

Logică și structuri discrete

Arbori

Marius Minea

marius@cs.upt.ro

<http://www.cs.upt.ro/~marius/curs/lsd/>

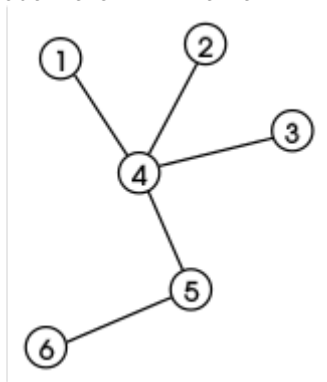
15 decembrie 2014

Arbori

Un arbore e un *graf conex fără cicluri*.

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

⇒ un arbore cu n noduri are $n - 1$ ramuri



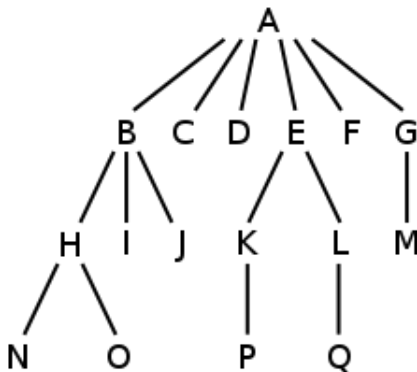
Arbore cu rădăcină

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*



Arborele definit recursiv

Un arbore e fie arborele *vid* sau un *nod* cu mai mulți *subarbori*

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

⇒ considerăm *subarbori* nevizi, doar *tot* arborele poate fi vid

```
type 'a tree = T of 'a * 'a tree list
let t = T('S', [T('A', ['a']), T('b', []),
  T('B', [T(S, ['a']); T('b', [])])])
```

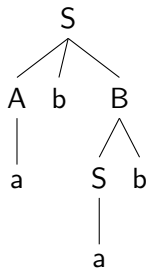
Dacă nu avem arbore vid, tipul e suficient.

Pentru valori care pot lipsi, ML are tipul

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

⇒ lucrăm cu valori de tipul 'a tree option

None sau Some t, cu t de tip 'a tree



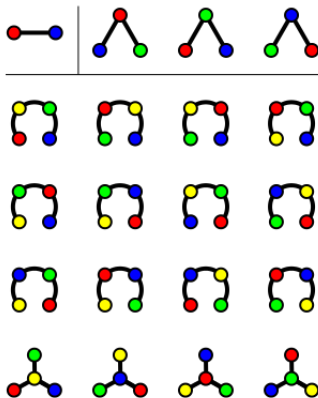
```
let f = function (* parametru arbore *)
  | None -> (* prelucram arborele vid *)
  | Some T(r, tl) -> (* radacina r, lista de copii tl *)
```

Arbori ordonați și neordonați

Ordinea dintre copii unui nod poate să conteze sau nu într-un arbore sintactic, de obicei contează

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

⇒ Un arbore cu n noduri poate fi reprezentat unic ca șir de $n - 2$ numere de la 1 la n : *codul Prüfer*

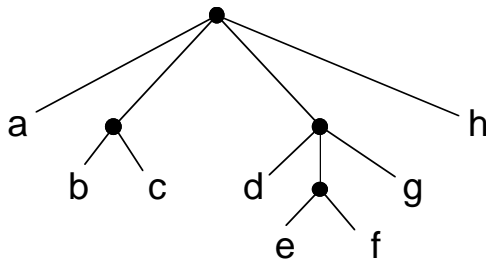


Arbori neetichetați

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)

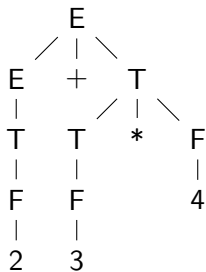
[a, [b, c], [d, [e, f], g], h]

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

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

Arbori în informatică

- Arborii sunt un mod natural de a reprezenta structuri *ierarhice*
- organigrama într-o instituție
- arborele sintactic într-o gramatică (ex. expresie)
- sistemul de fișiere (subarborii sunt cataloagele)
- fișierele XML



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

Arbori binari

Într-un arbore binar, fiecare nod are cel mult doi copii

⇒ un arbore binar e fie

arborele vid

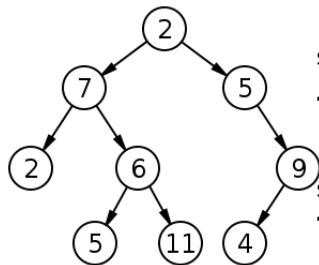
un nod cu cel mult doi subarbori

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

Exemplu de tip, instanțiat pentru noduri întregi:

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

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



subarborile stâng:

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

subarborile drept:

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


Arbori binari stricți

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

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

Parcurgerea arborilor

în *preordine*: întâi rădăcina, apoi subarborii

în *inordine*: arborele stâng, apoi rădăcina, apoi arborele drept

în *postordine*: întâi subarborii, apoi rădăcina

Pentru expresii, obținem astfel formele prefix, infix și postfix

Parcurgerea în pre- și post-ordine se definește la fel pentru orice arbori (nu doar binari)