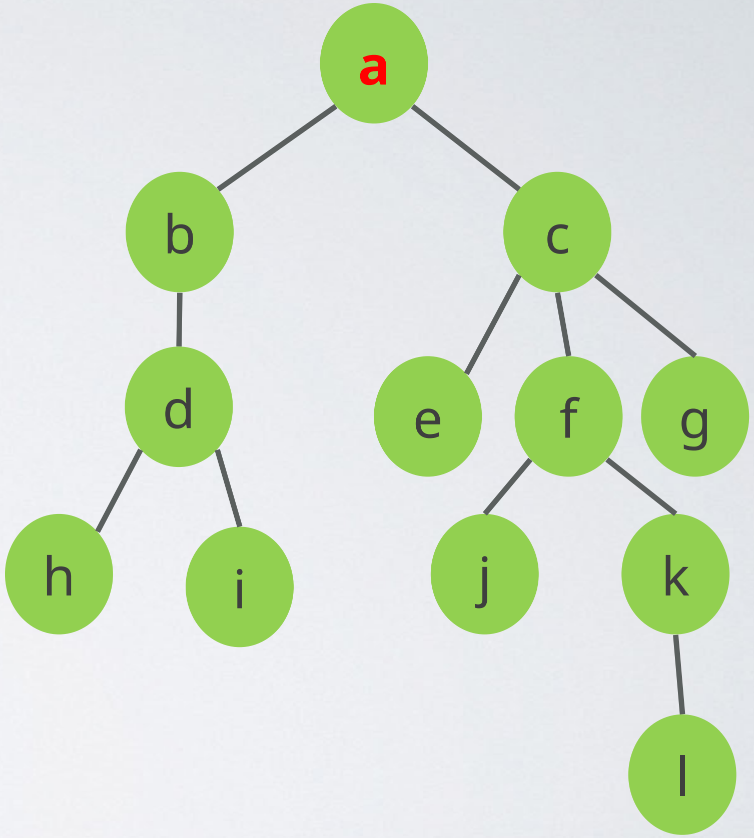
SDA CURS 3: ARBORI

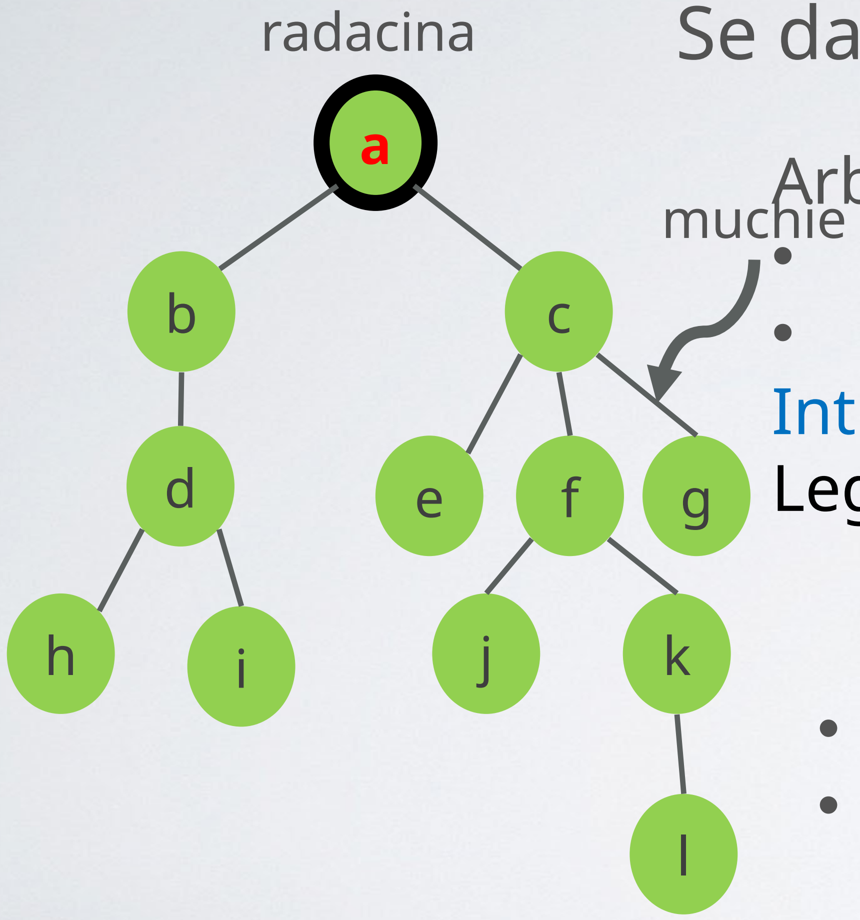
Definitie. Arbori oarecare. Parcurgeri. Arbori cu etichete si arbori pentru expresii. Arbore ADT. Implementari ale arborilor. Arbori binari de cautare

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

ARBORI

- <u>Arbori oarecare:</u> colectie de elemente numite noduri - avand un nod radacina si o relatie de paternitate intre noduri – fapt ce impune o ***structura ierarhica*** a nodurilor.
- <u>Definitie</u>: *structura recursiva*, colectie ierarhica de noduri, fiecare nod:
  - fie este gol (nil, NULL)
  - fie este o structura care contine o cheie si o colectie de referinte catre noduri *copil*, radacinile $n_1, n_2, ..., n_k$ ale sub-arborilor $T_1, T_2, ..., T_k$
- Structura de date pentru colectii non-liniare





Se da un arbore oarecare T = (V, E) cu radacina r in V:

Arborele din figura are:

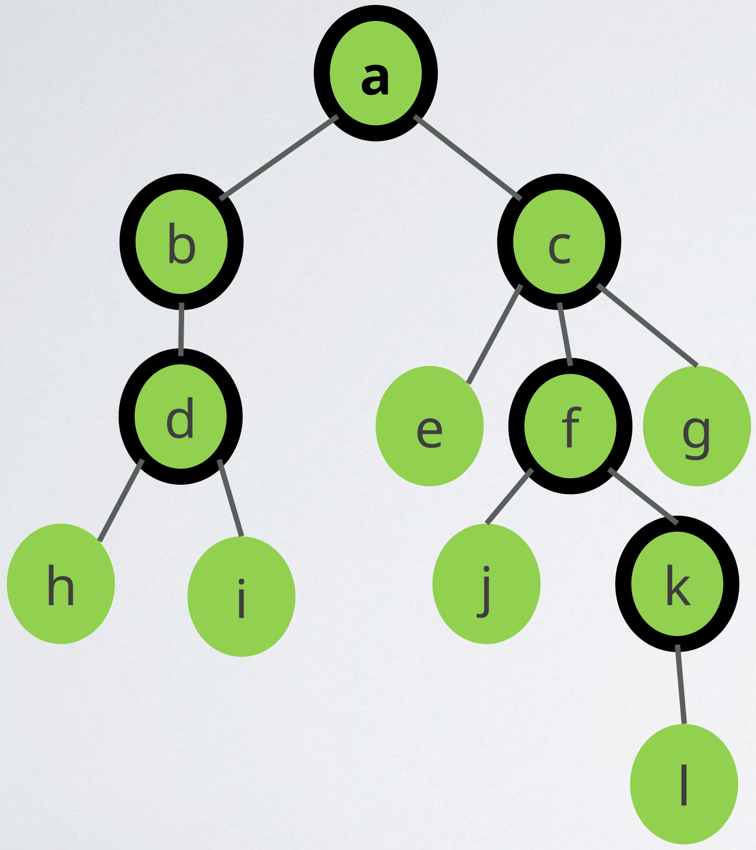
- 12 noduri: |V| = 12
- 11 muchii: |E| = 11

Intr-un arbore cu n noduri vom avea n-1 muchii.

Legatura dintre 2 noduri se numeste muchie.

- Primul nod este radacina.
- Un arbore are o singura radacina,

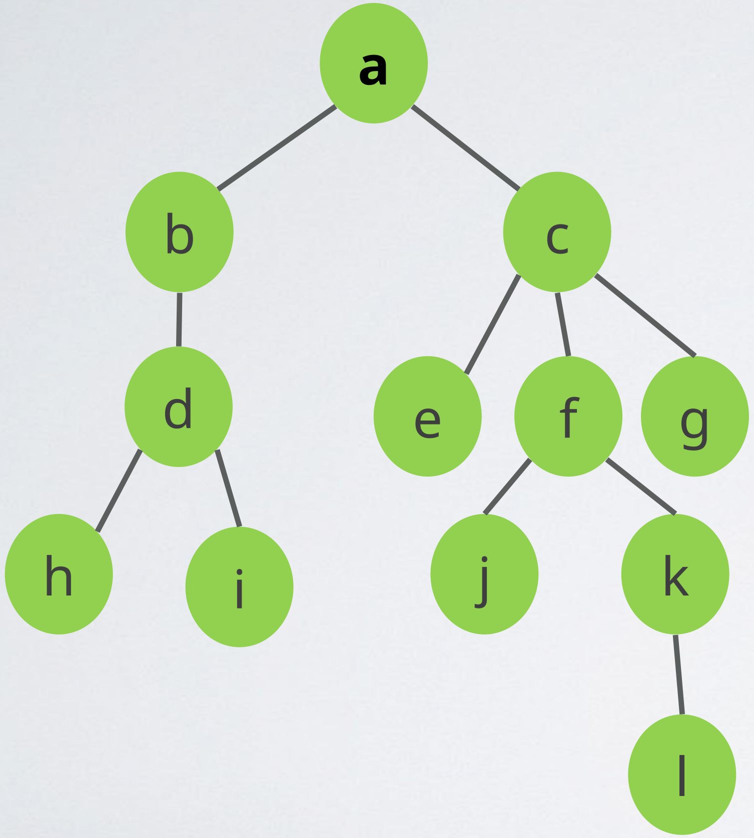
TERMINOLOGIA PENTRU ARBORI – PARINTE, COPII



Intr-un arbore nodul care precede un alt nod se numeste parinte.
Intr-un arbore nodul care descende din alt nod se numeste copil.
Intr-un arbore oarecare, un nod parinte poate avea oricate noduri copil.
Intr-un arbore, toate nodurile sunt copii, cu exceptia radacinii.

- Noduri parinte in exemplul din figura:
  - a, b,d,c,f,k
- Copiii lui a sunt b si c
- Copiii lui d sunt h si i

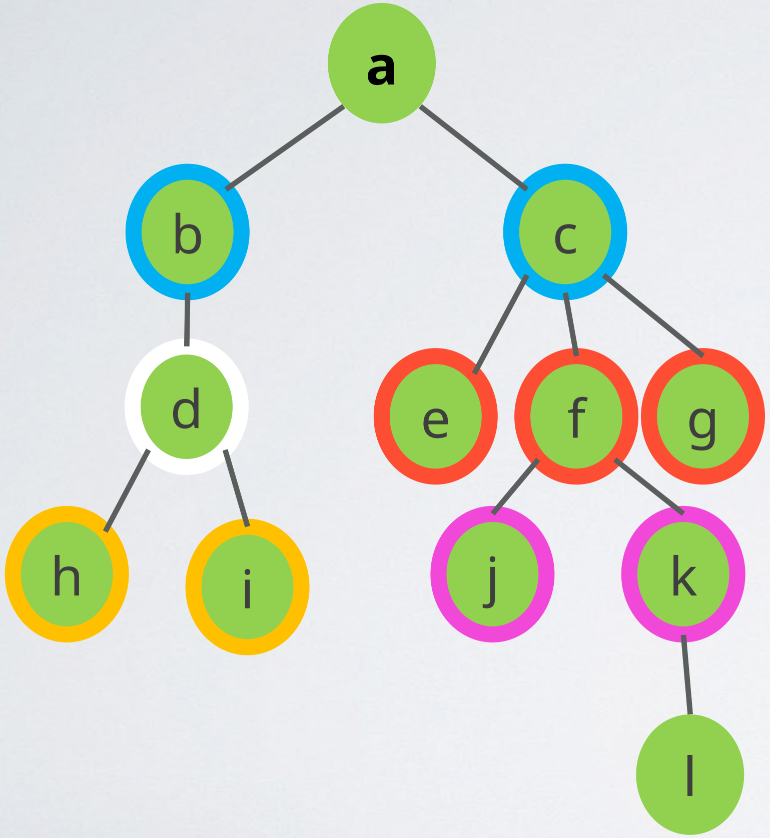
TERMINOLOGIA PENTRU ARBORI – DESCENDENTI, STRAMOSI



Descendent al lui v: Orice nod in care se ajunge parcurgand muchiile de la v in jos (pe relatie copil)
Stramos al lui v: orice nod in care se ajunge parcurgand muchiile de la v in sus (pe relatie parinte)

- Descendentii lui c sunt e, f, g, j, k, l
- Stramosii lui k sunt f, c, a

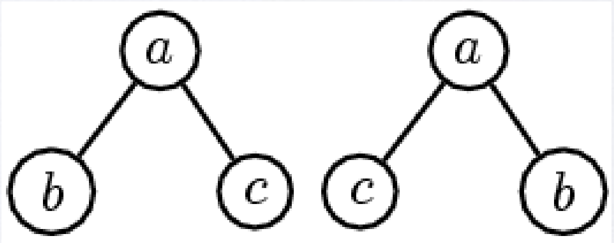
TERMINOLOGIA PENTRU ARBORI – FRATI



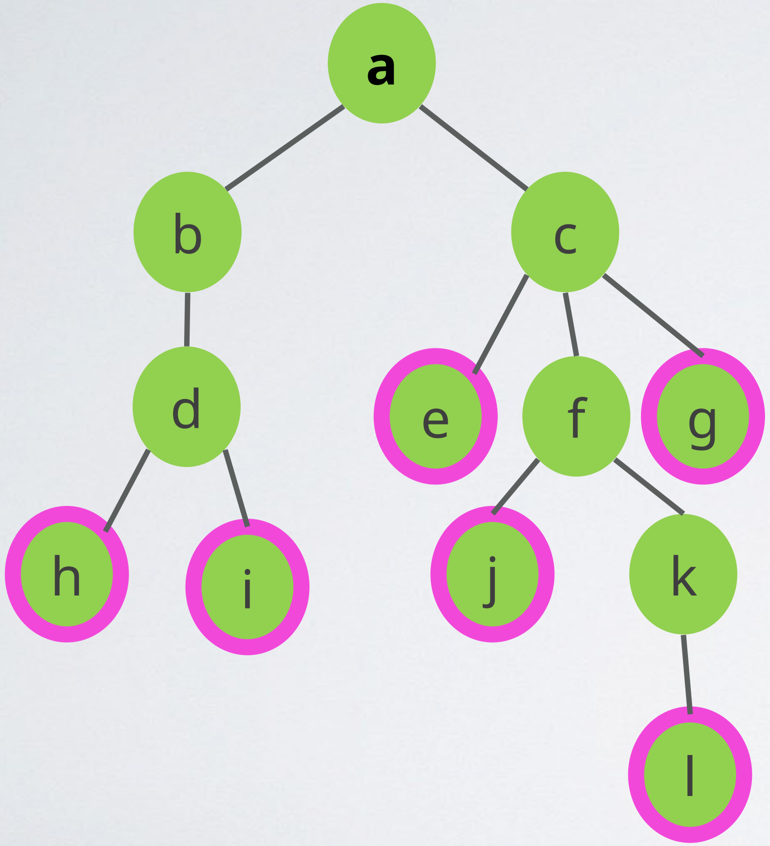
Nodurile care au acelasi parinte se numesc frati.
Fratii pentru exemplul din figura sunt:

