplul anterior apar în figura 8.7.2.b.

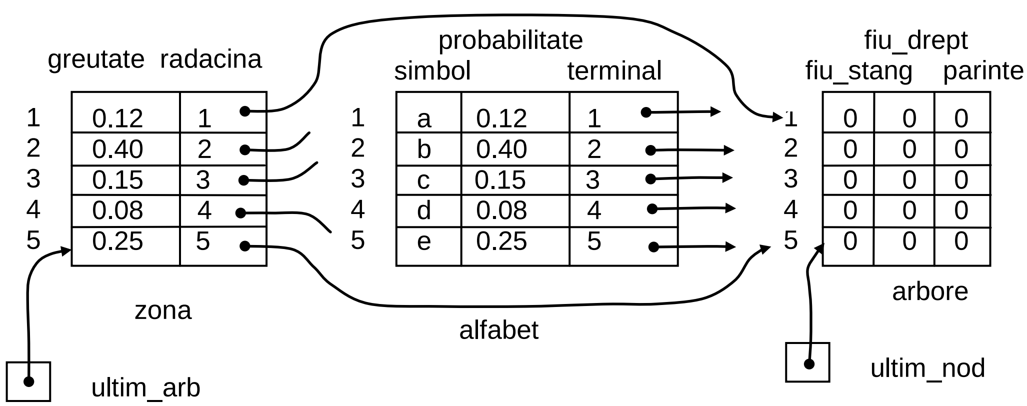


**Fig.8.7.2.b.** Valori iniţiale pentru structurile de date utilizate

- Se presupune că înregistrările de la `1` la `ultim_arb` din `zona` şi cele de la `1` la `ultim_nod` din `arbore` sunt integral ocupate.
- În secvenţele care urmează sunt omise comparaţiile între valorile cursorilor şi limitele maxime ale tablourilor utilizate.
- O primă **formă pseudocod** a **algoritmului lui Huffmann** apare în secvenţa [8.7.2.b].

```
-------------------------------------------------------------
/*Algoritmul lui Huffmann – varianta pseudocod*/

cat timp (există mai mult de un arbore în zona)      [8.7.2.b]
   i=indexul în zona al arborelui cu greutatea minimă;
   j=indexul în zona al arborelui cu greutatea minimă
      următoare;
   *crează în tabloul arbore un nod nou care are ca fiu
       stâng pe zona[i].radacina şi ca fiu drept pe
       zona[j].radacina;
   *înlocuieşte arborele i din zona cu arborele
        a cărui radacină este nodul nou creat şi a cărui
        greutate este zona[i].greutate+zona[j].greutate;
   *suprimă arborele j din zona;
   □
-------------------------------------------------------------
```

- Pentru **rafinarea** acestei forme se descriu:

  - (1) Procedura **`GreutateMinima`** care determină indicii `i` şi `j` din tabloul `zona` ai arborilor cu cele mai mici greutăţi [8.7.2.c].

```
-------------------------------------------------------------
/*Determinarea arborilor minimi – varianta pseudocod*/

 Subprogram GreutateMinima(int min,int min1)

/*poziţionează pe min şi min1 pe arborii cei mai uşori din
  zona. Se presupune ca în zona există cel puţin 2 arbori*/
  int i; /*indice cautare*/
  daca(zona[1].greutate<=zona[2].greutate)
      min=1;
      min1=2;
      □
    altfel                                       [8.7.2.c]
      min=2;
      min1=1;
      □
  pentru(i=3 la ultim_arb)
    daca(zona[i].gretate<zona[min].greutate)
         min1=min;
         min=i;
         □
     altfel
        daca(zona[i].greutate<zona[min1].greutate)
          min1=i;
    □
/*GreutateMinima*/
-------------------------------------------------------------
```

  - (2) Functia **`Creeaza`** care generează un nou arbore in tabloul `arbore` [8.7.2.d].

```
-------------------------------------------------------------
/*Generarea unui nou arbore in tabloul arbore – varianta C*/

int Creeaza(int arb_stang,int arb_drept)
                                            [8.7.2.d]
   ultim_nod= ultim_nod+1; /*arborele nou creat este
                              arbore[ultim_nod]*/
   arbore[ultim_nod].fiu_stang=zona[arb_stang].radacina;
```

```
    arbore[ultim_nod].fiu_drept=zona[arb_drept].radacina;
    arbore[ultim_nod].parinte=0;
    arbore[zona[arb_stang].radacina].parinte=ultim_nod;
    arbore[zona[arb_drept].radacina].parinte=ultim_nod;
    returneaza ultim_nod;
/*Creeaza*/
------------------------------------------------------------
```

- Procedura **Huffmann** care apare în secvenţa [8.7.2.e]

