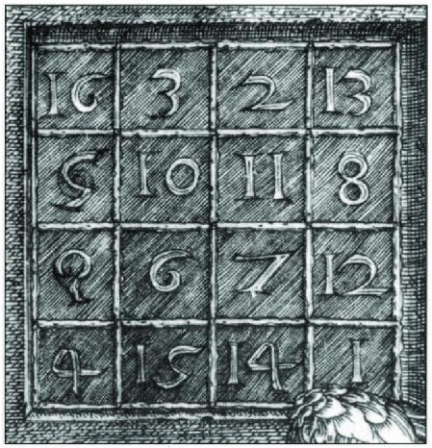
Logică și Structuri Discrete -LSD

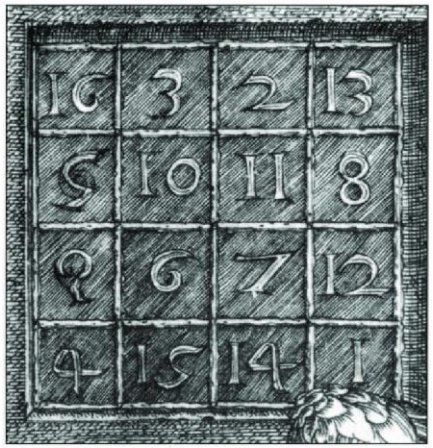


Cursul 8 – Arbori. Tupluri

dr. ing. Cătălin Iapă

catalin.iapa@cs.upt.ro

Ce am parcurs până acum?



Funcții

Funcții recursive

Liste

Mulțimi

Relații

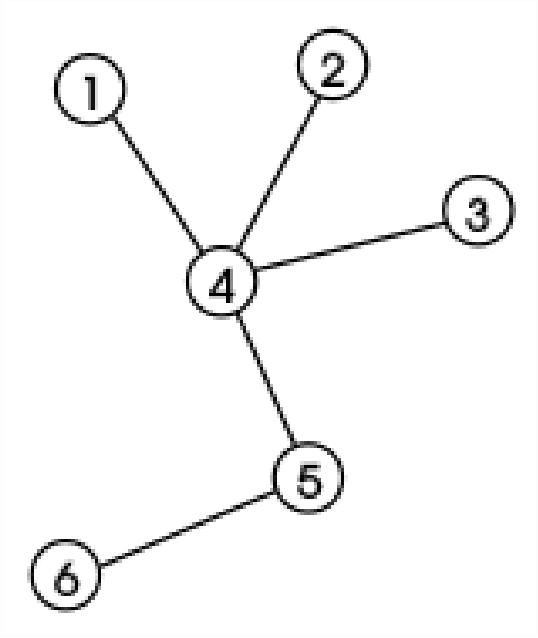
Dicționare

Grafuri

Arbori

Un arbore e un *graf neorientat conex și fără cicluri*.

Arborii reprezintă grafurile cele mai simple ca structură din clasa grafurilor conexe, ei fiind și cei mai frecvent utilizați în practică.



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

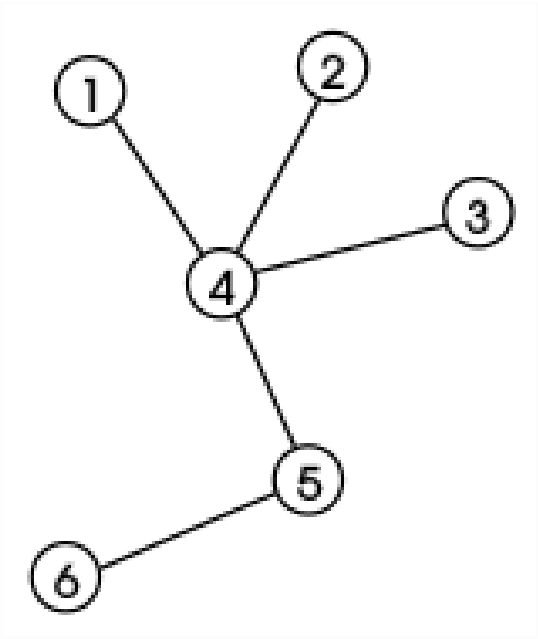
Arbori

Un arbore e un *graf neorientat conex și 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