- b, c.
- h,I
- e,f,g
- j,k

- Ordinea fratilor poate sau nu sa conteze



TERMINOLOGIA PENTRU ARBORI – FRUNZE

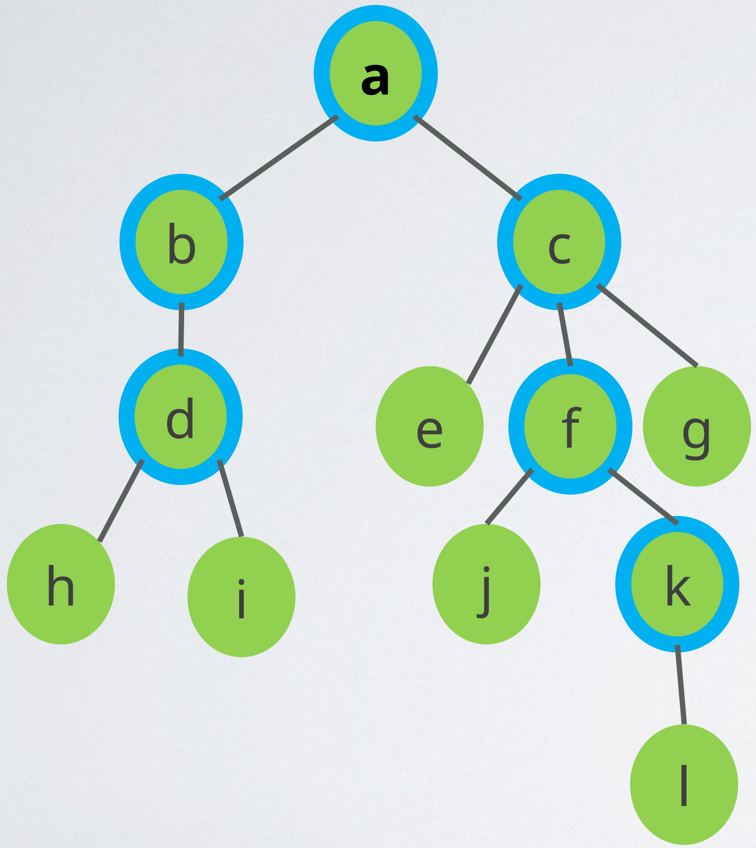


Intr-un arbore, nodurile care nu au copii se numesc frunze, sau noduri terminale, sau noduri externe.

Frunzele din exemplu sunt:

- h,i,e,g,j,l

TERMINOLOGIA PENTRU ARBORI – NODURI INTERNE

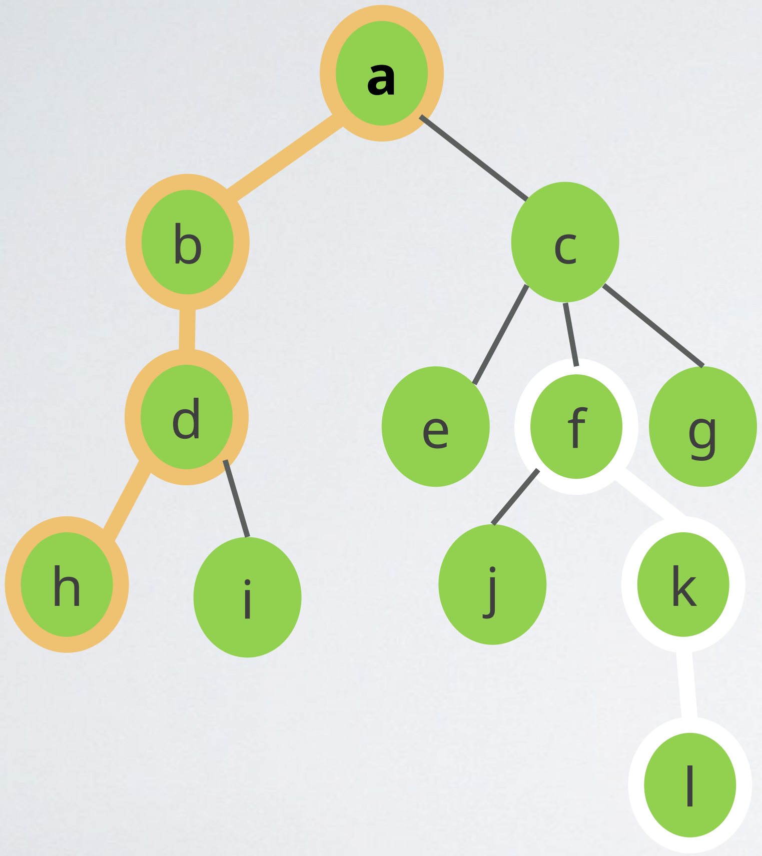


Nod intern – nod care are cel putin un copil.
Se mai numeste nod non-terminal.
Radacina e considerata nod intern.

Nodurile interne din exemplu sunt:

- a,b,d,c,f,k

TERMINOLOGIA PENTRU ARBORI – DRUM, CALE



Cale(en. *path*) dintre doua noduri este secventa de noduri si muchii cuprinsa intre cele doua noduri.

$\langle n_1, n_2, \ldots, n_k \rangle$ astfel incat n_i = parintele lui n_{i+1} pentru $1 \le i \le k$.

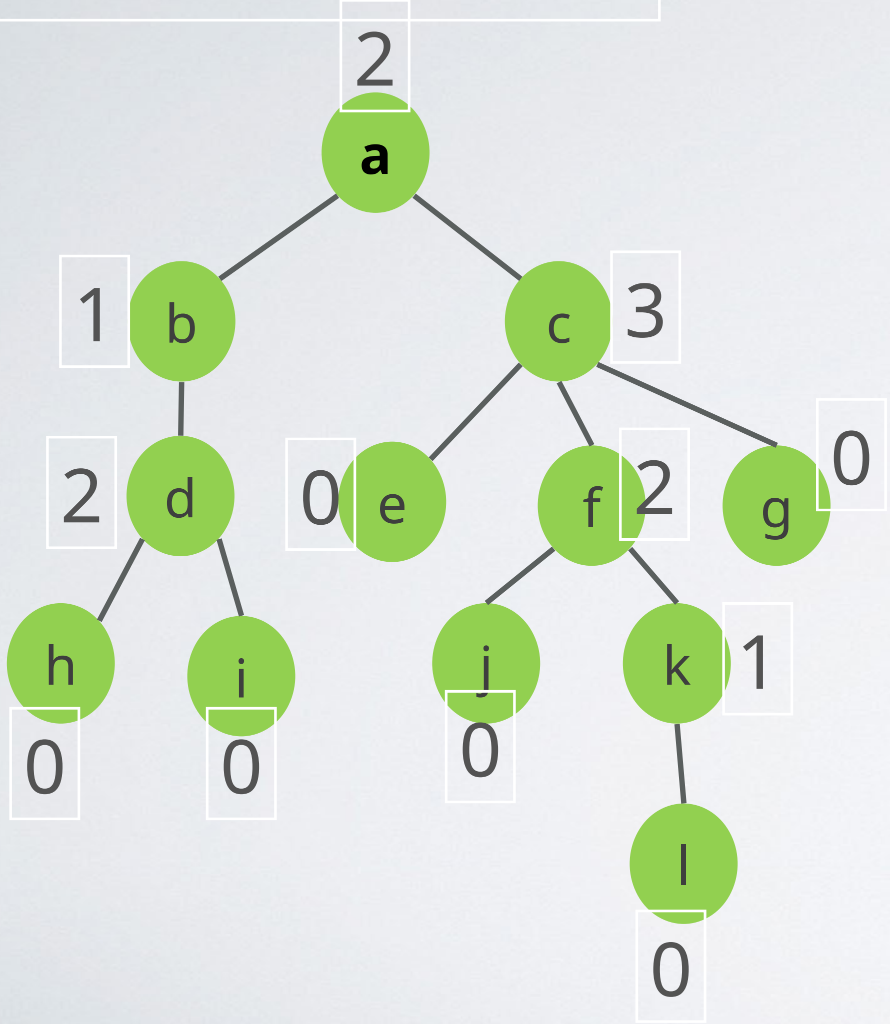
$lungime$(cale) = nb_noduri-1 = nb_muchii

Exemplu :

- Drumul dintre a si h este:
  - a-b-d-h
  - Lungimea drumului este 3
- Drumul dintre f si l:
  - f,k,l
  - Lungime = 2

TERMINOLOGIA PENTRU ARBORI – GRADUL UNUI NOD

Gradele fiecarui nod:

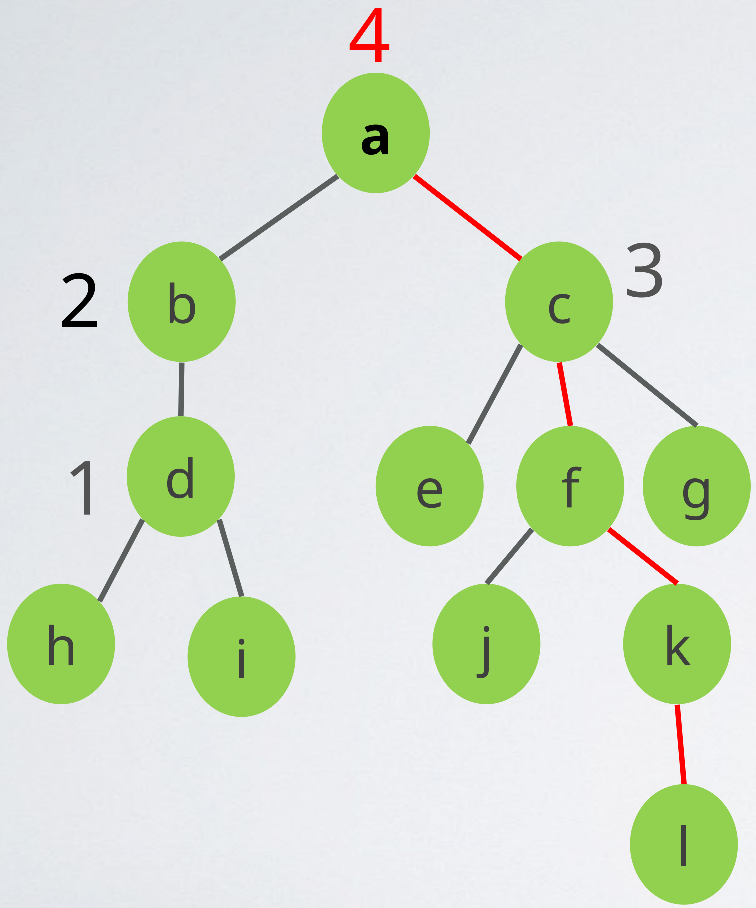


Gradul unui nod este egal cu numarul de copii ai nodului.
Gradul arborelui este maximul dintre gradele nodurilor din arbore.

In exemplu:

- Grad(a) = 2
- Grad(b)=1
- Grad(h) = 0
- Grad(c) = 3
- ….

TERMINOLOGIA PENTRU ARBORI – INALTIME



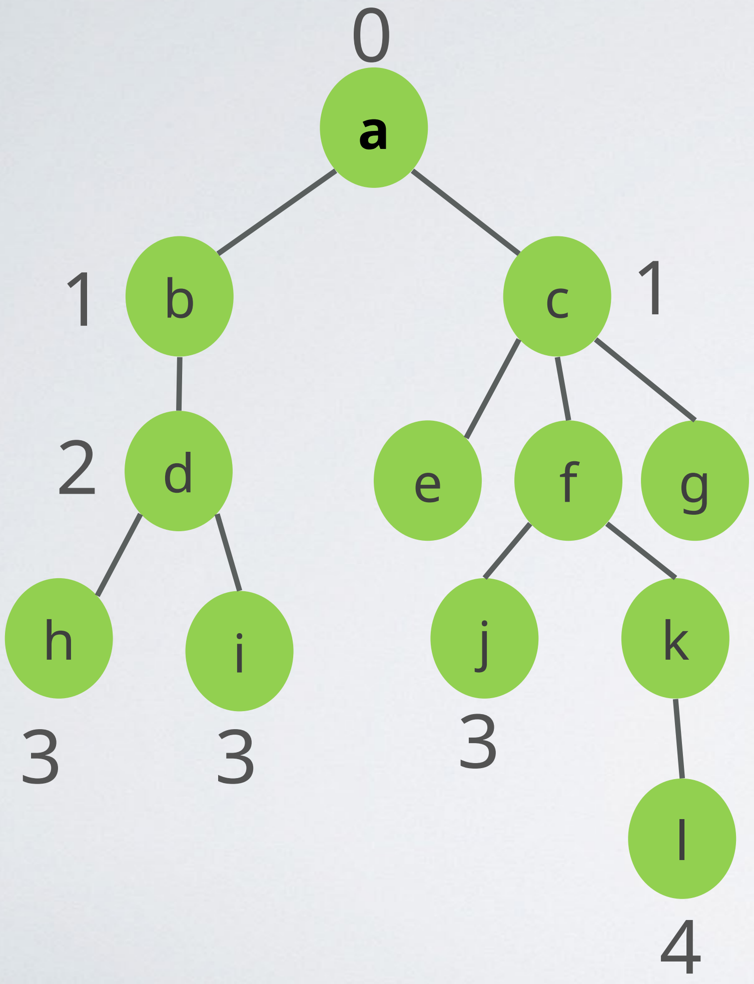
Inaltimea (height) a unui nod v

`inaltime(v) = lungimea celui mai lung drum de la v la o frunza.`

Inaltimea arborelui T este inaltimea radacinii r (Inaltime(T)=Inaltime(r)).
Inaltimea frunzelor este 0.