    - Procedura **Huffmann nu** are parametri de intrare sau de ieşire, ea operând asupra unor structuri de date globale.

```
------------------------------------------------------------
/*Construcţia unui arbore Huffmann – varianta pseudocod*/

Subprogram Huffmann;

  int i,j;  /*indicatori pentru arborii cei mai mici*/
  int radnou; /*rădaciana arborelui nou creat*/

  cat timp(ultim_arb>1)
   |GreutateMinima(i,j);                   [8.7.2.e]
   |radnou=Creeaza(i,j);
   |/*se înlocuieşte arborele i din zona cu arborele
   |    a cărui rădacina este radnou*/
   |zona[i].greutate=zona[i].greutate+zona[j].greutate;
   |zona[i].radacina=radnou;
   |/*se suprima arborele j din zona înlocuindu-l cu
   |    ultimul arbore*/
   |zona[j].greutate=zona[ultim_arb].greutate;
   |zona[j].radacina=zona[ultim_arb].radacina;
   |ultim_arb = ultim_arb-1 /*actualizare ultim_arb*/
   |□
/*Huffmann*/
------------------------------------------------------------
```

- După terminarea execuţiei procedurii **Huffmann**, **codul pentru fiecare simbol** se determină după cum urmează:

    - (1) Se caută simbolul în tabloul `alfabet`.

        - Câmpul `terminal` al înregistrării corespunzătoare simbolului, este cursorul înregistrării din tabloul `arbore` care corespunde nodului terminal asociat simbolului.

    - (2) În continuare, în mod repetat, se determină **părintele** `p` al nodului curent `n`, până se ajunge la rădăcina arborelui.

        - În acest scop se utilizează câmpul `parinte` a cărui valoare este un cursor tot în tabloul `arbore`.

    - (3) Pentru fiecare părinte `p`, se verifică dacă nodul curent `n` este fiul său din stânga sau cel din dreapta, memorându-se un `"0"` respectiv un `"1"`.

    - (4) **Secvenţa finală de cifre binare** rezultată reprezintă **codul simbolului** în **ordine inversă**.

## 8.8. Reprezentarea grafică a structurilor arbore

- În cadrul acestui paragraf se abordează problema **reprezentării grafice** a unei **structuri arbore** pe ecranul unui display în **mod caracter**.

- Maniera de afişare este **secvenţială** şi **discretă**.

    - Se afişează şiruri de caractere numai de la stânga la dreapta şi de sus în jos, rând după rând (mod ecran, 25 de rânduri, fiecare a câte 80 de caractere).

- În acest scop:

  - (1) Într-o primă etapă se generează **structura topologică a arborelui** care se doreşte a fi reprezentat.

  - (2) Într-o a doua etapă această structură se transformă într-una reprezentabilă prin afişare în maniera mai sus precizată, determinând coordonatele efective ale nodurilor şi ale conexiunilor care le unesc în aşezarea lor pe ecran.

- Pentru prima etapă, soluţia cea mai potrivită de **generare a unei structuri arbore** este cea bazată pe un **algoritm recursiv**.

- Se va utiliza drept suport al reprezentării **structura arbore binar optim** definită în &8.6.

- În acest acop se redactează funcţia **`Arbore`**`(i,j:index):TipRef` (secvenţa [8.8.c]),

    - Funcţia **`Arbore`**:

        - (1) Porneşte de la matricea `r` care conţine rădăcinile subarborilor binari optimi.

        - (2) Generează **structura de date** corespunzătoare arborelui optim $A_{ij}$.

        - (3) Returnează referinţa la rădăcina **arborelui binar optim** construit.

            - Parametrii `i` şi `j` sunt cei care precizează indicii limită ai nodurilor **arborelui binar optim** a cărui rădăcină este nodul având cheia `k` memorată în `r[i,j]`.

- Se definesc următoarele **tipuri de date** [8.8.a].

```
------------------------------------------------------------
{Reprezentarea grafică a structurilor arbore. Structuri de
date}

  TYPE TipRef=^TipNod;
       TipNod=RECORD
                cheie:string;                   [8.8.a]
                poz:pozLin;
                stang,drept,leg:TipRef
              END;
------------------------------------------------------------
```

- Câmpurile `poz` şi `leg` sunt prevăzute pentru scopuri care vor fi discutate ulterior.

* După cum s-a precizat, funcţia **`Arbore`** are drept punct de pornire matricea `r` care memorează **indicii cheilor radăcinilor subarborilor binari optimi**.