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

Exemple de arbori

(i) - arbore

(ii) - arbore

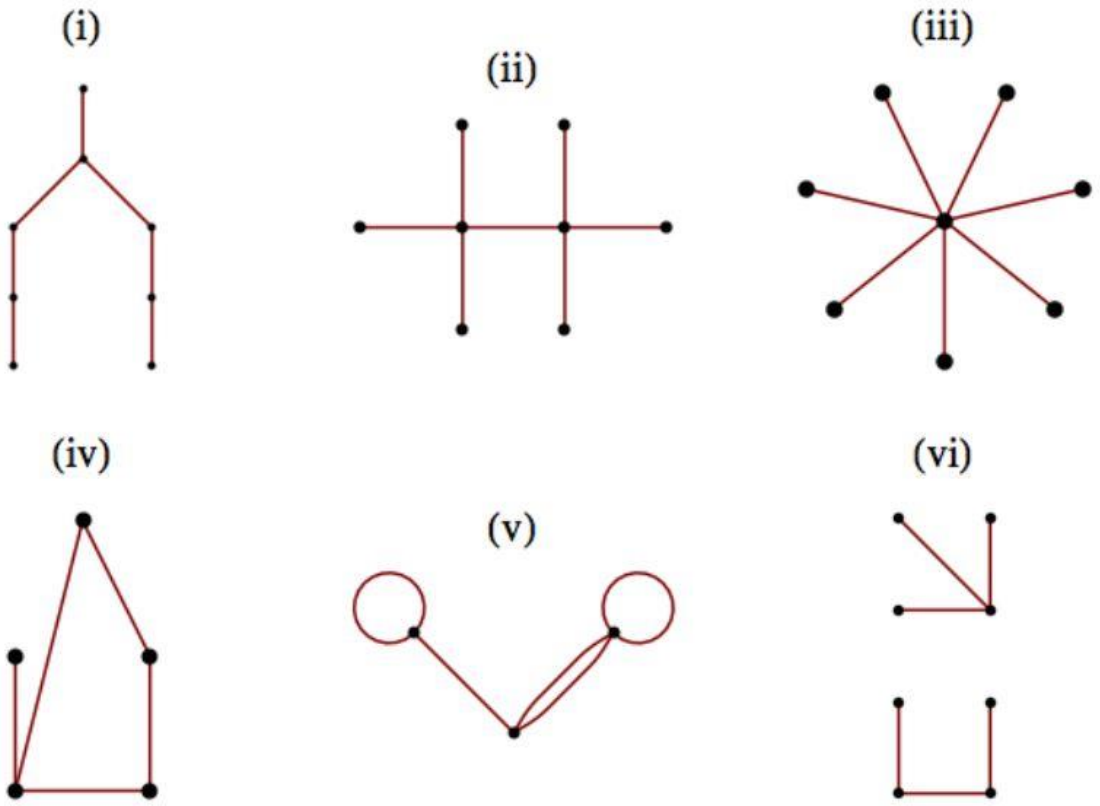
(iii) - arbore

(iv) - nu e arbore

(v) - nu e arbore

(vi) - nu e arbore

(e pădure)



Pădure

Un tip de graf strâns legat de conceptul de arbore, dar care nu îndeplinește toate condițiile unui arbore e *pădurea*.

O *pădure* este un graf neorientat neconex a cărui componente conexe sunt *arbori*.

Condiții pentru ca un graf să fie arbore

Dacă avem graful G = (V, E) neorientat și fără cicluri, iar |V| = n, propozițiile următoare sunt echivalente:

- G e un *arbore*
- Pentru fiecare 2 noduri distincte din V, există *un singur drum* între ele
- G este conex, și dacă avem o muchie e, atunci graful (V, E – {e}) *nu e conex*
- G nu conține cicluri, dar *dacă adăugăm o muchie* în plus vom avea un ciclu
- G este conex, iar *|E| = n-1*