Inaltimea(a) = 4
Inaltime(b)=2;
Inaltime(k)=1;
Inaltime(h)=0;
….

TERMINOLOGIA PENTRU ARBORI – ADANCIME



Adancimea unui nod (varf) $v \in V$:

`adancime(v) = lungimea drumului de la r la v`

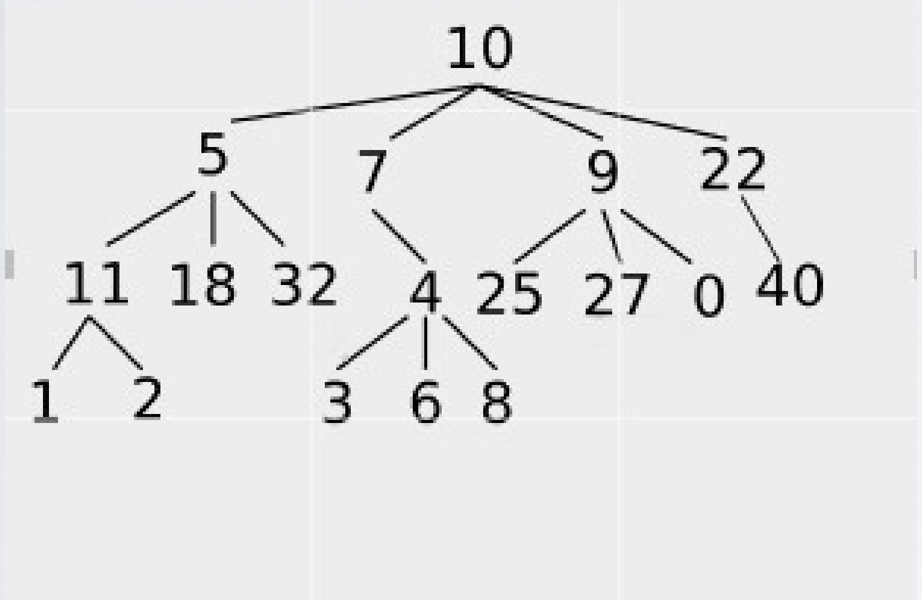
- numarul de muchii continute de drumul de la radacina pana la acel nod.
- Adancimea radacinii este 0.
- Adancimea arborelui este maximul dintre adancimile frunzelor.

Exemplu:

- Adancime(a)=0
- Adancime(d)=2
- Adancime(k)=3
- Adancime(l)=4
- ….

TERMINOLOGIA PENTRU ARBORI – EXERCITIU

inaltime(v) = lungimea celui mai lung drum de la v la o frunza.
adancime(*v*) = lungimea drumului de la r la v

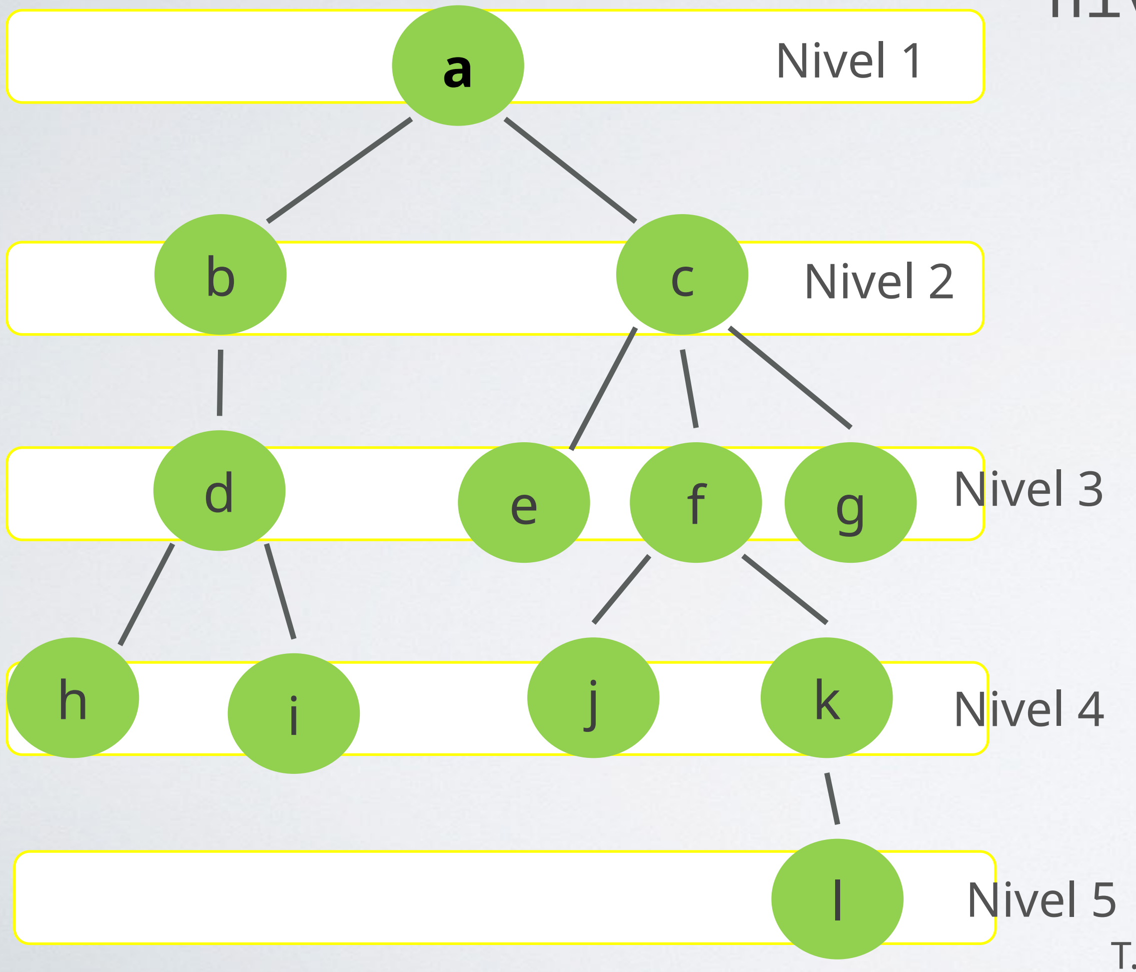


Care este inaltimea si adancimea nodului 10 ?
Care este inaltimea si adancimea nodului 4 ?

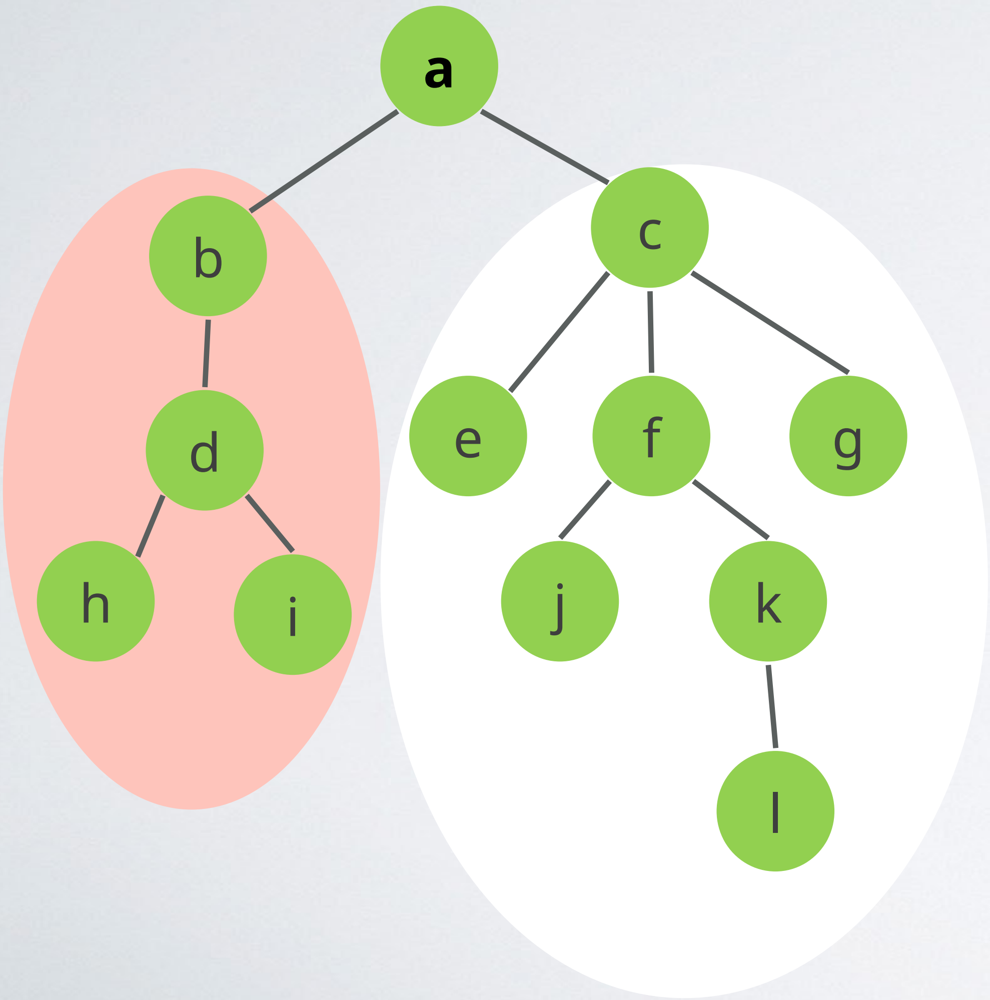
TERMINOLOGIA PENTRU ARBORI – NIVEL

Nivelul unui varf v∈V este

`nivel(v) = 1 + adancime(`*v*`)`



TERMINOLOGIA PENTRU ARBORI – SUBARBORE



Sub-arborele generat de un varf $v \in V$ este un arbore care consta in nodul radacina v si toti descendentii sai din T.

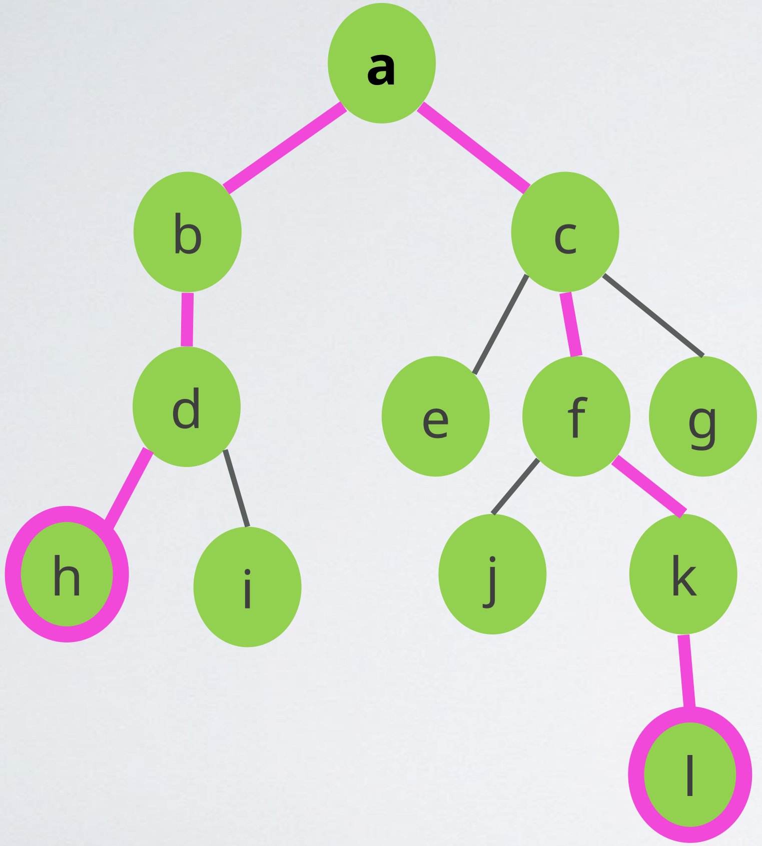
Subarborele generat de b contine nodurile:

- b,d,h,i

Subarborele generat de f contine nodurile:

- f,j,k,l

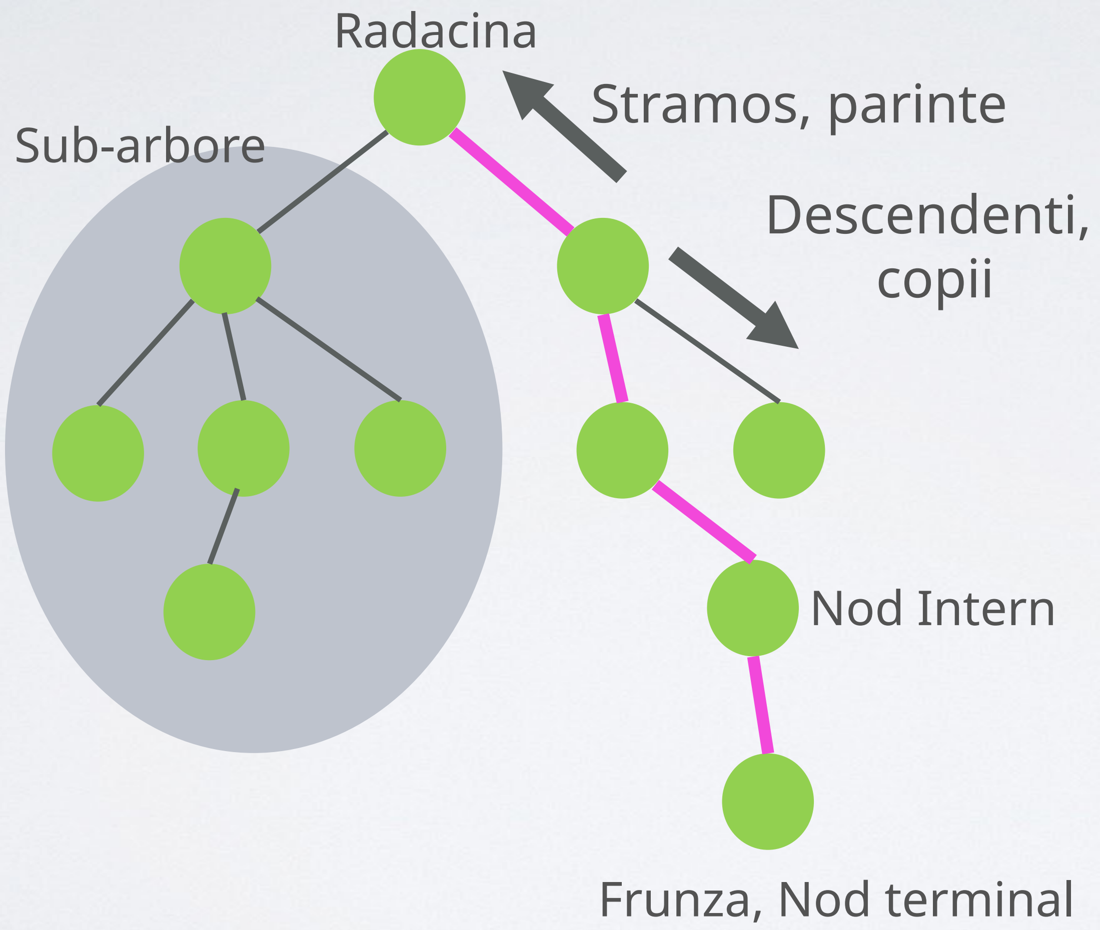
TERMINOLOGIA PENTRU ARBORI – DIAMETRU



Diametrul unui arbore: lungimea maxima a unei cai intre 2 noduri (frunze), in arbore