  * **Ideea** algoritmului de generare este următoarea:

    * Pentru arborele binar având rădăcina `k` (valoarea `r[i,j]`), funcţia **`Arbore`** generează **recursiv** subarborele său stâng având drept rădăcină cheia $k_s$ (valoarea `r[i,k-1]`) respectiv subarborele drept având drept rădăcină cheia $k_d$ (valoarea `r[k,j]`).

    * Funcţia realizează de fapt o **traversare în inordine** a nodurilor arborelui binar optim care se construieşte, în speţă a arborelui $A_{on}$.

      * În consecinţă cheile nodurilor vor fi parcurse în **ordine alfabetică**.

    * Funcţia **`Arbore`** contorizează numărul nodurilor generate în variabila globală `k`.

      * Cel de-al `k`-lea nod generat, este atribuit celei de-a `k`-a chei.

    * Deoarece **numărul total de chei** este cunoscut şi **cheile sunt ordonate alfabetic**, coordonata orizontală `poz` a fiecărei chei în cadrul liniei de ecran pe care va fi afişată, se poate determina simplu înmulţind pe `k` cu valoarea unui **factor de scară** (`dimensiunea în caractere a rândului/număr total de chei`).

      * Această coordonată se memorează în câmpul `poz` al fiecărui nod pe măsură ce nodurile sunt create (vezi secvenţa [8.8.a]).

    * Se precizează de asemenea faptul că:

      * Cuvintele cheie sunt de tip **şir de caractere** (string).

      * Cuvintele cheie sunt memorate în tabloul predefinit de caractere `chei`, în ordine alfabetică, indicele de intrare în tabel fiind chiar valoarea `k`.

* Afişarea arborelui este realizată de către procedura **`AfiseazaArbore`** (secvenţa [8.8.c]).

* Procedura **`AfiseazaArbore`** utilizează drept **intrări**:

    * Setul de `n` cuvinte cheie memorate în tabloul `chei`.

    * Structura arbore binar optim generată de funcţia **`Arbore`**`(0,n)`.

* În etapa întâi, se generează **structura preliminară a arborelui de reprezentat** (secvenţa [8.8.b]) în care:

    * (1) Coordonata orizontală a fiecărui nod (cheie) a arborelui de afişat este înregistrată în câmpul `poz`.

    * (2) Coordonata verticală se va determina **implicit** în momentul afişării, funcţie de nivelul nodului în cadrul structurii arborelui.

```
------------------------------------------------------------
{Generarea unei structuri de arbore binar optim}

     k:=0; radacina:=Arbore(0,n);                    [8.8.b]
------------------------------------------------------------
```

* În continuare se poate aborda etapa a doua şi anume, **afişarea arborelui** pe ecranul monitorului.
* Acest lucru se realizează plecând de la rădăcină în jos, prin prelucrarea în fiecare pas a unui rând (nivel) de noduri al arborelui.
    * Se realizează de fapt o parcurgere **prin cuprindere** a arborelui realizată prin **metoda celor două cozi**.
* Pentru a realiza accesul la nodurile unui rând (nivel) al structurii de arbore, se utilizează câmpul `leg` precizat în secvenţa [8.8.a].
    * Nodurile care trebuiesc afişate în rândul curent sunt înlănţuite prin câmpul `leg` în lista `curent`. Lista `curent` este **prima coadă** utilizată în traversare.
    * Pe parcursul prelucrării nodurilor listei `curent`, se identifică descendenţii fiecărui nod şi se alcătuieşte cu aceştia o a doua listă numită `urm`, înlănţuirile realizându-se tot prin câmpul `leg`. Lista `urm` este cea de-a **doua coadă**.
    * Când se trece la nivelul următor coada (lista) `urm` devine coada `curent` şi se iniţializează noua coadă (listă) `urm`.
    * Aşa cum s-a precizat, listele `curent` respectiv `urm` sunt de fapt **structuri de date coadă**, iar **parcurgerea prin cuprindere** este realizată în baza tehnicii prezentate la &8.2.5.2.
* **Detaliile** de implementare ale algoritmului de afişare apar în secvenţa [8.8.c].
* Se impun următoarele precizări:
    1. Lista `urm` care conţie nodurile nivelui următor al structurii de arbore, se generează prin tehnica inserţiei în faţă, nodul cel mai din stânga devenind astfel ultimul nod al listei.
    2. În vederea parcurgerii, lista trebuie **inversată**, activitate care se realizează în momentul în care lista `urm` devine lista `curent`.
    3. Pentru fiecare cheie (nod) din lista `curent` care urmează a fi afişată, se determină şi se afişează **conexiunile** pe stânga şi pe dreapta în forma unor segmente orizontale.
    4. Variabilele `u1,u2,u3` şi `u4` precizează poziţiile de început şi de sfârşit ale segmentelor din stânga, respectiv din dreapta unui nod, reprezentate ca o succesiune de caractere ”linie orizontală”.
    5. Afişarea nodurilor din linia curentă este precedată de afişarea pentru fiecare nod a unei linii verticale formate din trei segmente care marchează legătura dintre niveluri.
* Programul **`ReprezentareArbore`** [8.8.c] este destinat prelucrării unor texte sursă `Pascal`.
* În cadrul structurii programului se definesc următoarele proceduri şi funcţii:
    * Funcţia **`DrumArbEch`**`(i,j:index):integer`;
        * Generează în manieră recursivă matricea `r` corespunzătoare **arborelui perfect echilibrat** care poate fi construit utilizând cele `n` chei date şi returnează **lungimea drumului** acestui arbore.
        * Arborele este precizat prin indicii `i` şi `j` ai nodurilor sale extreme.
        * Se utilizează următorul **procedeu**: întrucât tabloul `chei` este un tablou ordonat, pentru fiecare apel al funcţiei, se alege pe post de rădăcină a arborelui curent, indicele `k` al **cheii mediane** a intervalului delimitat de