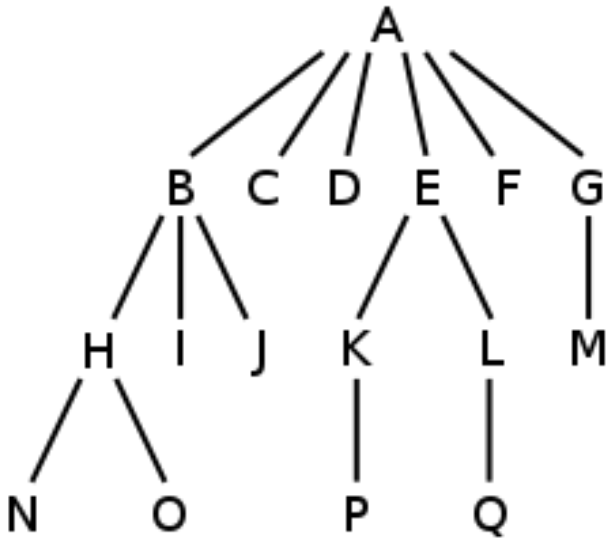
Arbore cu rădăcină

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*



Imagine: http://en.wikipedia.org/wiki/File:N-ary_to_binary.svg

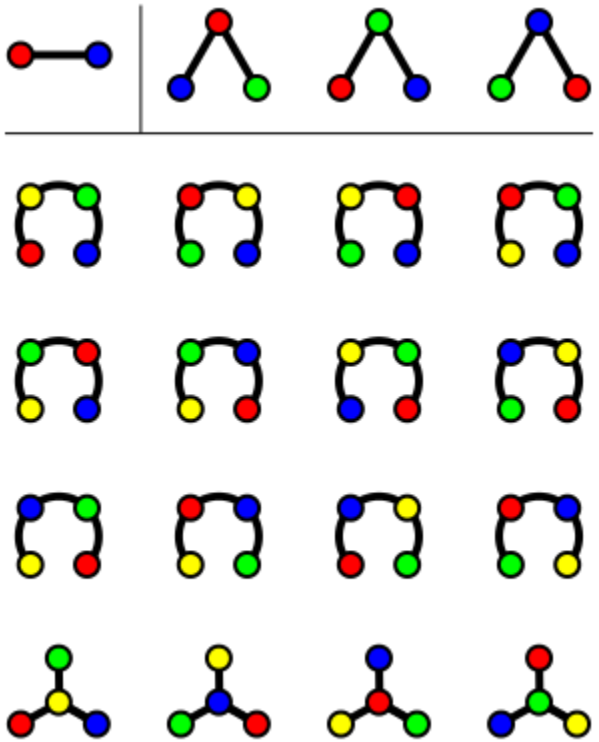
Arbori în informatică

- 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)

Arbori ordonați și neordonați

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

Arborii neordonați cu
2 – 4 noduri – în figură:



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

Imagine: http://en.wikipedia.org/wiki/File:Cayley's_formula_2-4.svg

Arbori – structuri recursive

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

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

În funcție de problemă, nodurile conțin *informație*

Arbori binari

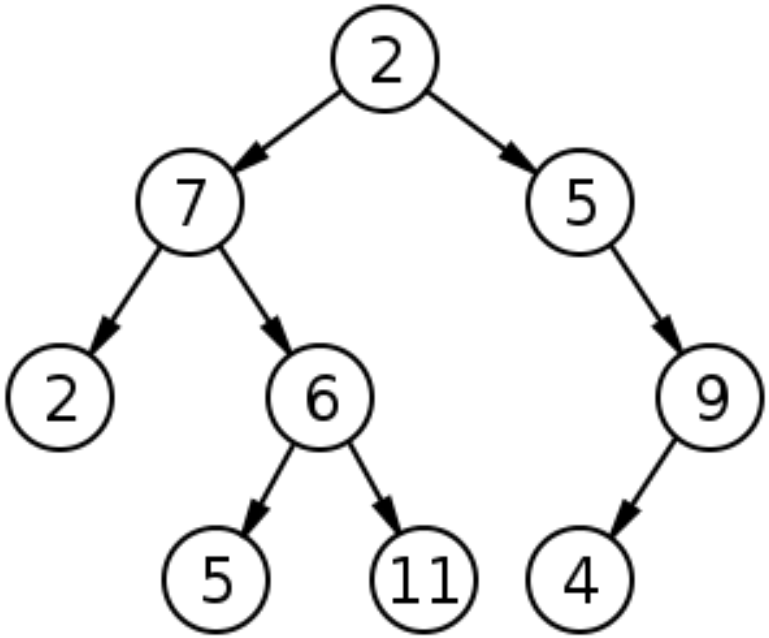
Într-un arbore binar, fiecare nod are cel mult doi copii, identificați ca fiul stâng și fiul drept (oricare/ambii pot lipsi)

⇒ un arbore binar e:

- arborele vid sau
- un nod cu cel mult doi subarbori

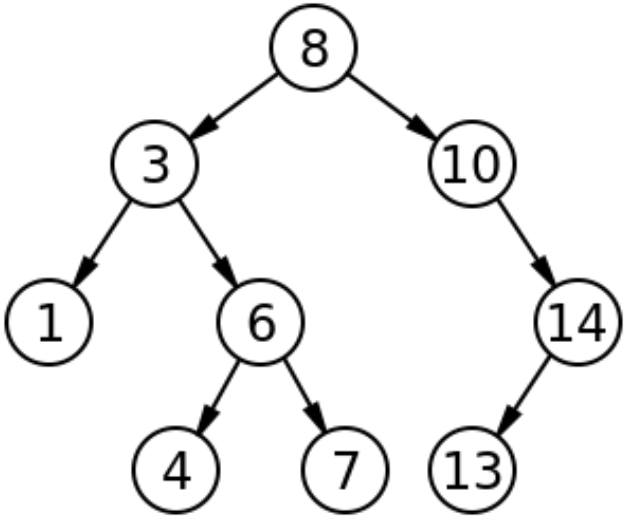
Arbori binari

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



Arbori binari de căutare

Arborii binari de căutare sunt arbori binari care 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 *am găsit* elementul în arbore
- dacă e < rădăcina curentă, se continuă căutarea în subarborele stâng
- dacă e > rădăcina curentă, se continuă căutarea în subarborele drept

Sortarea cu ajutorul arborilor de căutare

Arborii de căutare pot fi folosiți la *sortarea unui șir de obiecte* care pot fi ordonate.