Exemplu: diametrul arborelui din figura este 7

TERMINOLOGIA FOLOSITA PENTRU ARBORI



TERMINOLOGIA PENTRU ARBORI OARECARE

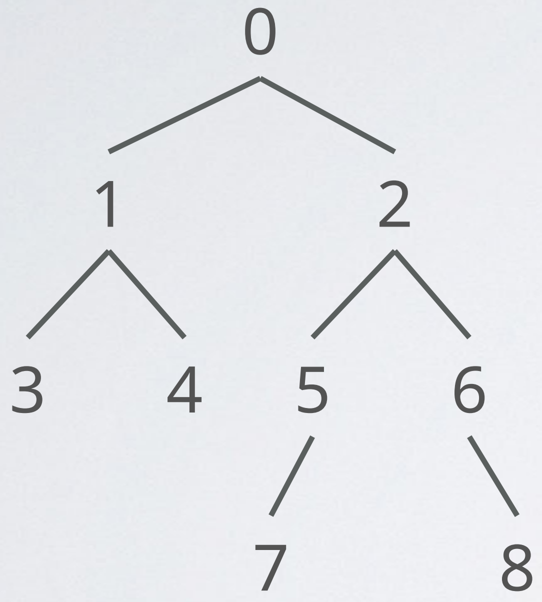
- Un arbore oarecare este:
  - ***m-ary*** daca fiecare varf intern are cel mult *m* fii.
    - $m = 2 \rightarrow$ **arbore binar**; $m = 3 \rightarrow$ arbore ternar
  - ***m-ary intreg ("full")*** – fiecare nod intern are exact m fii
  - ***complet m-ary*** – daca este arbore full si toate frunzele sunt la acelasi nivel
- Limite:
- Inaltimea maxima a unui arbore cu n varfuri este $n - 1$.
- Inaltimea maxima a unui arbore plin (full) cu n varfuri este $(n - 1)/m$
- Inaltimea minima a unui arbore cu n varfuri este $\lfloor \log_m n \rfloor$

ADT (ABSTRACT DATA TYPE) TREE

- `parent(`*n*`, `*T*`)`: returneaza parintele nodului n in arborele T. Pentru radacina returneaza un arbore vid (NIL).
  - Input: nod, arbore; Output: nod sau NIL
- `leftmostChild(`*n*`, `*T*`)`: returneaza fiul cel mai din stanga al nodului n din arborele T sau NIL pentru o frunza.
  - Input: nod, arbore; Output: nod sau NIL
- `rightSibling(`*n*`, `*T*`)`: returneaza fratele din dreapta al nodului n in arborele T sau NIL pentru cel mai din dreapta frate.
  - Input: nod, arbore; Output: nod sau NIL
- `label(`*n*`, `*T*`)`: returneaza eticheta (valoarea asociata) nodului n in arborele T
  - Input: nod, arbore; Output: eticheta
- `root(`*T*`)`: returneaza radacina arborelui *T*
  - Input: arbore; Output: nod sau NIL
- *`inord(T), preord(T), postord(T)`*

IMPLEMENTAREA ARBORILOR CU VECTORI

Model logic



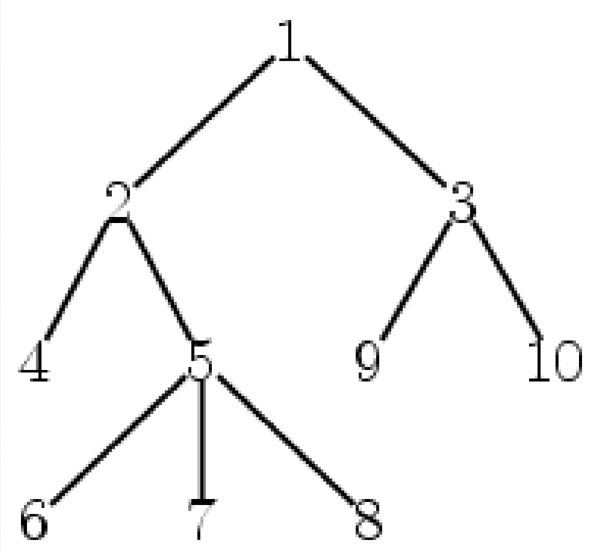
Structura fizica

idNod	0	1	2	3	4	5	6	7	8
Indicele parintilor	-1	0	0	1	1	2	2	5	6

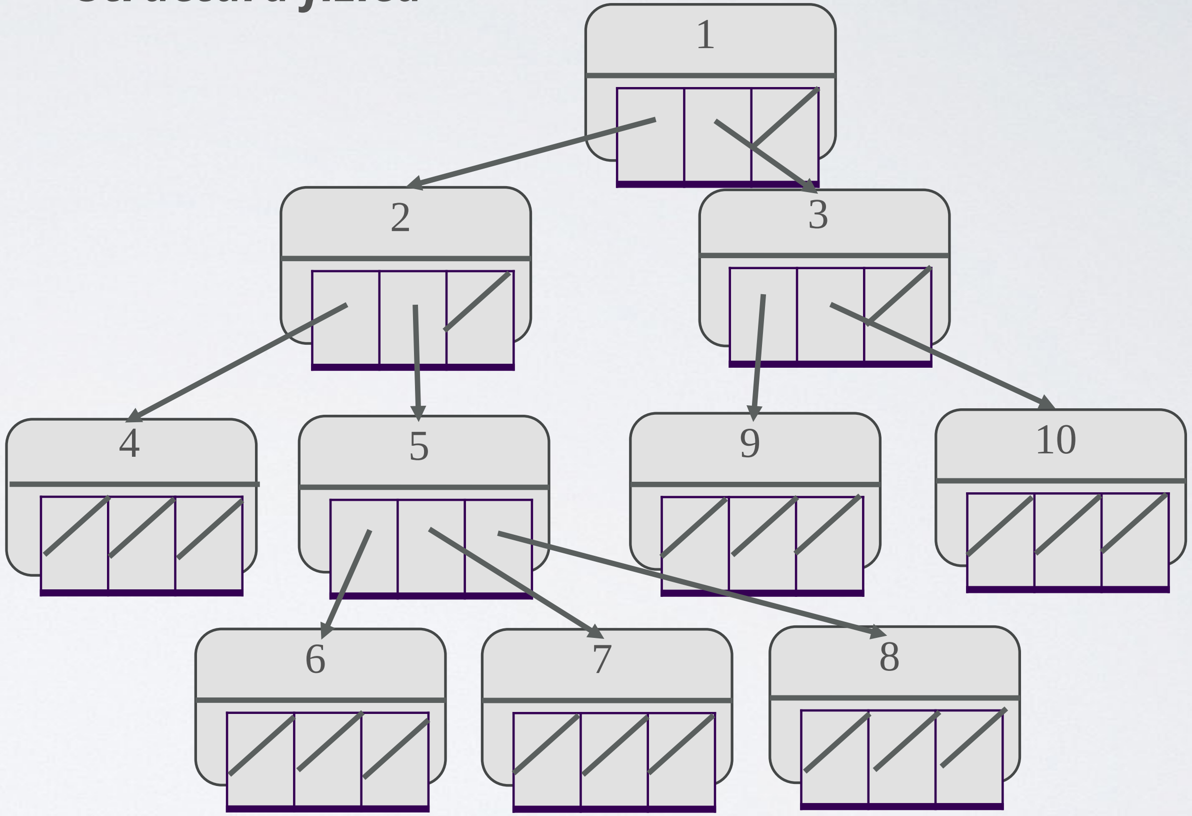
- fiecare nod are referinta catre indexul nodului parinte stocat in vector
- Indicele radacinii este -1

IMPLEMENTAREA ARBORILOR. LISTE DE COPII

Model logic



Structura fizica

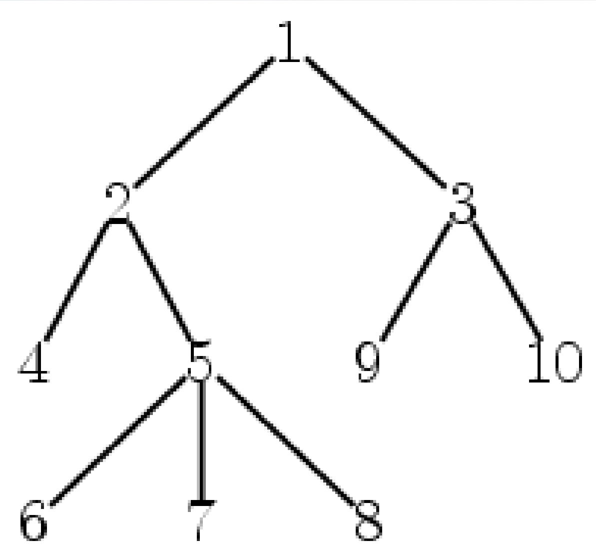


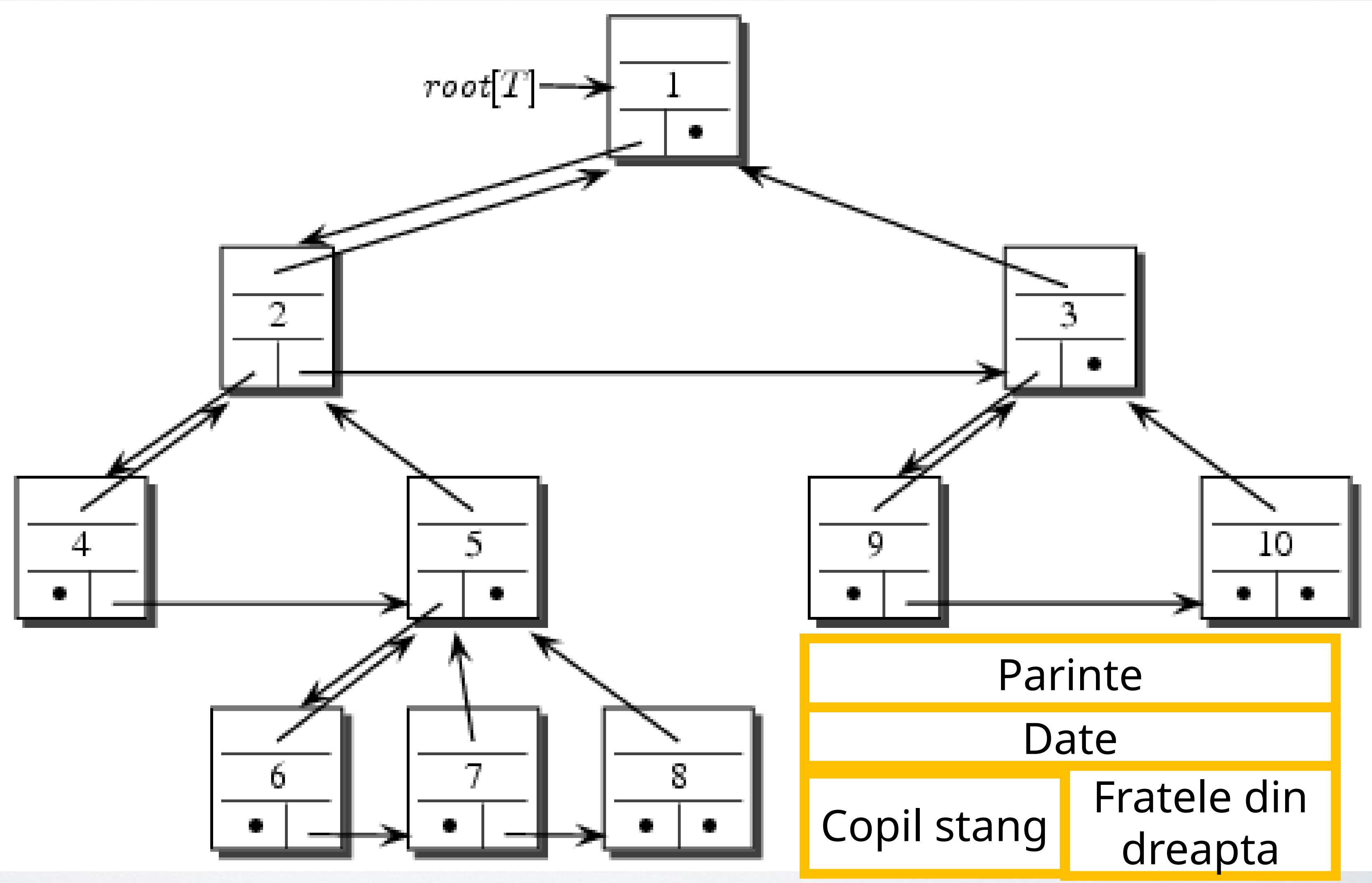
Date
Lista de referinte catre noduri copil

Lista de copii – sir sau lista inlantuita

REPREZENTARE DE ARBORE BINAR A ARBORILOR MULTICAI

Model logic



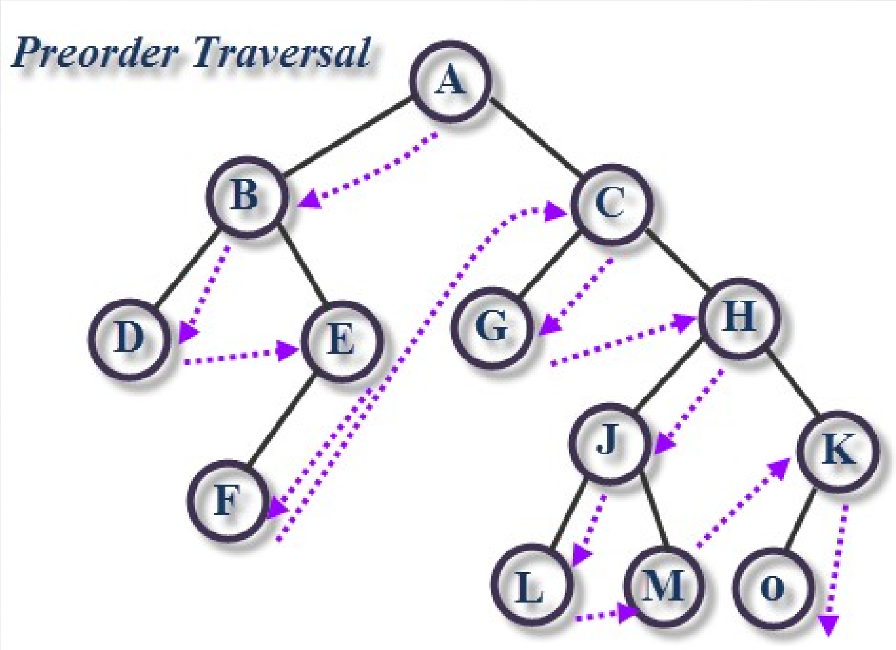


PARCURGERILE UNUI ARBORE

Preordine – se viziteaza radacina, apoi tot in preordine se viziteaza nodurile subarborilor care au ca parinte radacina, incepand cu sub-arborele cel mai din stanga.

