

Logică și structuri discrete

Arbori

Casandra Holotescu

casandra@cs.upt.ro

<https://tinyurl.com/lecturesLSD>

Arbori. Definiție

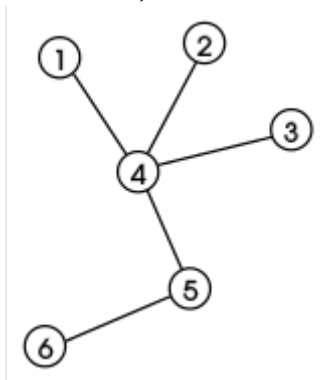
Arbori

Un arbore e un *graf conex fără cicluri*.

conex = drum între orice 2 noduri (din 1 sau mai mulți pași)

E compus din *noduri* și *ramuri* (muchii).

⇒ un arbore cu n noduri are $n - 1$ ramuri
(demonstrăm prin *inducție*)



Un arbore cu n noduri are $n-1$ muchii

Demons. prin inducție: $P(1)$: $n = 1$ e trivial (un nod fără muchii)

$P(n-1) \Rightarrow P(n)$:

Fie mulțimea *tuturor drumurilor* într-un arbore cu $n > 1$ noduri.

Cum arborele e *aciclic*, un drum are doar noduri *distincte* (altfel, un drum $v_i \dots v_j$ cu $v_i = v_j$ e un ciclu).

\Rightarrow un drum are cel mult n noduri \Rightarrow numărul de drumuri e *finit*

\Rightarrow există un drum de *lungime maximă* $v_{i_1} v_{i_2} \dots v_{i_k}$

Atunci, v_{i_1} e legat doar de v_{i_2} , altfel, $v_{alt} v_{i_1} v_{i_2} \dots v_{i_k}$ ar fi mai lung!

Stergem nodul v_{i_1} și muchia (v_{i_1}, v_{i_2}) . Graful obținut rămâne conex (niciun drum în graful inițial nu are v_{i_1} ca nod interior).

E evident și aciclic (nu am adăugat muchii noi), deci e un arbore.

Din ipoteza de inducție, având $n - 1$ noduri, are $n - 2$ muchii, deci graful inițial avea cu un nod și o muchie în plus, q.e.d.

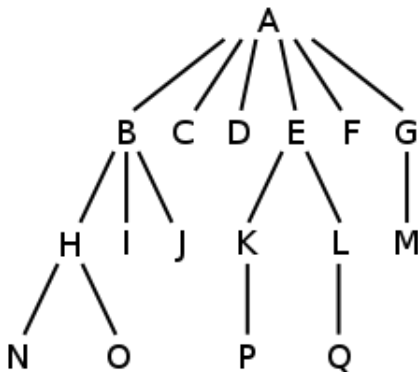
Arbore cu rădăcină

De obicei identificăm un nod anume numit *rădăcina*, și *orientăm* muchiile în același sens față de rădăcină

Orice nod în afară de rădăcină are un unic *părinte*

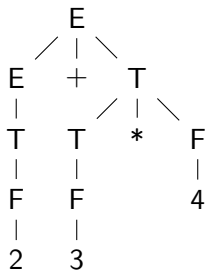
Un nod poate avea mai mulți *copii* (*fii*)

Nodurile fără copii se numesc noduri *frunză*



Arbori în informatică

Arborii sunt un mod natural de a reprezenta structuri *ierarhice*
sistemul de fișiere (subarborii sunt cataloagele)
arborele sintactic într-o gramatică (ex. expresie)
ierarhia de clase în programarea orientată pe obiecte
fișierele XML (elementele conțin alte elemente)



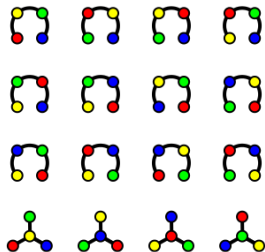
```
<order>
  <item>
    <title="Data Structures"/>
    <price="24.99"/>
  </item>
  <item>
    <title="Mathematical Logic"/>
    <price="39.99"/>
  </item>
</order>
```

Arbori ordonați și neordonați

Ordinea dintre copii poate conta (ex. arbore sintactic) sau nu



Arborii neordonați
cu 2 – 4 noduri:



Există n^{n-2} arbori neordonați cu n noduri (*formula lui Cayley*)

Formula lui Cayley

Există n^{n-2} arbori neordonați cu n noduri.

Demonstr.: Fie un arbore neordonat cu n noduri, etichetate de la 1 la n .