Mai întâi se *creează arborele* de căutare cu obiectele din șir:

- primul obiectva fi rădăcina arborelui
- următoarele obiecte se introduc în subarborele stâng sau drept, în funcție de valoare

Iar apoi *se parcurge arborele de căutare în inordine* (arbore stâng, rădăcină, arbore drept) și vom obține obiectele din șir ordonate.

Parcurgerea arborilor

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

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

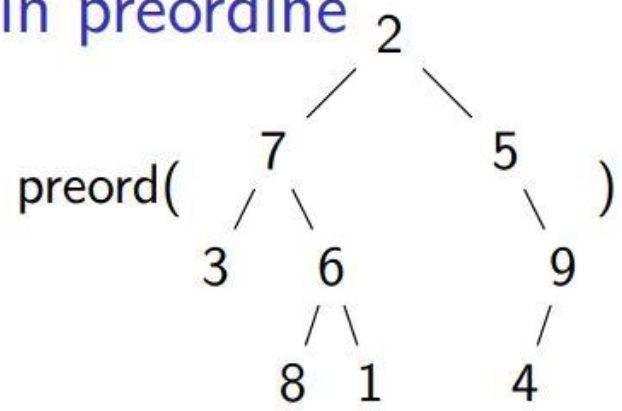
în *postordine*: subarborele *stâng*, subarborele *drept*, *rădăcina*

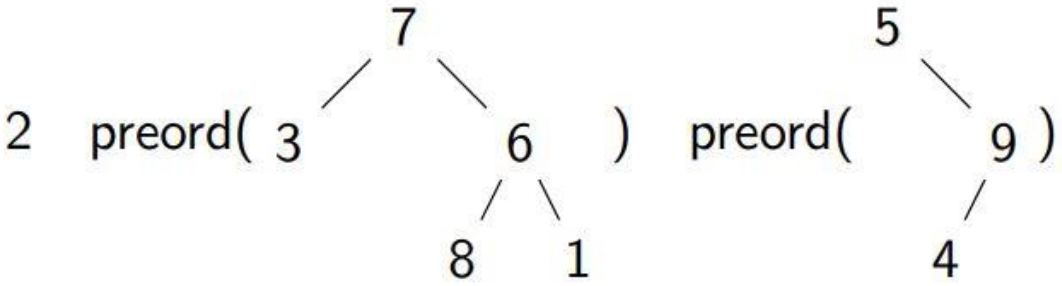
subarborii se parcurg și ei tot în ordinea dată (pre/in/post ordine)!

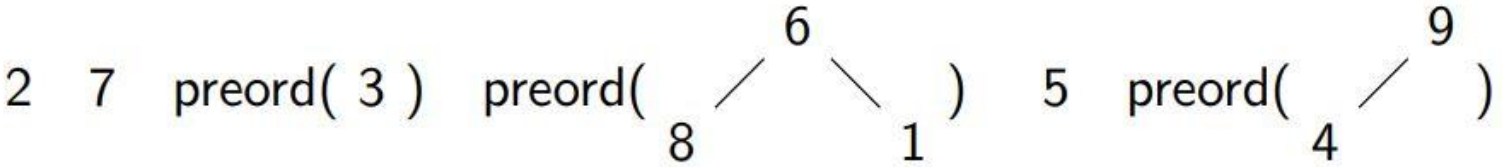
Pentru expresii, obținem astfel formele prefix, infix și postfix