`Preordine (`*n*`)`

- proceseaza *n*
- pentru fiecare fiu c al lui *n*, in ordine de la cel mai din stanga fiu executa `Preordine(`*c*`)`



Sursa foto: *

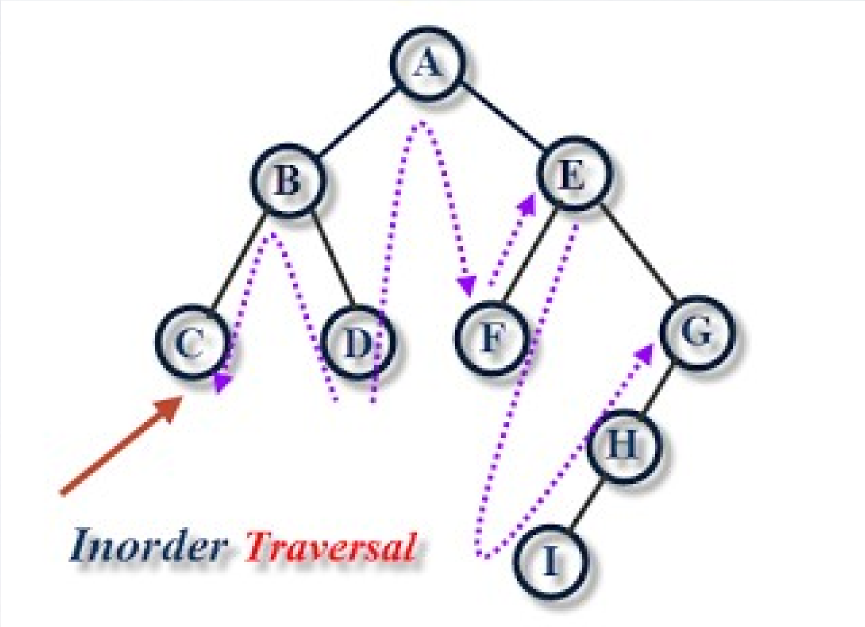
*http://techfinite.blogspot.ro/2013/12/binary-tree-traversals-and-tree-iterations.html

PARCURGERILE UNUI ARBORE

Inordine – se viziteaza in inordine primul copil, dupa care se proceseaza radacina, dupa care se viziteaza, in inordine, restul copiilor

`Inordine (n)`

- `Inordine(fiul cel mai din stanga a lui n)`
- proceseaza n
- pentru fiecare fiu c al lui n, exceptie facand nodul cel mai din stanga, in ordine de la stanga la dreapta se executa `Inordine(c)`



Sursa foto: *

Complexitate : O(n)
Inorder traversal : C B D A F E I H G

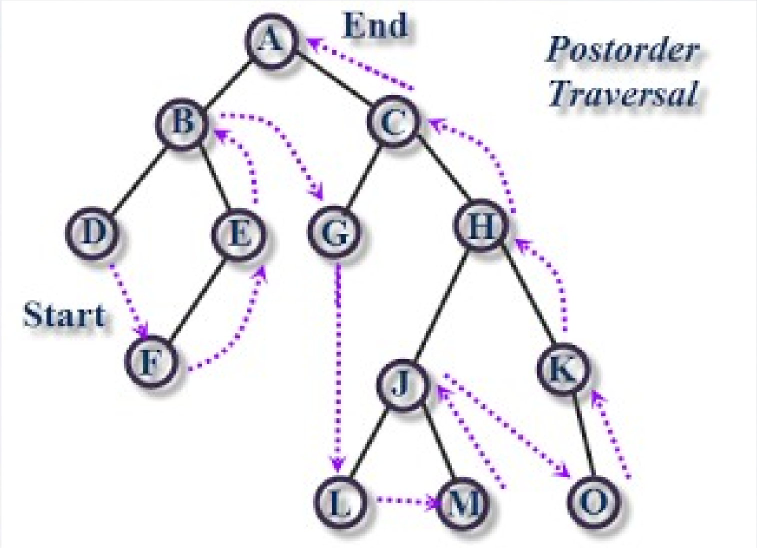
*http://techfinite.blogspot.ro/2013/12/binary-tree-traversals-and-tree-iterations.html

PARCURGERILE UNUI ARBORE

Postordine– pentru un nod se viziteaza in postordine toti sub-arborii care au ca radacini pe fii nodului dat, apoi se viziteaza nodul. Se incepe parcurgerea de la radacina.

`Postordine(n)`

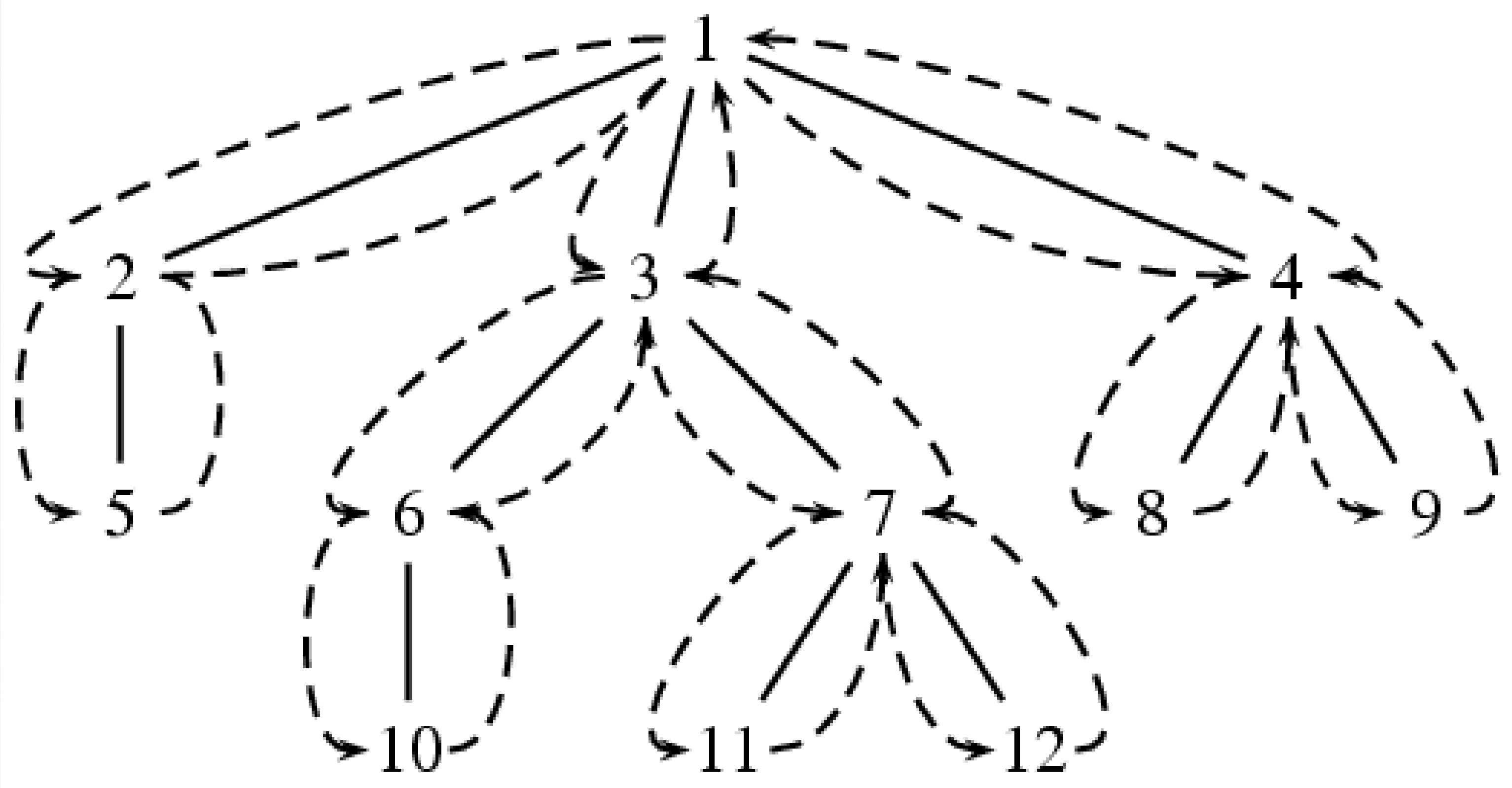
- pentru fiecare fiu c al lui n, executa `Postordine(c)`
- proceseaza n



Sursa foto: *

*http://techfinite.blogspot.ro/2013/12/binary-tree-traversals-and-tree-iterations.html

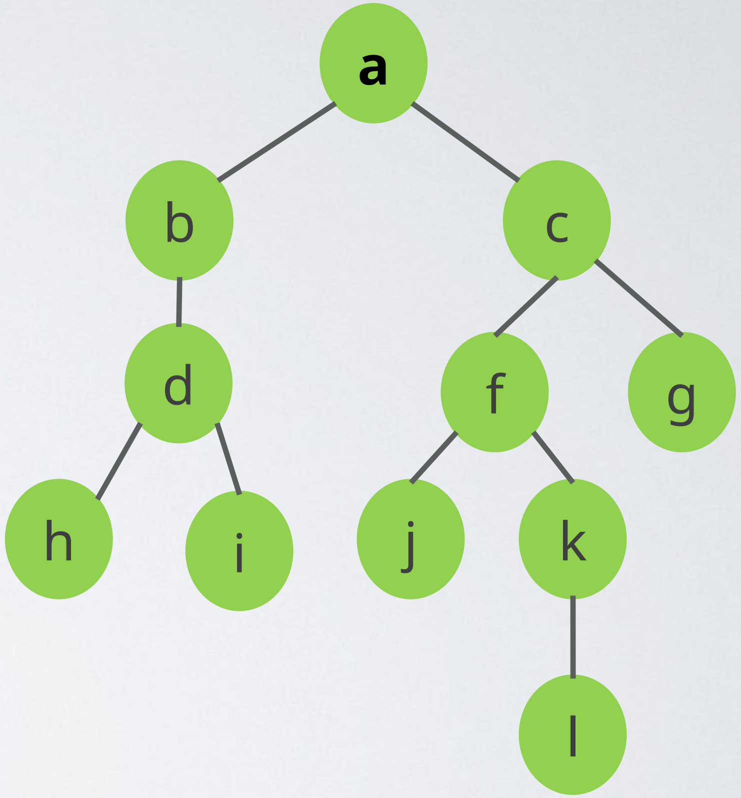
EXEMPLU DE PARCURGERI



- preordine: 1, 2, 5, 3, 6, 10, 7, 11, 12, 4, 8, 9.
- postordine: 5, 2, 10, 6, 11, 12, 7, 3, 8, 9, 4, 1.
- inordine: 5, 2, 1, 10, 6, 3, 11, 7, 12, 8, 4, 9.

ARBORE BINAR

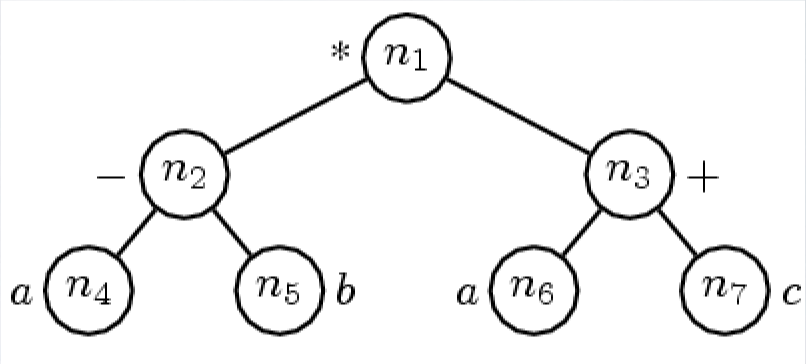
- **Tip de data recursiv (ADT):**
  - *NIL* (NULL)
  - nod, denumit radacina, impreuna cu doi arbori binari - subarborele stang (*left*) si subarborele drept (*right*)



- **Structura: reprezentare inlantuita:**
  - campurile cheie, *left (stang)*, *right(drept)*, optional si *p(parinte)*

ARBORI ETICHETATI SI ARBORI PENTRU EXPRESII

- Arborii binari se pot folosi pentru a reprezenta expresii precum:
  - Propozitii compuse
  - Combinatii de multimi
  - Expresii aritmetice



(a-b)*(a+c)

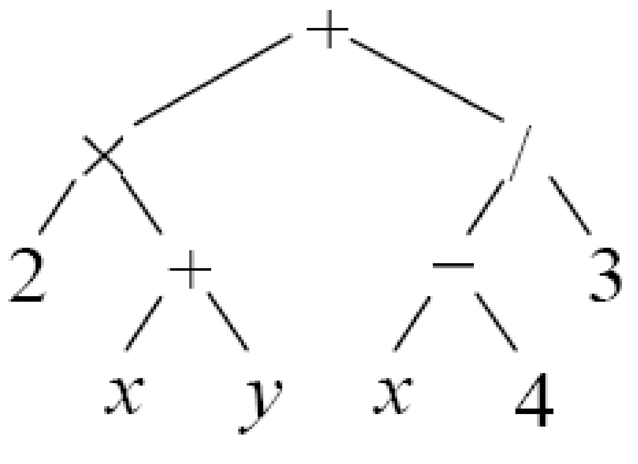
- **Arbore etichetat:** fiecare nod are asociata o eticheta sau o valoare
- Arbore pentru expresii aritmetice: nodurile interne reprezinta operatori si frunzele sunt operanzi

  Operator binar: primul operand este pe frunza stanga iar al doilea operand este pe frunza dreapta
  Operatori unari: un singur operand pe frunza dreapta

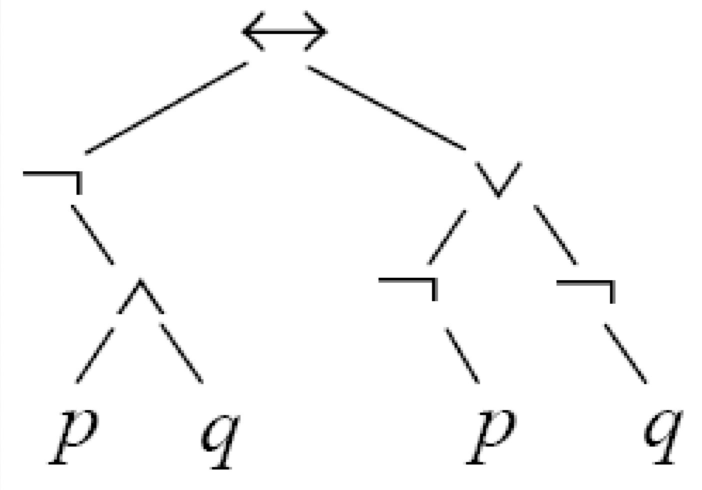
FORME PREFIX, POSTFIX, INFIX

- Folosind arborii binari se pot obtine expresii aritmetice in trei reprezentari:
  - Forma infixata:
    - Parcurgere in inordine
    - Se folosesc parantezele pentru a evita ambiguitatile
  - Forma prefixata:
    - Parcurgere in preordine
    - Nu sunt necesare parantezele
  - Forma postfixa:
    - Se foloseste parcurgerea in postordine
    - Nu sunt necesare parantezele
- Expresiile in forma prefixata si postfixa sunt folosite in stiinta calculatoarelor.