indicii `i` şi `j`, indice care se memorează în `r[i,j]`.

- În continuare, se determină lungimea drumului prin apelul recursiv al funcţiei **`DrumArbEch`** pentru subarborele stâng, respectiv pentru cel drept, după care se adaugă ponderea `w[i,j]` conform formulei [8.6.1.f].

- Procedura **`ArbOpt`** - construieşte **arborele binar optim** pornind de la distribuţia `w` a ponderilor nodurilor.
    - De fapt procedura completeză matricea `r` cu indicii cheilor care reprezintă rădăcinile subarborilor optimi şi matricea `p` cu lungimile corespunzătoare ale drumurilor asociate.
- Procedura **`AfiseazaArbore`** realizează afişarea efectivă a structurii arborelui şi are drept parametri de intrare indicii `i` şi `j` care delimitează arborele de afişat.
    - În cadrul procedurii **`AfiseazaArbore`** se defineşte funcţia **`Arbore`**, utilizată la generarea structurii arborelui ce urmează a fi afişat pornind de la matricea `r` corespunzătoare.
    - Ambele proceduri au fost descrise anterior.
- Programul principal.

- Mersul **programului principal** este următorul.
- (1) În prima parte a programului principal:
    - Se iniţializează tabelul cuvintelor cheie şi contoarele memorate în tablourile `a` şi `b`.
    - Se citeşte textul sursă (un program `Pascal`) de la dispozitivul de intrare.
    - Pe măsură ce este citit textul, sunt recunoscuţi identificatorii şi cuvintele cheie, actualizându-se contoarele de frecvenţă $a_i$ şi $b_j$ (bucla **`repeat`**).
    - $a_i$ se referă la cuvintele cheie $k_i$ iar $b_j$ la identificatorii situaţi între $k_j$ şi $k_{j+1}$.
    - În continuare, pentru fiecare cuvânt cheie se afişează frecvenţele de acces $b_{i-1}$ şi $a_i$.
    - Se afişează de asemenea o statistică a acestora, respectiv suma frecvenţelor de acces pentru $a_i$ şi $b_j$.
- (2) În cea de-a doua parte a programului:
    - Pornind de la frecvenţele de acces se calculează matricea `w` a ponderilor.
- (3) În cea de-a treia parte a programului:
    - Se apelează funcţia **`DrumArbEch`** care construieşte matricea `r` memorând rădăcinile subarborilor corespunzători **arborelelui binar perfect echilibrat** cu limitele `0` şi `n`, căruia îi calculează şi lungimea drumului. Este vorba despre arborele binar perfect echilibrat al **cuvintelor cheie**.
    - După aceasta se tipăreşte lungimea medie a drumului ponderat şi se afişează **arborele binar perfect echilibrat** prin apelul procedurii **`AfiseazaArbore`**.
- (4) În cea de-a patra parte a programului:
    - Se apelează procedura **`ArbOpt`** care generează **arborele binar optim** pornind de la matricea `w`.

- Se calculează şi se afişează pentru acest arbore lungimea medie a drumului după care este afişat arborele prin apelul procedurii **AfiseazaArbore**.

- (5) În final:

  - Se recalculează matricea `w` luând în considerare de această dată numai frecvenţele de acces la chei $a_i$, cu alte cuvinte frecvenţele $b_j$ se consideră egale cu `0`.

  - Se apelează din nou procedura **ArbOpt** pentru a determina arborele binar optim pentru această situaţie.

  - Se afişează alura arborelui astfel determinat prin apelul procedurii de afişare **AfiseazaArbore**.