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

Exemplu: parcurgere în preordine

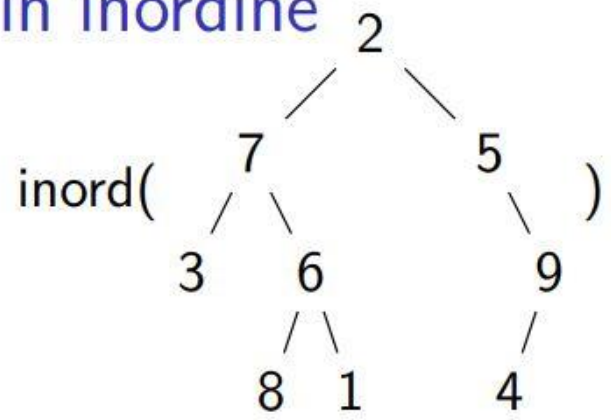


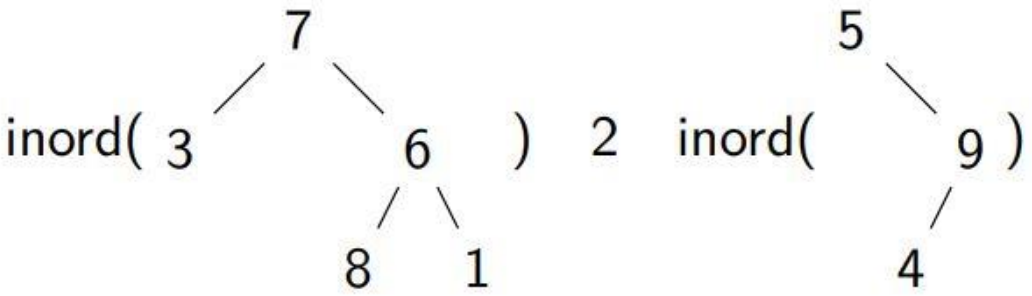


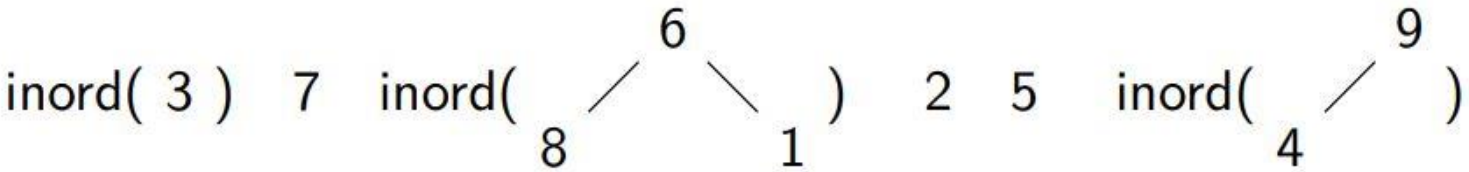


2 7 3 6 8 1 5 9 4

Exemplu: parcurgere în inordine

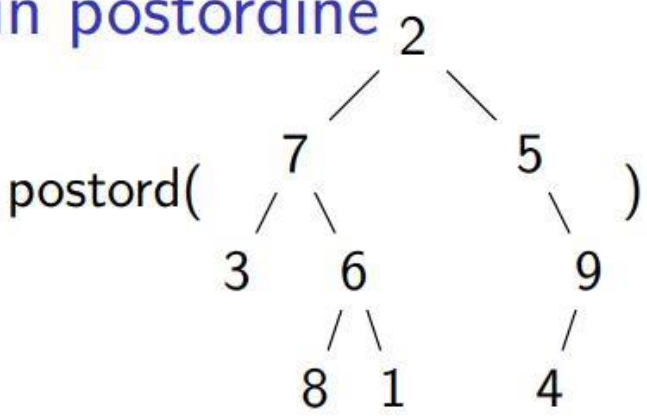


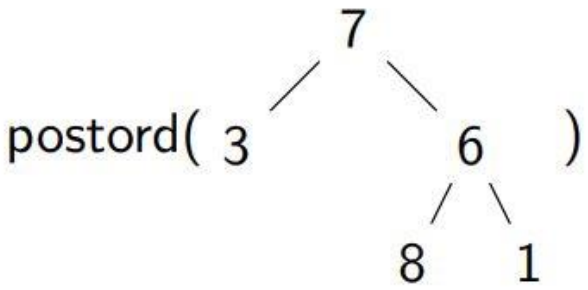


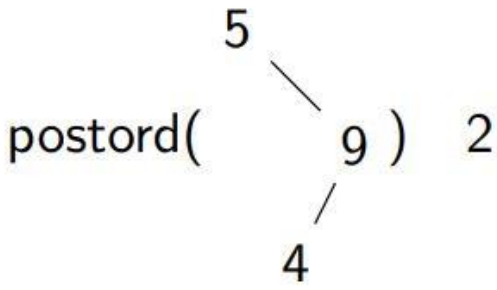


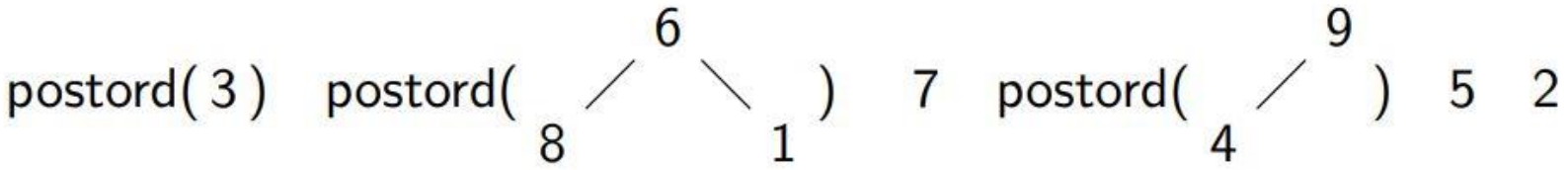
3 7 8 6 1 2 5 4 9

Exemplu: parcurgere în postordine









3 8 1 6 7 4 9 5 2

Reprezentarea unui arbore oarecare

Pentru a reprezenta un arbore, pentru fiecare nod vom avea un *dicționar* care va conține două perechi: valoarea nodului și lista valorilor copiilor săi. Arborele va fi reprezentat de o listă care conține toate nodurile sale sub forma:

Nod:

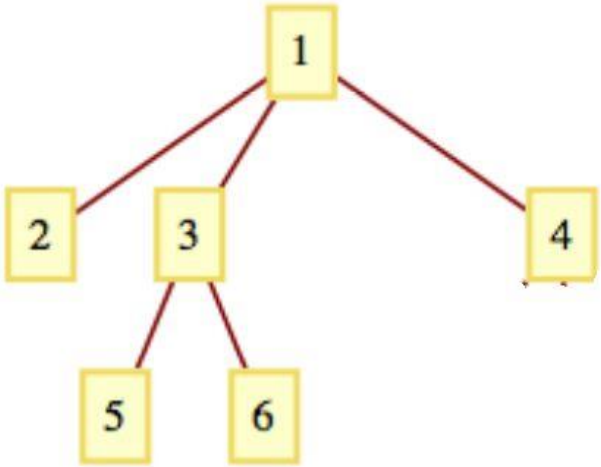
{"valore" : None, "copii" : []}

Arbore:

[{"valore" : None, "copii" : […]}, …]

Reprezentarea unui arbore - exemplu

```
arbore_oarecare = [
    {"valoare" : 1, "copii" : [2, 3, 4]},
    {"valoare" : 2, " copii" : []},
    {"valoare" : 3, " copii" : [5, 6]},
    {"valoare" : 4, " copii" : []},
    {"valoare" : 5, " copii" : []},
]
```



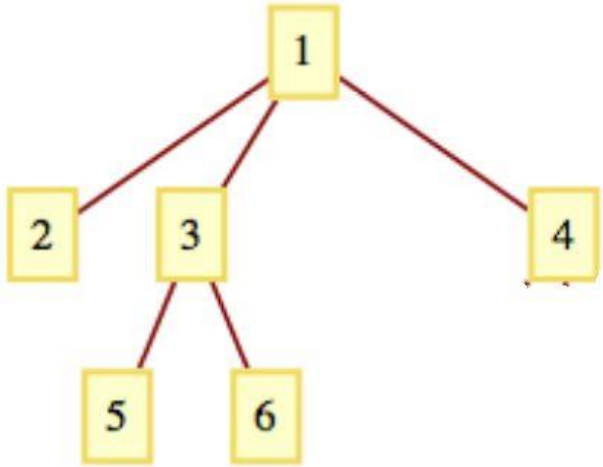
Reprezentarea unui arbore

Un alt mod de a reprezenta arborele este ca în lista copiilor unui nod să fie reținute direct informațiile sub forma de listă de dicționare, nu doar sub formă listă de valori.

În acest fel ne folosim de structura recursivă a unui arbore.

Reprezentarea unui arbore - exemplu

```
arbore_oarecare2 = { "valoare": 1, " copii":
        [
           { "valoare": 2, " copii": []},
           { "valoare": 3, " copii":
              [
                 { "valoare" :5, " copii": []},
                 { "valoare" :6, " copii": []}
              ]
           },
           { "valoare": 4, "copii": []}
        ]
     }
```

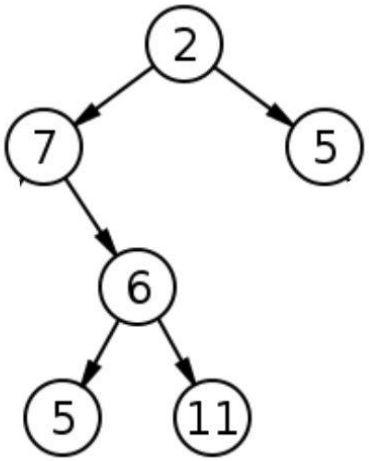


Reprezentarea unui arbore binar

Un *arbore binar* îl putem reprezenta recursive ca un dicționar cu 3 perechi: *valoare*, *arbore stâng* și *arbore drept*.

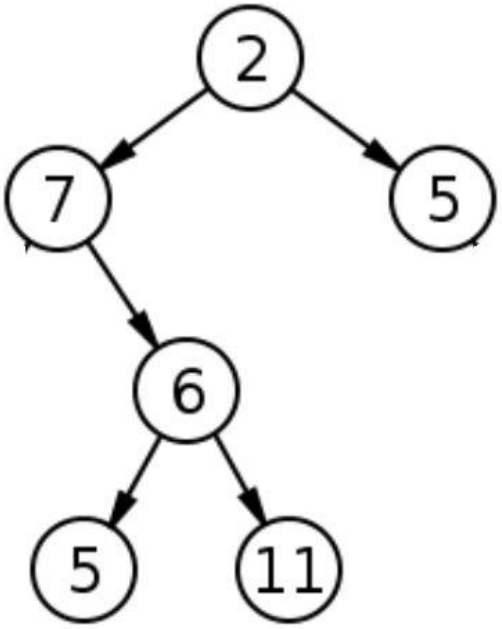
arbore = {"value": None, "left": None, "right": None}

```
tree2 = { "value" : 2, "left":
        {
            "value": 7, "left": None, "right":
            {
                "value": 6, "left":
                {
                    "value": 5, "left": None, "right": None
                }, "right":
                {
                    "value":11, "left": None, "right": None
                },
            },
        }, "right":
            {
                "value": 5, "left": None, "right": None
            }
    }
```