EXEMPLU DE ARBORE PENTRU EXPRESII:



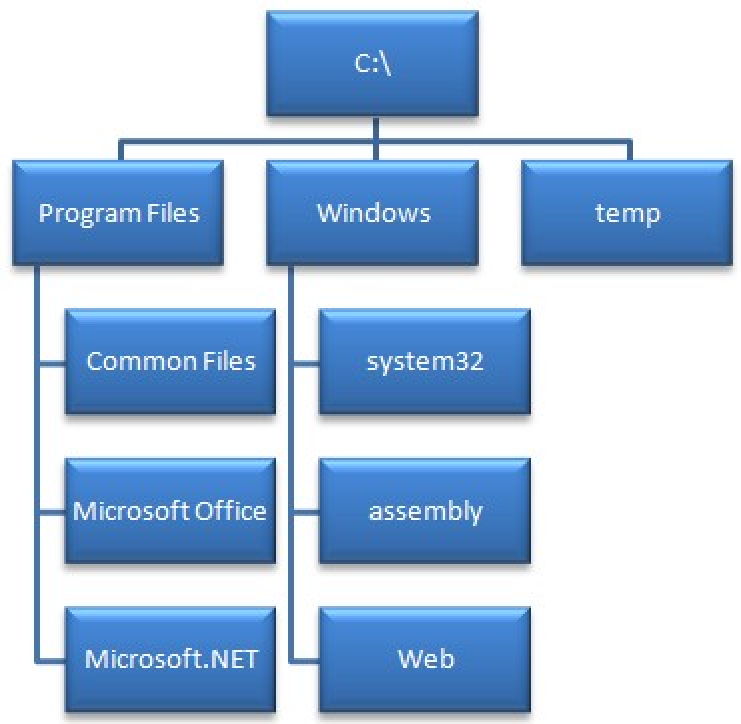
infix:	$(2 \times (x + y)) + ((x - 4) / 3)$
prefix:	$+ \times 2 + x\ y / - x\ 4\ 3$
postfix:	$2\ x\ y + \times x\ 4 - 3 / +$



infix:	$(\neg(p \wedge q)) \leftrightarrow (\neg p \vee \neg q)$
prefix:	$\leftrightarrow \neg \wedge p\ q \vee \neg p \neg q$
postfix:	$p\ q \wedge \neg p \neg q \neg \vee \leftrightarrow$

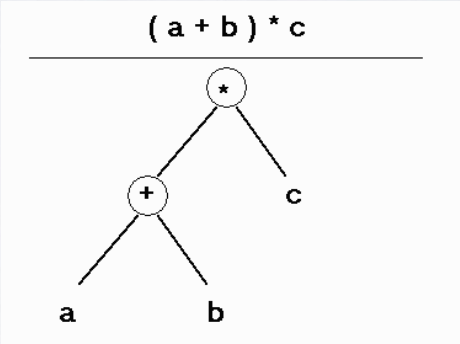
ARBORI OARECARE APLICATII

Fisierele unui system de operare



Sursa foto https://dvanderboom.wordpress.com

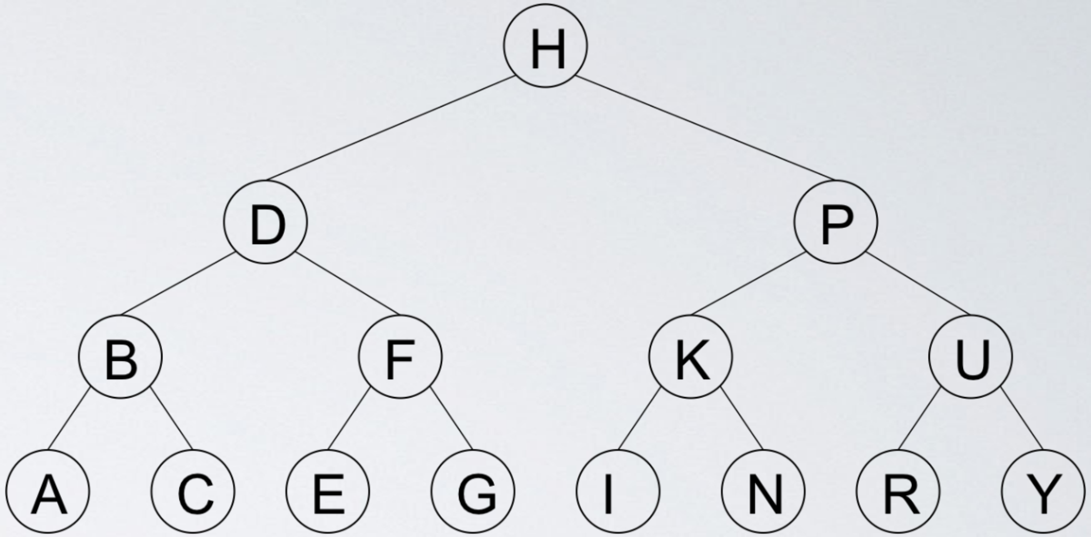
Evaluarea expresiilor:



Compresia datelor (e.g. Huffman coding) – later in course

Natural language processing (e.g. syntax, dependency trees)

ARBORI BINARI DE CAUTARE

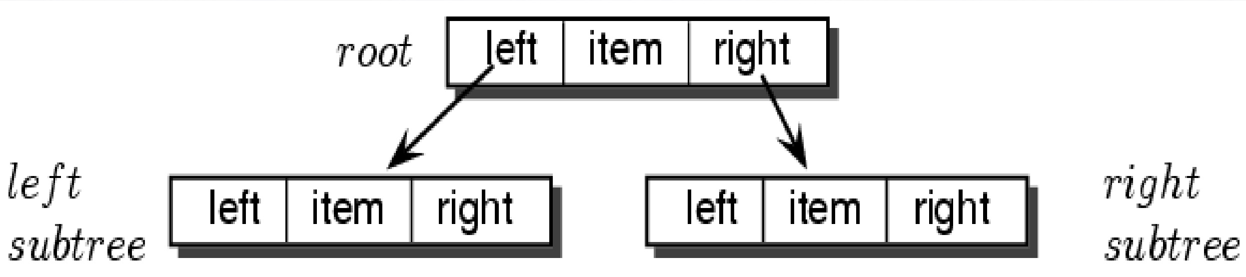


- **Definitie:**
  - arbore binar, chei care pot fi comparate (relatie de ordine)
  - nodurile cu chei mai mici decat valoarea *x* a cheii asociate unui anumit nod se gasesc in subarborele stang al acestuia
  - nodurile ale caror chei au valori mai mari decat *x* se gasesc in subarborele său drept
  - Subarborele stâng şi subarborele drept al oricărui nod sunt şi ei arbori binari de căutare.

Operatii:
Tree-search (T, key)
Tree-insert(T, key)
Tree-delete(T, node)
traverse - inorder, preorder, portorder (T)
height(T), diameter(T), Tree-successor(node), Tree-predecessor(node), etc…

IMPLEMENTARE ARBORE BINAR DE CAUTARE

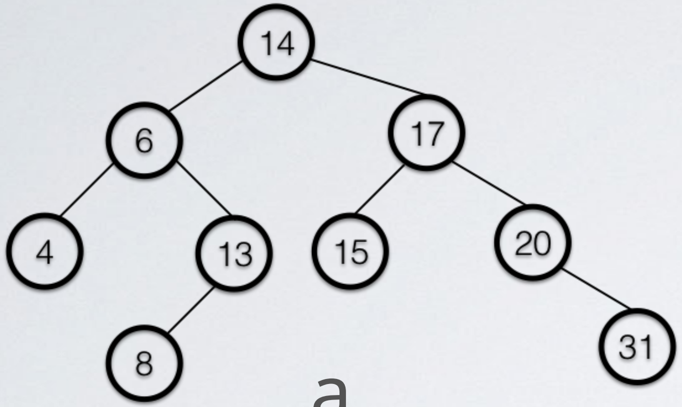
```
typedef struct treeNode{
    int key;
    struct treeNode *left;
    struct treeNode *right;
    struct treeNode *parent; //optional
} TreeNode;
```



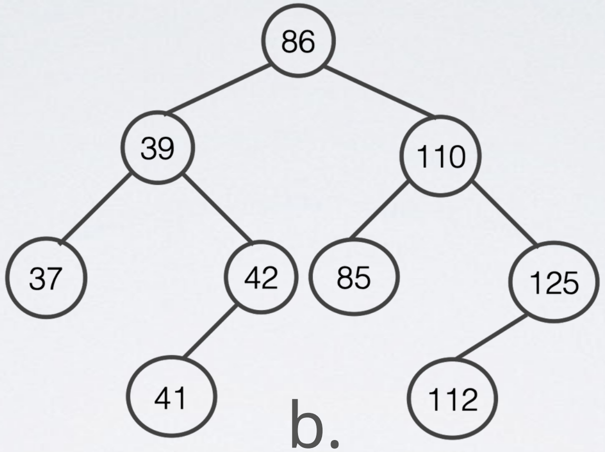
PAUSE AND EVALUATE ...

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

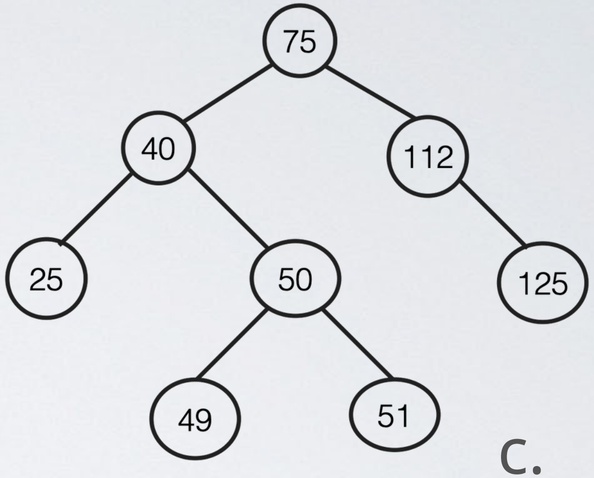
Care dintre arborii de mai jos NU este ABC?



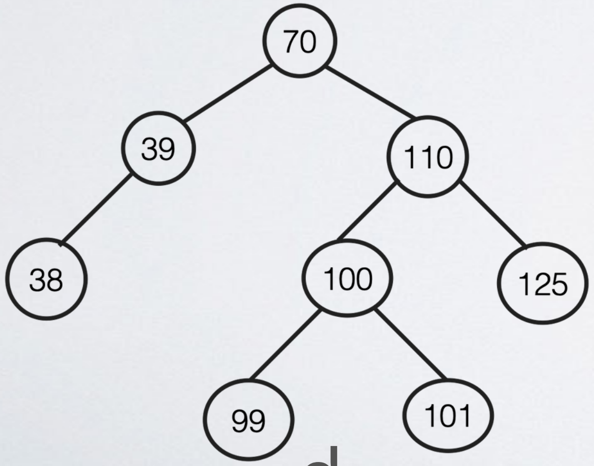
a.



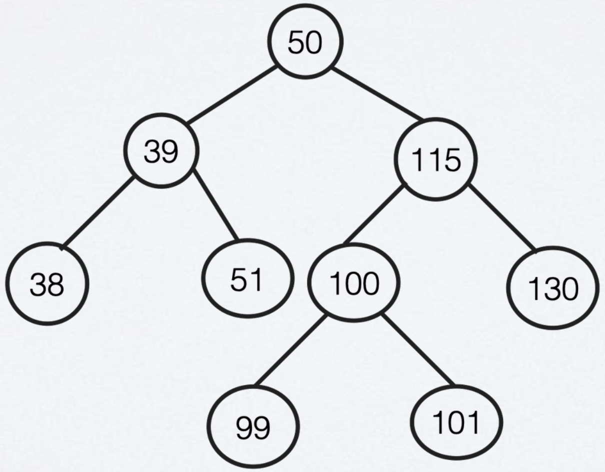
b.



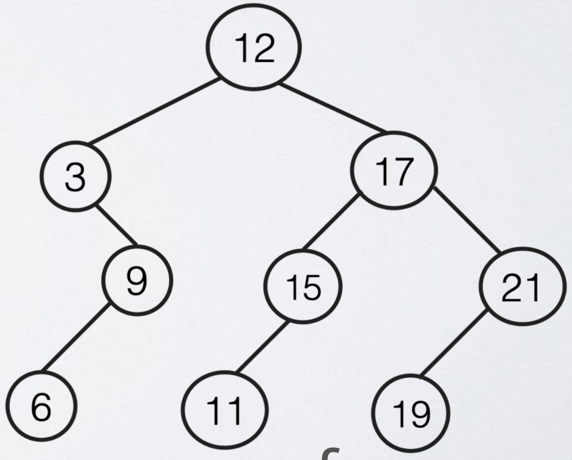
c.



d.