- Experimentele arată că de regulă în acest context:

  - (1) Arborele **perfect echilibrat nu** poate fi considerat nici pe departe optim.

  - (2) Frecvenţele identificatorilor care **nu** sunt cuvinte cheie influenţează în mod decisiv structura arborelui optim care se construieşte.

------------------------------------------------------------

```
PROGRAM ReprezentareArbore;

CONST n=31; {numar chei}
      lch=10; { lungime maximă cheie}

TYPE  index=0..n;
      alfa=string[lch];

VAR ch:char;
    k1,k2:integer;
    id:alfa; {identificator sau cheie}              [8.8.c]
    chei:ARRAY[1..n] OF alfa;
    i,j,k:integer;
    a:ARRAY[1..n] OF integer;
    b:ARRAY[index] OF integer;
    p,w:ARRAY[index,index] OF integer;
    r:ARRAY[index,index] OF index;
    suma,sumb:integer;
    litere:SET OF char;
    cifre:SET OF char;

FUNCTION DrumArbEch(i,j:index):integer;
{Generează în manieră recursivă matricea r corespunzătoare
arborelui perfect echilibrat care poate fi construit
utilizând cele n chei date şi returnează lungimea drumului
acestui arbore}

  VAR k:integer;
  BEGIN
    k:=(i+j+1) DIV 2;
    r[i,j]:=k; {k este cheia mediană}
    IF i>=j THEN
        DrumArbEch:=b[k]
      ELSE
        DrumArbEch:= DrumArbEch(i,k-1)+ DrumArbEch(k,j)
                              +w[i,j]
  END; {DrumArbEch}
```

```
                                                    [8.8.c]
PROCEDURE ArbOpt;
{Construieşte arborele binar optim pornind de la matricea w a
ponderilor nodurilor. De fapt se generează matricea r cu
indicii cheilor care reprezintă rădăcinile subarborilor
optimi şi matricea p cu lungimile corespunzătoare ale
drumurilor asociate}

    VAR x,min:integer;
        i,j,k,h,m:index;
    BEGIN {intrare:w; iesire:p,r}
      {construcţie arbori de lăţime nulă(h=0)}
      FOR i:=0 TO n DO
        p[i,i]:=w[i,i];
      {construcţie arbori de lăţime arbore=1 (h=1)}
      FOR i:=0 TO n-1 DO
        BEGIN
          j:=i+1;
          p[i,j]:=p[i,i]+p[j,j]+w[ij];
          r[i,j]:=j
        END;
      {construcţie arbori de lăţime arbore>1}
      FOR h:=2 TO n DO    {h=lăţimea arborelui considerat}
        FOR i:=0 TO n-h DO    {i=indexul stâng al arborelui}
          BEGIN               {j=indexul sau drept}
            j:=i+h;
            m:=r[i,j-1]; min:=p[i,m-1]+p[m,j];
            FOR k:=m+1 TO r[i+1,j] DO
              BEGIN
                x:=p[i,k-1]+p[k,j];
                IF x<min THEN
                  BEGIN
                    m:=k;
                    min:=x
                  END
              END;
            p[i,j]:=min+w[i,j]; {pondere arbore optim}
            r[i,j]:=m           {rădăcină arbore optim}
          END
    END; {ArbOpt}

PROCEDURE AfiseazaArbore;
{Realizează afişarea efectivă a structurii arborelui şi are
drept parametri de intrare indicii i şi j care delimitează
arborele de afişat}

  CONST ll=80; {lăţime linie de afişat}

  TYPE  ref=^nod;
        pozLin=0..ll;
        nod=RECORD
              cheie:alfa;
              poz:pozLin;
              sting,drept,leg:ref
            END;

  VAR radacina,curent,urm:ref;
      q,q1,q2:ref;