Parcurgerea în preordine

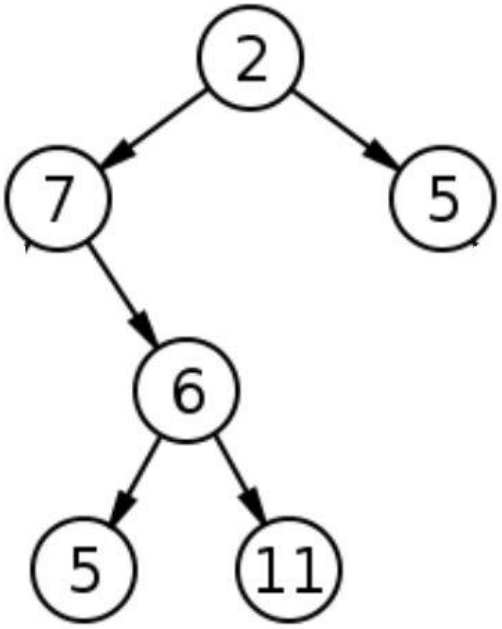
```
def rsd(tree):
    if (tree != None):
        return [tree["value"]] + rsd(tree["left"]) + rsd(tree["right"])
    else:
        return []
```




```
print(rsd(binary_tree))
# [2, 7, 6, 5, 11, 5]
```

Parcurgerea în inordine

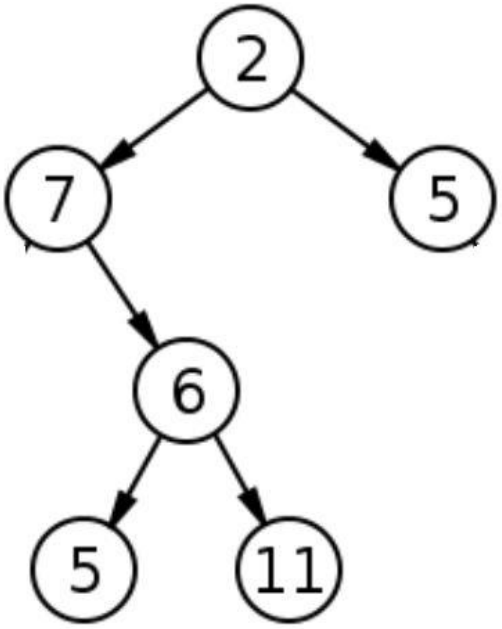
```
def srd(tree):
    if (tree != None):
        return  srd(tree["left"]) + [tree["value"]] + srd(tree["right"])
    else:
        return []
```




```
print(srd(binary_tree))
# [7, 5, 6, 11, 2, 5]
```

Parcurgerea în postordine

```
def sdr(tree):
    if (tree != None):
        return  sdr(tree["left"]) + sdr(tree["right"]) + [tree["value"]]
    else:
        return []
```

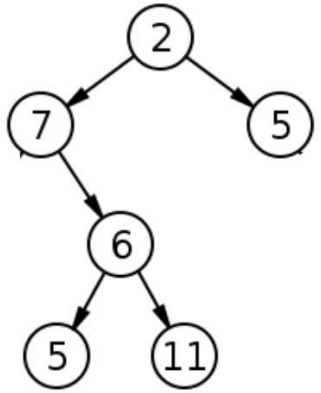



```
print(sdr(binary_tree))
#[5, 11, 6, 7, 5, 2]
```

Adaugarea unui nod nou

Adăugarea unui nod nou la un părinte și o poziție anume:

```
def adaugare_nod_pozitie(parinte, nod_nou, pozitie):
    if (parinte[pozitie] == None):
        parinte[pozitie] = nod_nou
    return parinte
```



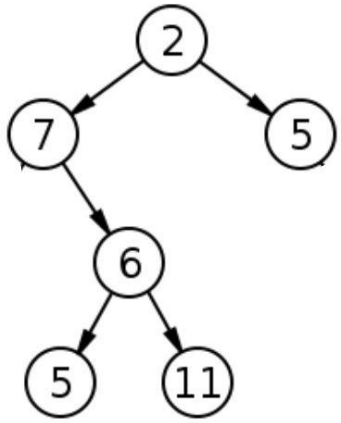

```
binary_tree["left"]=adaugare_nod_pozitie(binary_tree["left"],
{"value": 100, "left": None, "right": None}, "left")
print(rsd(binary_tree))
#[2, 7, 100, 6, 5, 11, 5]
```