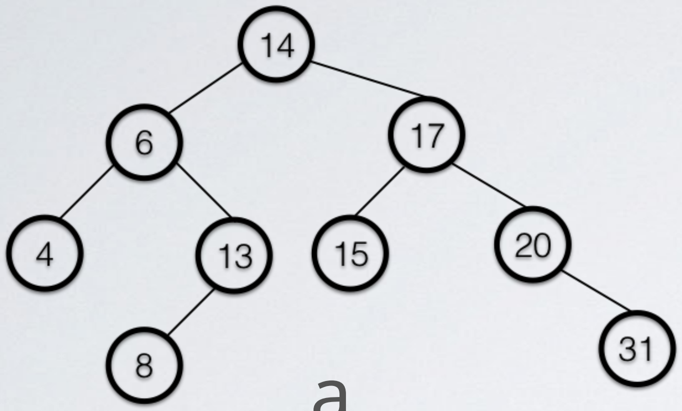
e.



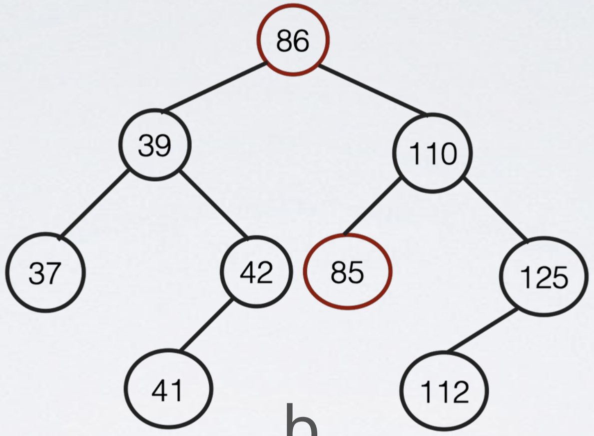
f.

PAUSE AND EVALUATE ...

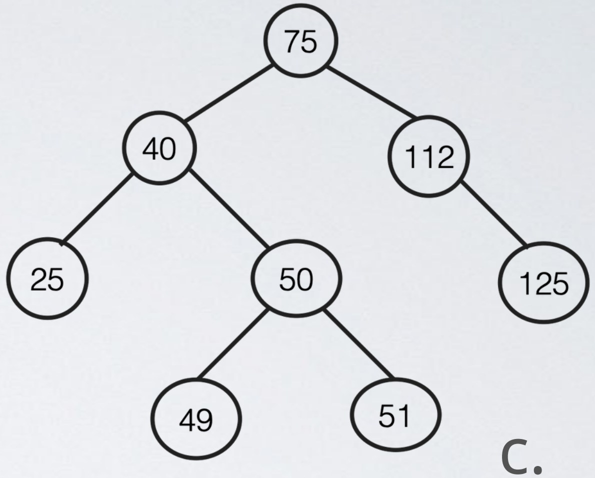
Care dintre arborii de mai jos NU este ABC?



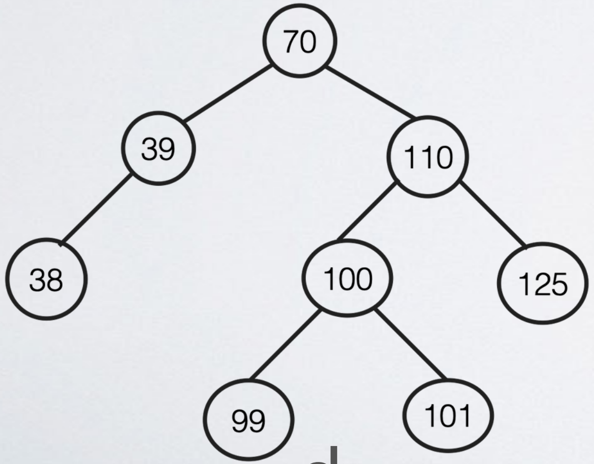
a.



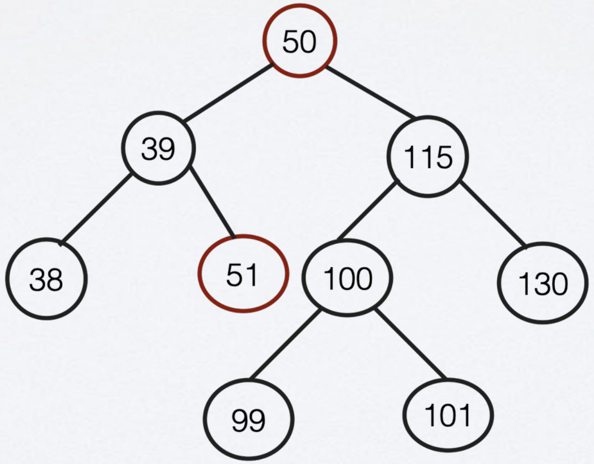
b.



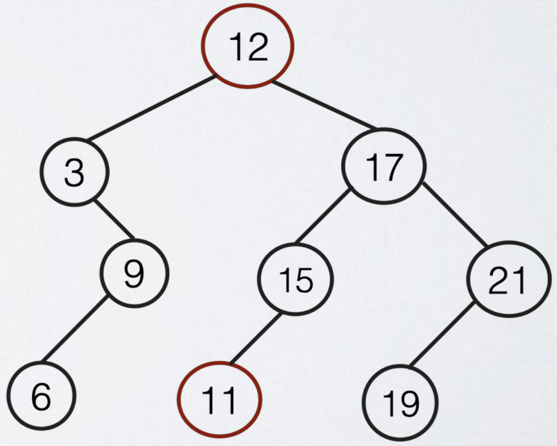
c.



d.



e.



f.

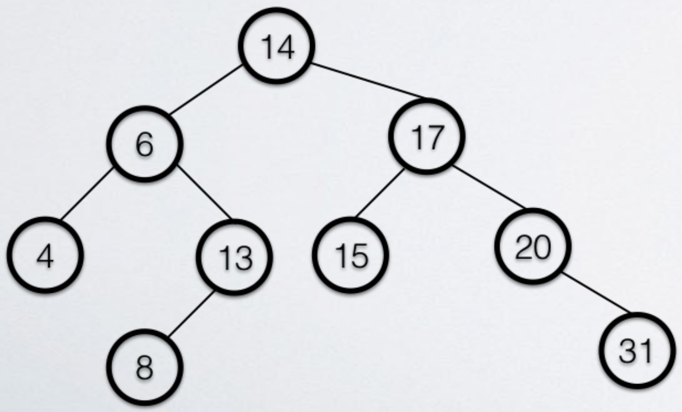
EXEMPLU PSEUDOCOD: PREORDINE

Varianta recursiva

```
preorder(node)
  if node = NIL then
    return
  visit(node)
  preorder(node.left)
  preorder(node.right)
```

Varianta iterativa

```
preorder(node)
s ← empty stack
if node ≠ NIL then
    s.push(node)
while not s.isEmpty()
    node ← s.pop()
    visit(node)
    if  node.right ≠ null then
      s.push(node.right)
    if node.left ≠ null then
      s.push(node.left)
```



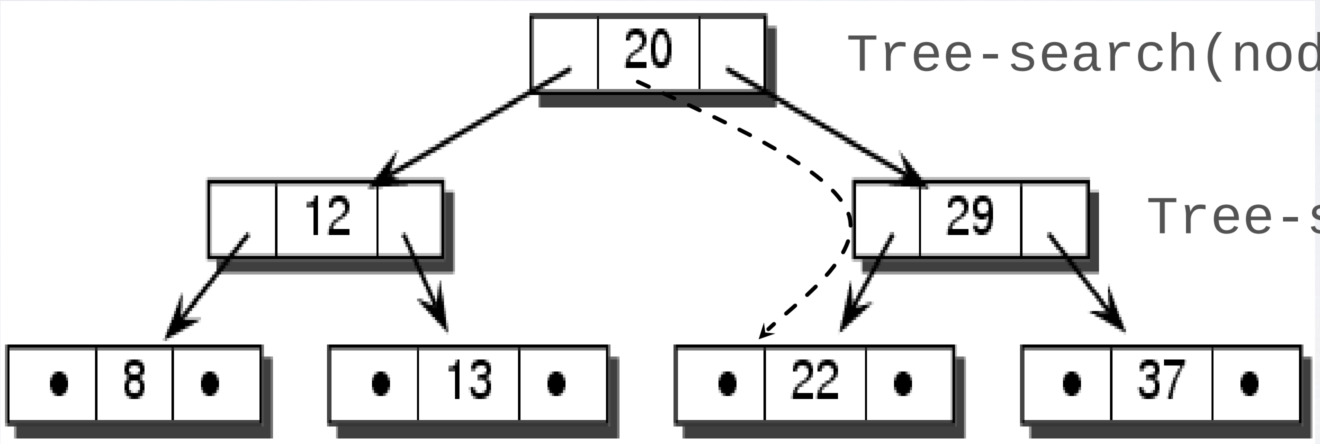
? Care sunt secventele generate de parcurgerile in:

- preordine
- inordine
- postordine

? Care este complexitatea unei parcurgeri?

OPERATIA DE CAUTARE

- E.g.
  - Tree-search(T, 22)




```
Tree-search(node.right, key)
```

```
Tree-search(node.left, key)
```

```
return n->item;
```

OPERATIA DE CAUTARE - ALGORITM

Varianta recursiva

```
Tree-search(node, key)
    if node = NIL then
        return NIL
    if node.key = key then
        return node
    else if key < node.key then
        return Tree-search(node.left, key)
    else
        return Tree-search(node.right, key)
```

Varianta iterativa

```
Tree-search(node, key)
    crt ← node
    while crt != NIL and crt.key != key do
        if key < crt.key then
            crt ← crt.left
        else
            crt ← crt.right
    return crt
```

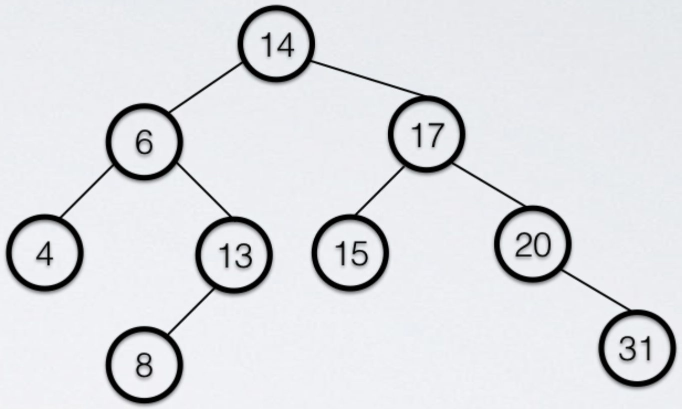
Performanta?

PERFORMANTA OPERATIEI DE CAUTARE

- Inaltime h
  - Noduri parcurse pe un drum de la radacina la o frunza
  - Avem nevoie de cel mult $h+1$ comparatii, deci $\mathrm{O}(h)$
- caz favorabil
- caz mediu:
  - pp. arborele aproximativ echilibrat: $O(\log n)$
- cazul defavorabil: O(n)

ALTE OPERATII DE CAUTARE

- cautarea nodului minim
- cautarea nodului maxim
- cautarea predecesorului
- cautarea succesorului



E.g.
- succesorul nodului cu cheia 14 este nodul cu cheia 15
- succesorul nodului cu cheia 13 este nodul cu cheia 14

Inordine: 4, 6, 8, 13, 14, 15, 17, 20, 31

SUCCESORUL UNUI NOD

```
Procedure Tree-successor(x)
if x.right != NIL then return Tree-minimum(x.right)
y ← x.parent
while y != NIL and x == y.right do
        x ← y
        y ← y.parent
return y
```

```
Procedure Tree-minimum(x)
while x.left != NIL do
        x ← x.left
return x
```

Pentru pseudocodul tuturor operatiilor de cautare:
see Th. Cormen, *Introduction to Algorithms*, 3rd edition, *pp. 295-298*

SUCCESORUL UNUI NOD

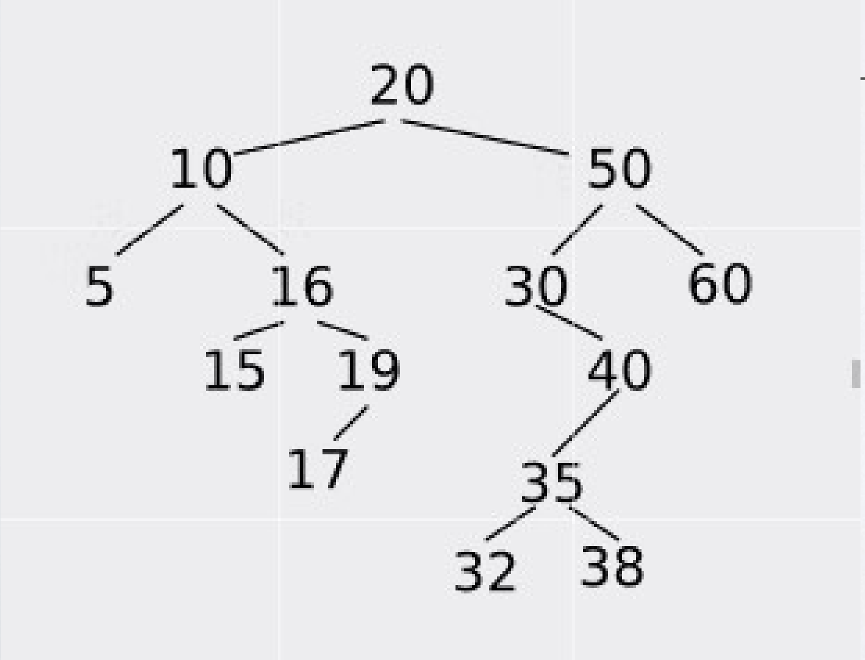
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Procedure Tree-successor(x)

if x.right != NIL **then** return *Tree-minimum*(x.right)
y ← x.parent
while y != NIL and x == y.right **do**
    x ← y
    y ← y.parent
return y

Procedure Tree-minimum(x)