Reprezentăm arborele ca șir de $n - 2$ numere de la 1 la n , astfel (citiți opțional despre *codul Prüfer*):

- ▶ ștergem din arbore frunza etichetată cu numărul *cel mai mic*
- ▶ adăugăm la cod numărul (eticheta) părintelului/vecinului său
- ▶ repetăm până rămân doar 2 noduri în arbore

Secvența obținută este *codul Prüfer* al arborelui, și este *unică* pentru fiecare arbore *neordonat* de n noduri.

Pe fiecare poziție a codului Prüfer putem avea un număr între 1 și n .

Există $n - 2$ poziții

⇒ n^{n-2} coduri Prüfer *diferite*

⇒ n^{n-2} arbori neordonați *diferiți*.

Codul Prüfer – exemplu

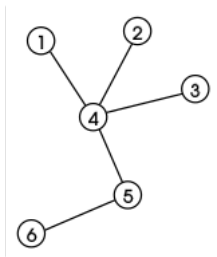


Image: http://en.wikipedia.org/wiki/File:Tree_graph.svg

Pentru arborele de mai sus, *codul Prüfer* este: 4, 4, 4, 5

ștergem frunza 1 (nodul etichetat cu nr. minim)

adăugăm la cod nr. părintelui său: 4

ștergem frunza 2 (urm. nr. minim), adăugăm nr. părintelui: 4, 4

ștergem frunza 3 (urm. nr. minim), adăugăm nr. părintelui: 4, 4, 4

4 a devenim frunză și e nodul cu nr. minim, deci e șters

adăugăm la cod nr. părintelui său: 4, 4, 4, 5

au mai rămas doar 2 noduri, deci am obținut *codul Prüfer*.

Arbori – structuri recursive

Arborele definit recursiv

Un arbore e un *nod* cu 0 sau mai mulți *subarbori*

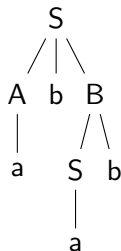
⇒ o *listă* de subarbori (frunzele au lista vidă)

În funcție de problemă, nodurile conțin *informație*

Toate nodurile au informație de același *tip*

```
type 'a tree = T of 'a * 'a tree list
```

```
let t = T('S', [  
    T('A', [  
        T('a', [])]);  
    T('b', []);  
    T('B', [  
        T('S', [  
            T('a', [])]);  
        T('b', [])]);  
])
```



Definiție recursivă: cu arbore vid

Definiția dată: bună când arborele totdeauna *există* (ex.: expresie)

Uneori, arborele *poate fi vid* (ex.: pentru reprezentarea de mulțimi).

Putem defini atunci:

Un arbore e fie arborele *vid*

fie un *nod* cu mai mulți *subarbori*

⇒ extindem tipul anterior cu o valoare pentru arborele vid

$$tip_nou = tip_vechi \cup \{valoare-specială\}$$

Definiție recursivă: cu arbore vid

```
type 'a option = None | Some of 'a
```

indică în ML o valoare de tipul 'a care poate eventual lipsi

⇒ lucrăm cu valori de tipul 'a tree option

None sau Some t, cu t de tip 'a tree definit anterior:

```
type 'a tree = T of 'a * 'a tree list
```

```
type 'a tree option = None | Some of 'a tree
```

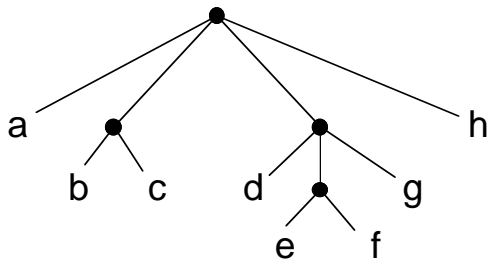
```
let f = function (* parametru arbore, vid sau nu *)  
  | None -> (* cazul de prelucrare pentru arborele vid *)  
  | Some T(r, tl) -> (* radacina r, lista de copii tl *)
```

Arbori neetichetați

Uneori, nu avem informație utilă decât în nodurile frunză:

⇒ reprezentăm explicit varianta de nod frunză

type 'a tree = L of 'a | T of 'a tree list



⇒ arborele e echivalent cu o *listă ierarhică* (listă de liste)

[a, [b, c], [d, [e, f], g], h]

dar o listă de liste trebuie să fie uniformă pe nivele (același tip)

T [L 'a'; T [L 'b'; L 'c'];

T [L 'd'; T [L 'e'; L 'f']; L 'g']; L 'h']

Arbori binari

Arbori binari

Într-un arbore binar, fiecare nod are cel mult doi copii,
identificați ca fiul stâng și fiul drept (oricare/ambii pot lipsi)

⇒ un arbore binar e fie: arborele vid
un nod cu cel mult doi subarbori

```
type 'a bintree = Nil | T of 'a bintree * 'a * 'a bintree
```