```

```
    i,k:integer;
    u,u1,u2,u3,u4:pozLin;

FUNCTION Arbore(i,j:index):ref;
 {Generează structura de date corespunzătoare arborelui
  optim. Porneşte de la matricea r care conţine rădăcinile
  subarborilor binari optimi si returnează referinţa la
  rădăcina arborelui optim construit}

  VAR p:ref;
  BEGIN
    IF i=j THEN p:=NIL                              [8.8.c]
      ELSE
        BEGIN
          new(p);
          p^.sting:=Arbore(i,r[i,j]-1);
          p^.poz:=(((ll-lch)*k) DIV (n-1)) + (lch DIV 2);
          k:=k+1;
          p^.cheie:=chei [r[i,j]];
          p^.drept:=Arbore(r[i,j],j)
        END;
    Arbore:=p
  END; {Arbore}

BEGIN {AfiseazaArbore}
  k:=0; radacina:=Arbore(0,n);
  curent:=radacina;
  radacina^.leg:=NIL;
  urm:=NIL;
  WHILE curent<>NIL DO
    BEGIN
      {se afişează liniile verticale de legătură între
       niveluri pentru toate cuvintele din linia curentă}
      FOR i:=1 TO 3 DO
        BEGIN
          u:=0; q:=curent;
          REPEAT
            u1:=q^.poz;
            REPEAT
              Write(' '); u:=u+1
            UNTIL u=u1;
            Write('I'); u:=u+1; q:=q^.leg
          UNTIL q=NIL;
          WriteLn
        END;
     {se afişează linia curentă; se determină descendenţii
       nodurilor din lista curent şi se formează lista
       rândului următor urm}
      q:=curent; u:=0;
      REPEAT
        i:=lch;
        WHILE q^.cheie[i]=' ' DO i:=i-1; {lungime cheie}
        u2:=q^.poz-((i-1) DIV 2); u3:=u2+i;
        q1:=q^.sting; q2:=q^.drept;
        IF q1=NIL THEN
            u1:=u2
          ELSE
            BEGIN
```

```
              u1:=q1^.poz; q1^.leg:=urm; urm:=q1
            END;
          IF q2=NIL THEN
            u4:=u3
          ELSE
            BEGIN
              u4:=q2^.poz+1; q2^.leg:=urm; urm:=q2
            END;
          i:=0;
          WHILE u<u1 DO BEGIN Write(' '); u:=u+1  END;
          WHILE u<u2 DO BEGIN Write('-'); u:=u+1  END;
          WHILE u<u3 DO
            BEGIN
              i:=i+1; Write(q^.cheie[i]); u:=u+1
            END;
          WHILE u<u4 DO BEGIN Write('-'); u:=u+1  END;
          q:=q^.leg
        UNTIL q=NIL;
        WriteLn;
        {se inversează lista urm şi se face curentă}
        curent:=NIL;
        WHILE urm<>NIL DO
          BEGIN
            q:=urm; urm:=q^.leg;
            q^.leg:=curent; curent:=q
          END                                  [8.8.c]
      END {WHILE}
  END; {AfiseazaArbore}

BEGIN {Programul principal - ReprezentareArbore}
  {se iniţializează static tabela de chei}
  chei[ 1]:='ARRAY     '; chei[ 2]:='BEGIN     ';
  chei[ 3]:='CASE      '; chei[ 4]:='CONST     ';
  chei[ 5]:='DIV       '; chei[ 6]:='DOWNTO    ';
  chei[ 7]:='DO        '; chei[ 8]:='ELSE      ';
  chei[ 9]:='END       '; chei[10]:='FILE      ';
  chei[11]:='FOR       '; chei[12]:='FUNCTION  ';
  chei[13]:='GOTO      '; chei[14]:='IF        ';
  chei[15]:='IN        '; chei[16]:='LABEL     ';
  chei[17]:='MOD       '; chei[18]:='NIL       ';
  chei[19]:='OF        '; chei[20]:='PROCEDURE ';
  chei[21]:='PROGRAM   '; chei[22]:='RECORD    ';