Adaugarea unui nod nou

Adăugarea unui nod nou *în arbore binar de căutare*:

```
def adaugare_nod(tree, nod_nou):
    if (tree == None):
        return nod_nou
    if (nod_nou["value"]<tree["value"]):
        tree["left"] = adaugare_nod(tree["left"], nod_nou)
    else:
        tree["right"] = adaugare_nod(tree["right"], nod_nou)
    return tree


print(rsd(adaugare_nod(binary_tree,{"value": 1, "left": None,
"right": None})))
#[2, 7, 1, 6, 5, 11, 5]
```

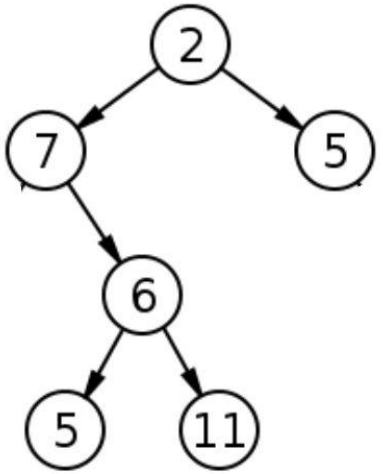


Ștergerea unui nod/ subarbore

Ștergerea unui nod (sau subarbore) de la un anumit părinte dat ca parametru:

```
def stergere_nod(parinte, valoare_nod):
    if (parinte["left"]["value"] == valoare_nod):
        parinte["left"] = None
    elif(parinte["right"]["value"] == valoare_nod):
        parinte["right"] = None

stergere_nod(binary_tree, 5)
print(rsd(binary_tree))
#[2, 7, 6, 5, 11]
```



Tupluri în PYTHON

Un *tuplu* este o coleție de date predefinite în PYTHON (pe lângă liste, mulțimi și dicționare).

Un tuplu este o colecție de date *ordonată* și *nu se mai poate schimba după creare*.

Un tuplu se scrie între paranteze rotunde:

tuplu = (2, 5, 7, 1, 5)

Tupluri în PYTHON

Elementele unui tuplu:

- sunt *ordonate* (se pot accesa prin index pozitiv sau negativ)
- *nu se mai pot schimba* după creare
- permit *duplicate*

Tupluri în PYTHON

Numărul de elemente din tuplu se poate afla cu funcția *len()*

```
a = (1, 6, 8)
print(len(a)) # 3
```

Pentru a crea un tuplu cu un singur element e nevoie să se pună *între paranteze rotunde și o virgulă la final*:

```
tuplu = (5,)
```

Tupluri în PYTHON

Se poate crea un tuplu și cu constructorul *tuple()*

```
a = tuple((4, 6, 8))
b = tuple([“Arad”, “Timisoara”])
```

Elementele se accesează prin indecși:

```
print(a[0])          # 4
print(b[1])          # Timisoara
print(b[-2])         # Arad
print(a[1:3])        # (6, 8)
print(9 in a)        # False
print(8 in a)        # True
```

Tupluri în PYTHON

Nu se pot adăuga elemente în tuplu și nici *nu se pot șterge* ulterior creerii acestuia.

Aceste operații sunt permise la lucrul cu liste. Dacă vrem să creem un nou tuplu cu elemente diferite se poate transforma în listă și prelucra:

```
a = (3, 5, 7, 3)
lista = list(a)
#prelucrarea listei lista
```

Tupluri în PYTHON

Putem *extrage elemente* din tuplu și apoi să le prelucrăm independent:

```
tuplu = (3, 3, 6, 8)
a, b, c, d = tuplu
print(a)                    # 3
print(b)                    #3
print(c)                    #6
```

Dacă numărul elementelor din tuplu e mai mare e obligatoriu să folosim un asterisc la ultimul obiect:

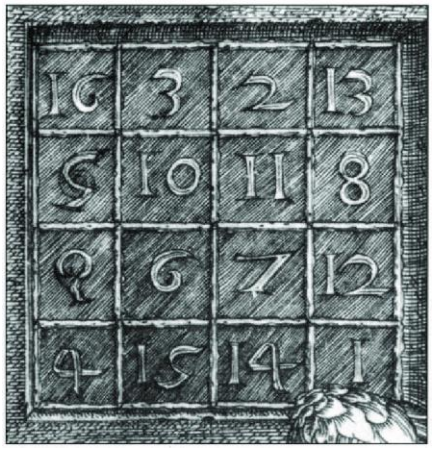
```
a, b, *c = tuplu            # c = (6, 8)
```

Tupluri în PYTHON

Se poate crea un tuplu nou cu elementele din alte 2 sau mai multe tupluri:

```
a = (1, 2, 3)
b = (“a”, “b”, “c”)
c = a + b

print(c)                    # (1, 2, 3, “a”, “b”, “c”)
```



Vă mulțumesc!

Bibliografie

- Conținutul cursului se bazează preponderent pe materialele de anii trecuți de la cursul de LSD, predat de conf. dr. ing.Marius Minea și ș.l. dr. ing. Casandra Holotescu (http://staff.cs.upt.ro/~marius/curs/lsd/index.html)