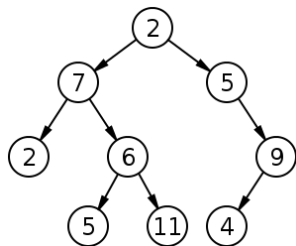
Instanțiind pentru noduri întregi:

```
type inttree = Nil | T of inttree * int * inttree
```


Arbori binari

```
type inttree = Nil | T of inttree * int * inttree
```

Un arbore binar de înălțime n are cel mult $2^{n+1} - 1$ noduri



subarborele stâng:

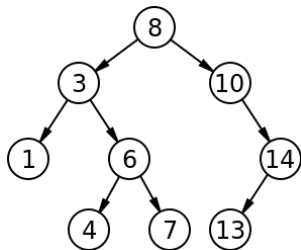
```
T (T(Nil, 2, Nil), 7,  
   T(T(Nil, 5, Nil), 6, T(Nil, 11, Nil)))
```

subarborele drept:

```
T (Nil, 5, T(T(Nil, 4, Nil), 9, Nil))
```

Arbori binari de căutare

Memorează valori sortate *în ordine*



Pentru fiecare nod,
relativ la valoarea din rădăcină:

subarborele *stâng* are valori *mai mici*

subarborele *drept* are valori *mai mari*

Căutarea se face *recursiv*, comparând mereu elementul căutat cu *rădăcina* subarborelui curent:

dacă sunt egale \Rightarrow am găsit elementul în arbore

dacă $e <$, se continuă căutarea în subarborele stâng

dacă $e >$, în subarborele drept

Pot fi folosiți pentru a reprezenta *mulțimi*

Arbori binari de căutare

Căutarea: recursiv în subarboarele potrivite:

```
let bsearch x = (* cauta x in arbore *)
  let rec srchx = function (* x fixat mai sus *)
    | Nil -> false
    | T (left, v, right) ->
      v = x || srchx (if x < v then left else right)
  in srchx
```

Arbori strict binari

engl. strictly binary tree, proper binary tree (binar propriu-zis)

Fiecare nod care nu e frunză are *exact* doi copii

de exemplu, un arbore pentru expresii cu operanzi binari

`type 'a bintree = L of 'a | T of 'a bintree * 'a * 'a bintree`

dacă avem același tip în frunze și celelalte noduri

Arbore strict binar cu n frunze $\Rightarrow n-1$ noduri ce nu sunt frunze

Un arbore strict binar de înălțime n are cel mult 2^n frunze

Arbori - parcurgeri

Parcurgerea arborilor

în *preordine*: rădăcina, subarborele *stâng*, subarborele *drept*

în *inordine*: subarborele *stâng*, rădăcina, subarborele *drept*

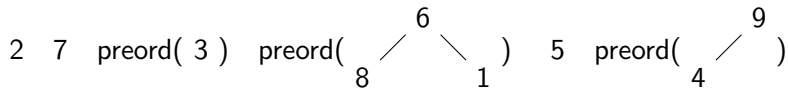
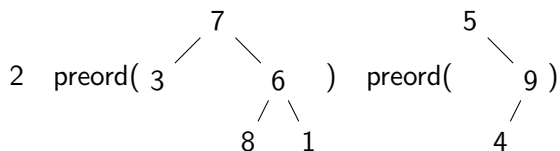
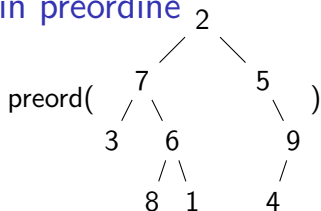
în *postordine*: subarborele *stâng*, subarborele *drept*, rădăcina

subarborii se parcurg și ei tot în ordinea dată (pre/in/post ordine)!

Pentru expresii, obținem astfel formele prefix, infix și postfix

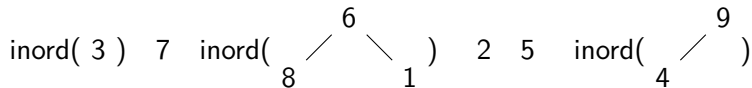
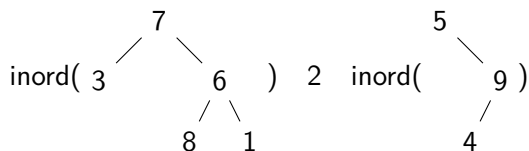
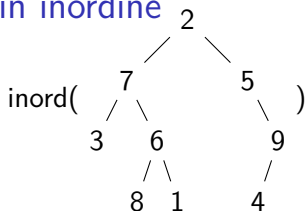
Parcurgerea în pre-/ post-ordine e definită la fel pentru orice arbori (nu doar binari).