  chei[23]:='REPEAT    '; chei[24]:='SET       ';
  chei[25]:='THEN      '; chei[26]:='TO        ';
  chei[27]:='TYPE      '; chei[28]:='UNTIL     ';
  chei[29]:='VAR       '; chei[30]:='WHILE     ';
  chei[31]:='WITH      ';
  FOR i:=1 TO n DO {se iniţializează contoarele a şi b}
    BEGIN
      a[i]:=0; b[i]:=0
    END;
  b[0]:=0; k2:=lch;
  litere:=['a'..'z', 'A'..'Z'];
  cifre:=['0'..'9'];
  {se balează textul de intrare, se identifică cheile şi
   identificatorii şi se determină a şi b}
  REPEAT
    Read(ch);
```

```
  IF ch IN litere THEN
      BEGIN {identificator sau cheie}
        k1:=0;
        REPEAT
          IF k1<lch THEN
            BEGIN
              k1:=k1+1; id[k1]:=ch
            END;
          Read(ch)
        UNTIL NOT((ch IN litere) OR (ch IN cifre));
        IF k1>=k2 THEN
            k2:=k1
          ELSE
            REPEAT
              id[k2]:=' '; k2:=k2-1
            UNTIL k2=k1;
        i:=1; j:=n;
        REPEAT
          k:=(i+j) DIV 2;
          IF chei [k]<=id THEN i:=k+1;
          IF chei [k]>=id THEN j:=k-1
        UNTIL i>j;
        IF  chei [k]=id THEN
            a[k]:=a[k]+1
          ELSE
            BEGIN
              k:=(i+j) DIV 2; b[k]:=b[k]+1
            END
      END
    ELSE
      IF ch='''' THEN
          REPEAT
            Read(ch)
          UNTIL ch=''''                        [8.8.c]
        ELSE
          IF ch='{' THEN
            REPEAT
              Read(ch)
            UNTIL ch='}'
UNTIL ch='$'; {caracter sfârşit text sursă}
{pentru fiecare cuvânt cheie se afişează frecvenţele a şi
 b}
WriteLn('Cuvintele cheie şi frecvenţele lor de
          apariţie');
suma:=0; sumb:=b[0];
FOR i:=1 TO n DO
  BEGIN
    suma:=suma+a[i]; sumb:=sumb+b[i];
    WriteLn('    ',b[i-1],'    ',a[i],'    ',chei[i],
          chei[i,1],chei[i,lch])
  END;
{se afişează suma frecvenţelor de acces pentru a şi b}
WriteLn('    ',b[n]);
WriteLn('   ------  ------');
WriteLn('    ',sumb,'    ',suma);
{se calculează matricea w din a şi b}
FOR i:=1 TO n DO
  BEGIN
```

```
      w[i,i]:=b[i];
      FOR j:=i+1 TO n DO
        w[i,j]:=w[i,j-1]+a[j]+b[j]
    END;
  {se construieşte şi se afisează arborele perfect
  echilibrat cu limitele 0 şi n}
  WriteLn;WriteLn;
  WriteLn('    lungimea medie a drumului arborelului
    echilibrat=',conv(DrumArbEch(0,n))/conv(w[0,n]));
  AfiseazaArbore;
  {se construieşte şi se afisează arborele optim cu limitele
   0 şi n}
  ArbOpt;
  WriteLn;WriteLn;
  WriteLn('    lungimea medie a drumului arborelui
    Optim =',conv(p[0,n])/conv(w[0,n]));
  AfiseazaArbore;
  {se recalculează w considerând numai cuvintele-cheie,
   adică făcând b=0}
  FOR i:=0 TO n DO
    BEGIN
      w[i,i]:=0;
      FOR j:=i+1 TO n DO
        w[i,j]:=w[i,j-1]+a[j]
    END;
  {se construieşte şi se afisează arborele optim cu limitele
   0 şi n care nu conţine decât cuvinte cheie}
  ArbOpt;                                                 [8.8.c]
  WriteLn;WriteLn;
  WriteLn('    arborele optim considerând numai
    cuvintele-cheie');
  AfiseazaArbore
END.{ReprezentareArbore}
------------------------------------------------------------
```