Exemplu: parcurgere în preordine



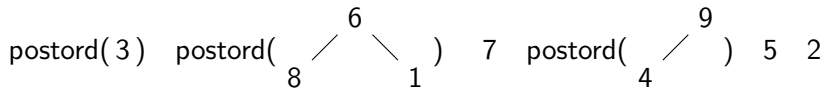
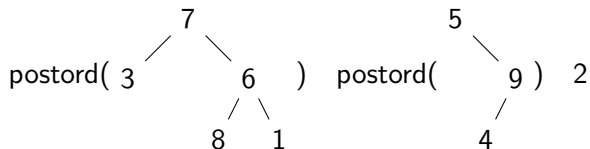
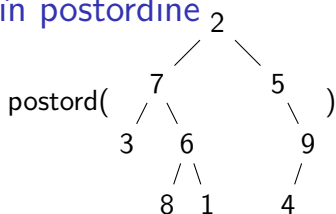
2 7 3 6 8 1 5 9 4

Exemplu: parcurgere în inordine



3 7 8 6 1 2 5 4 9

Exemplu: parcurgere în postordine



3 8 1 6 7 4 9 5 2

Parcurgeri pentru arbori de expresii

Parcurgere în *preordine* \Rightarrow expresii *prefix*

$$\text{pre}\left(\begin{array}{c} * \\ / \backslash \\ - \quad 5 \\ / \backslash \\ 2 \quad + \\ \quad / \backslash \\ \quad 3 \quad 4 \end{array}\right) = * \text{pre}\left(\begin{array}{c} - \\ / \backslash \\ 2 \quad + \\ \quad / \backslash \\ \quad 3 \quad 4 \end{array}\right) 5 = * - 2 \text{pre}\left(\begin{array}{c} + \\ / \backslash \\ 3 \quad 4 \end{array}\right) 5$$

* - 2 + 3 4 5

Parcurgere în *postordine* \Rightarrow expresii *postfix*

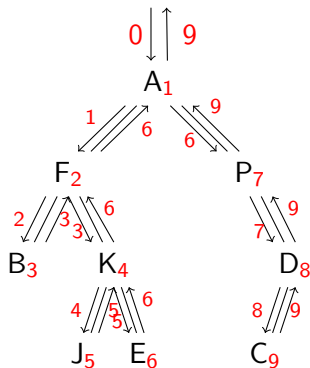
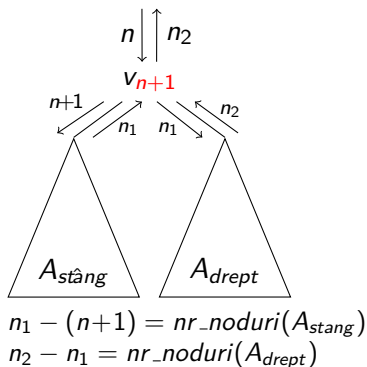
$$\text{post}\left(\begin{array}{c} * \\ / \backslash \\ - \quad 7 \\ / \backslash \\ 2 \quad - \\ \quad / \backslash \\ \quad 4 \quad 5 \end{array}\right) = \text{post}\left(\begin{array}{c} - \\ / \backslash \\ 2 \quad - \\ \quad / \backslash \\ \quad 4 \quad 5 \end{array}\right) 7 * = 2 \text{post}\left(\begin{array}{c} - \\ / \backslash \\ 4 \quad 5 \end{array}\right) - 7 *$$

2 4 5 - - 7 *

Traversări cu numărarea nodurilor (1)

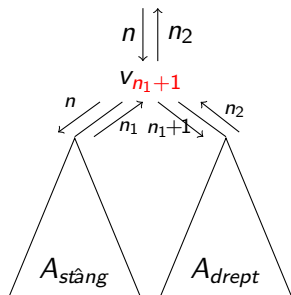
Funcția primește și returnează *ultimul* număr folosit la etichetare
(dacă un arbore primește n , primul nod va fi etichetat cu $n+1$)
Diferența între numărul returnat și primit e nr. de noduri din arbore.

Preordine

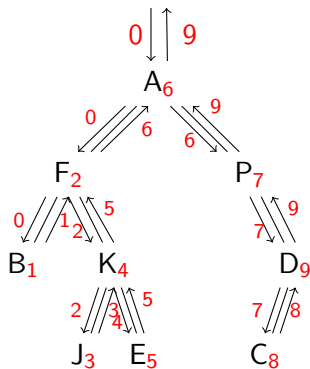


Traversări cu numărarea nodurilor (2)

Inordine



$$n_1 - n = nr_noduri(A_{stang})$$
$$n_2 - (n_1 + 1) = nr_noduri(A_{drept})$$



Traversări cu numărarea nodurilor (3)

Postordine