- Rezultatele execuţiei acestui program apar în figurile următoare după cum urmează.
    - În figura 8.8.a apare reprezentarea grafică a **arborelui binar perfect echilibrat** corespunzător **cuvintelor cheie**.
    - În figura 8.8.b apare reprezentarea grafică a **arborelui binar optim** pentru **cuvintele cheie şi identificatorii din programul sursă** [8.8.c.].
    - În figura 8.8.c apare reprezentarea grafică a aceluiaşi **arbore binar optim** considerând **numai cuvintele cheie**.

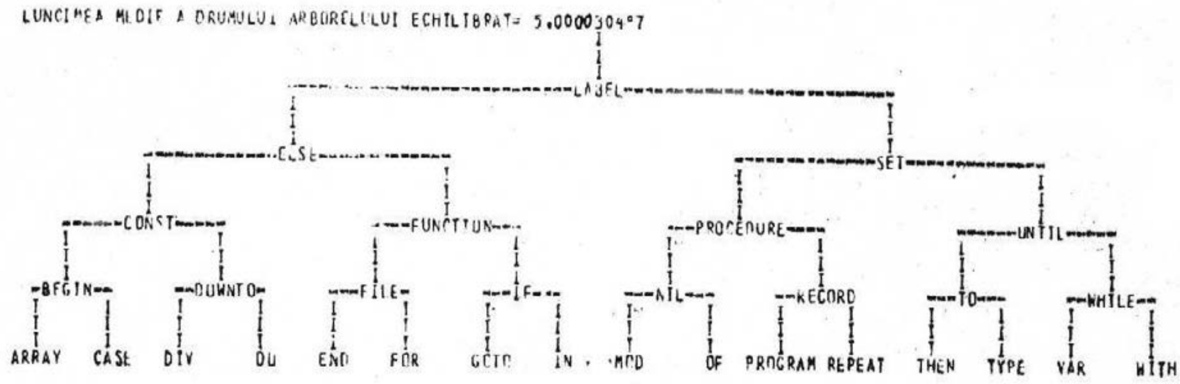


**Fig.8.8.a**. Reprezentarea grafică a arborelui binar perfect echilibrat corespunzător cuvintelor cheie

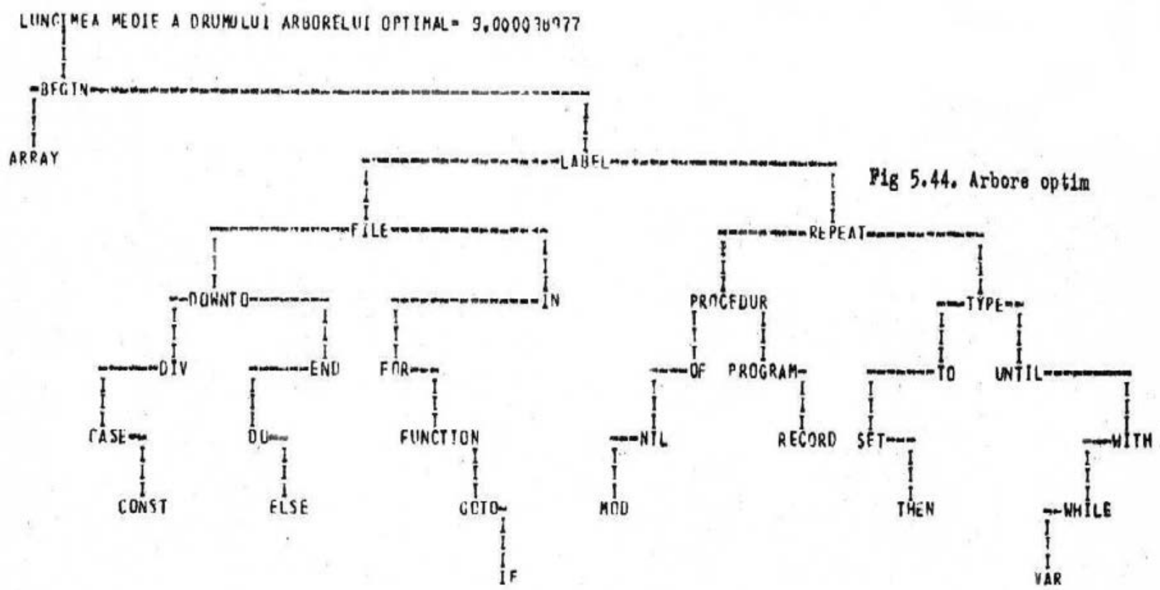


**Fig.8.8.b**. Reprezentarea grafică a arborelui binar optim

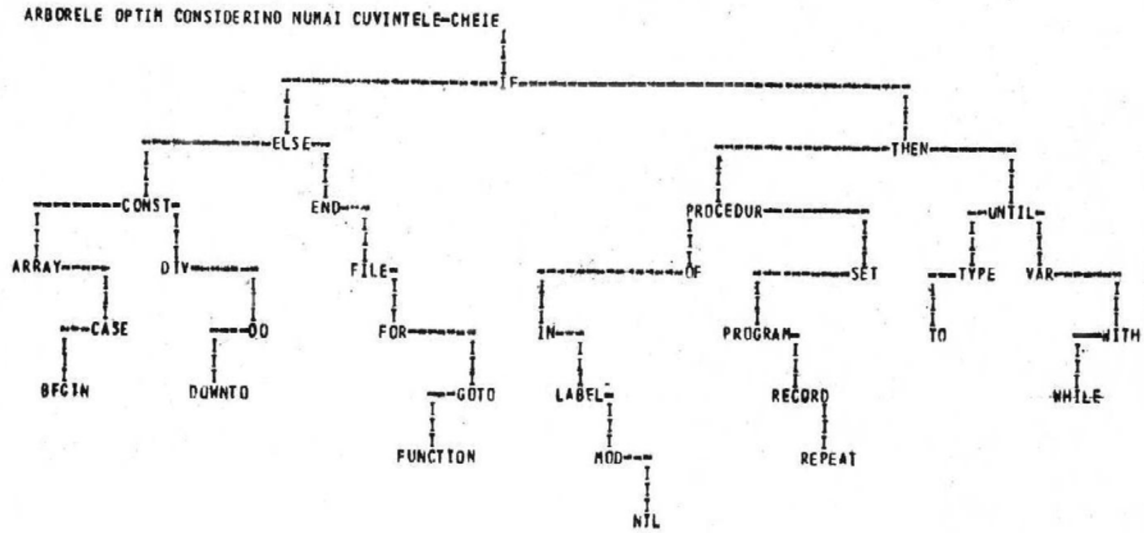


**Fig.8.8.c**. Reprezentarea grafică a arborelui binar optim considerând numai cuvintele cheie