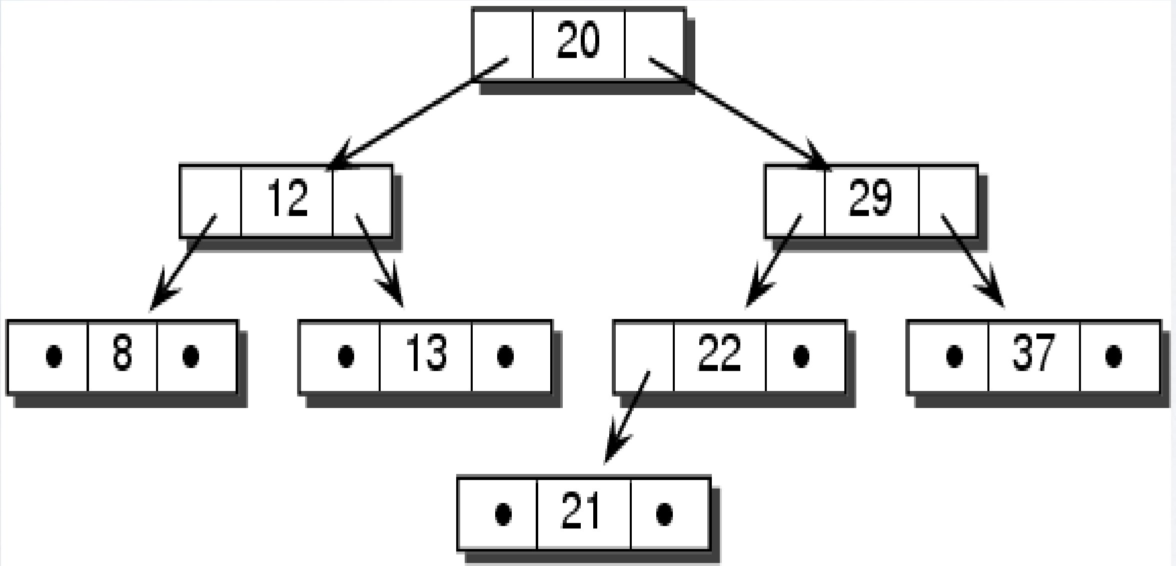
while x.left != NIL **do**
    x ← x.left
return x



Care este succesorul nodului cu cheia 10 ?
Care este succesorul nodului cu cheia 19 ?
Care este succesorul nodului cu cheia 30 ?

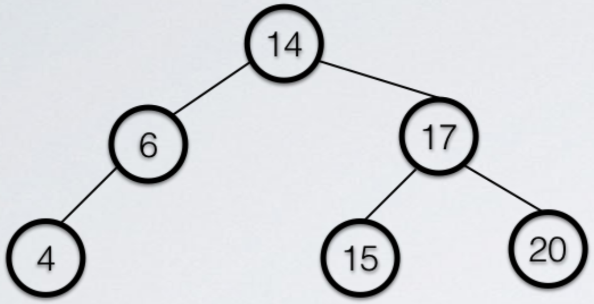
INSERAREA UNUI NOD

- **Intotdeauna ca frunza** !!!
- Se insereaza nodul cu cheia 21 in arbore

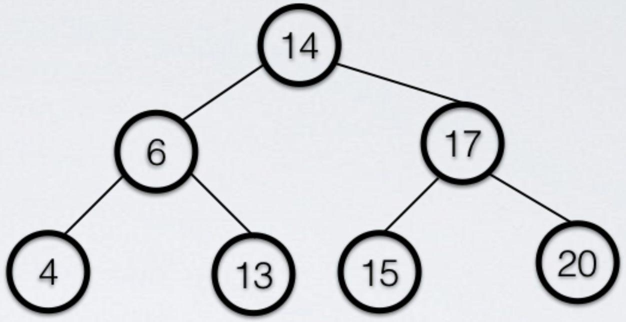


- Cautam pozitia corecta (intotdeauna frunza!)
- Cream nodul
- Il legam in arbore

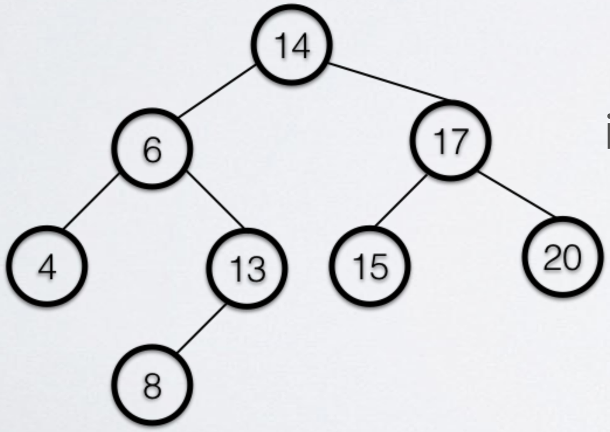
EXEMPLU INSERARE



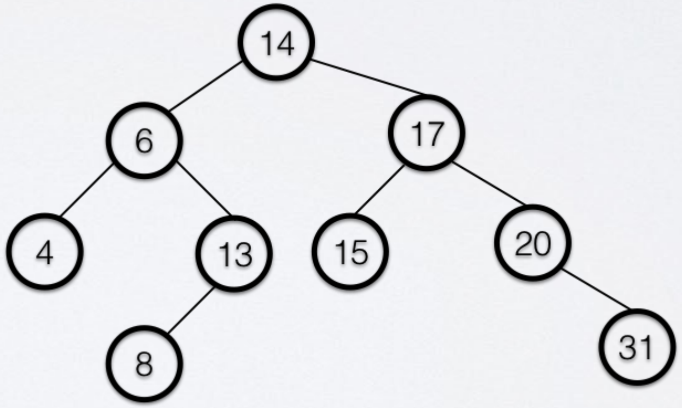
insert(13)



insert(8)



insert(31)



OPERATIA DE INSERARE - ALGORITM

Varianta recursiva

```
Tree-insert(node, key)
   if node = NIL then
        return createNode(key)
   else if key < node.key then
       node.left ← Tree-insert(node.left, key)
   else
       node.right ← Tree-insert(node.right, key)
   return node
```

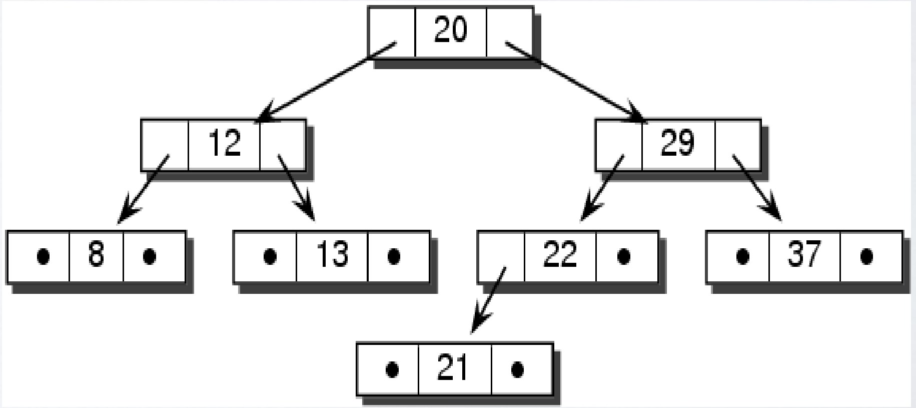
Codul aproape identic cu codul de cautare!
Complexitate?

Varianta iterativa

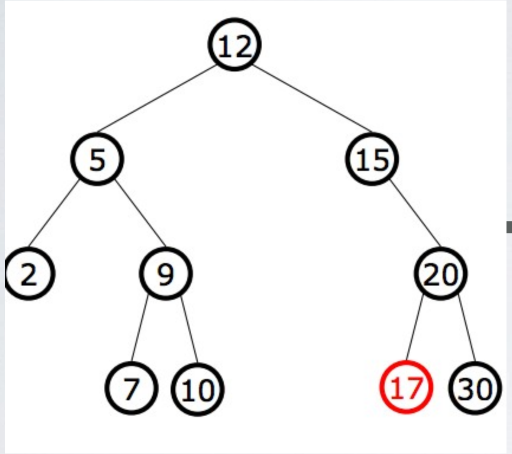
```
Tree-insert(node, key)
   crt ← node
   parent ← NIL
   dir ← NONE
   while crt != NIL do
        parent ← node
        if key < crt.key then
          crt ← crt.left
          dir ← LEFT
    else
          crt ← crt.right
          dir ← RIGHT
    if parent != NIL then
        if dir = LEFT then
          parent.left ← createNode(key)
        else
          parent.right ← createNode(key
    else
          node ← createNode(key)
```

STERGEREA UNUI NOD

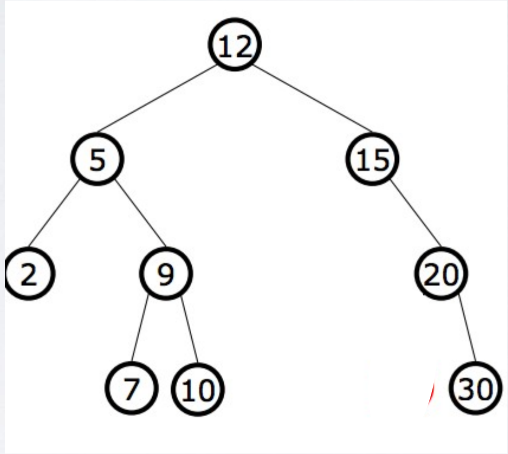
- Mai dificila decat inserarea!
- Idee:
  - se cauta nodul de sters
  - se elimina din structura
  - se reface proprietatea de arbore binar de cautare
- Cazuri pentru stergere:
  1. Nod terminal (frunza)
  2. Nod cu un singur fiu
  3. Nod cu doi fii
- Exemplu:
  1. Stergeti 8, 13, 21 sau 37
  2. Stergeti 22
  3. Stergeti12, 20 sau 29

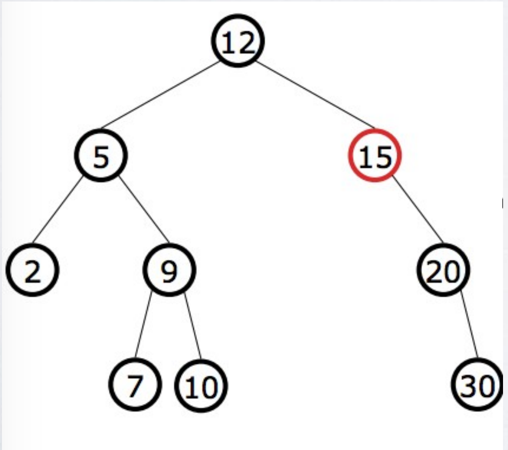


EXEMPLU STERGERE: CAZURI 1&2

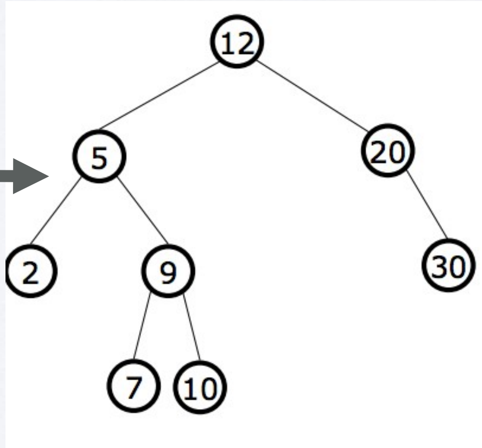


delete(17)
Nod terminal

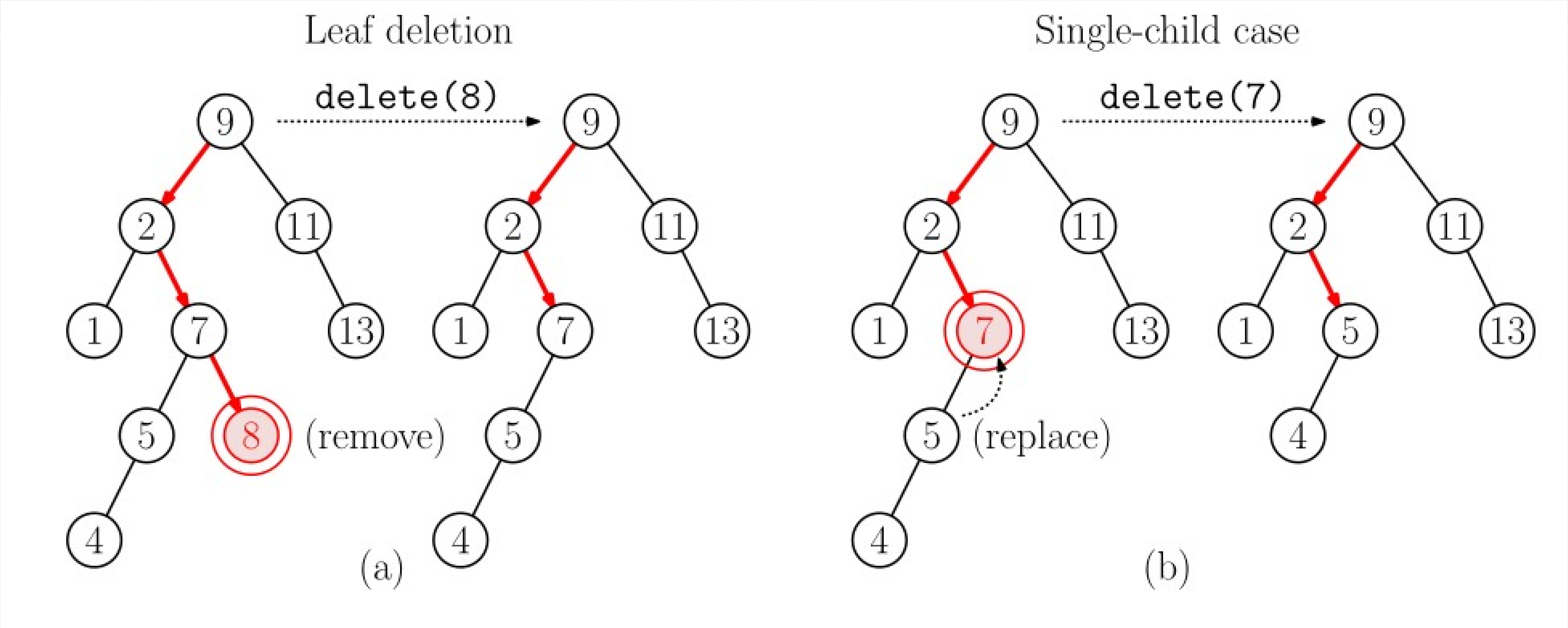




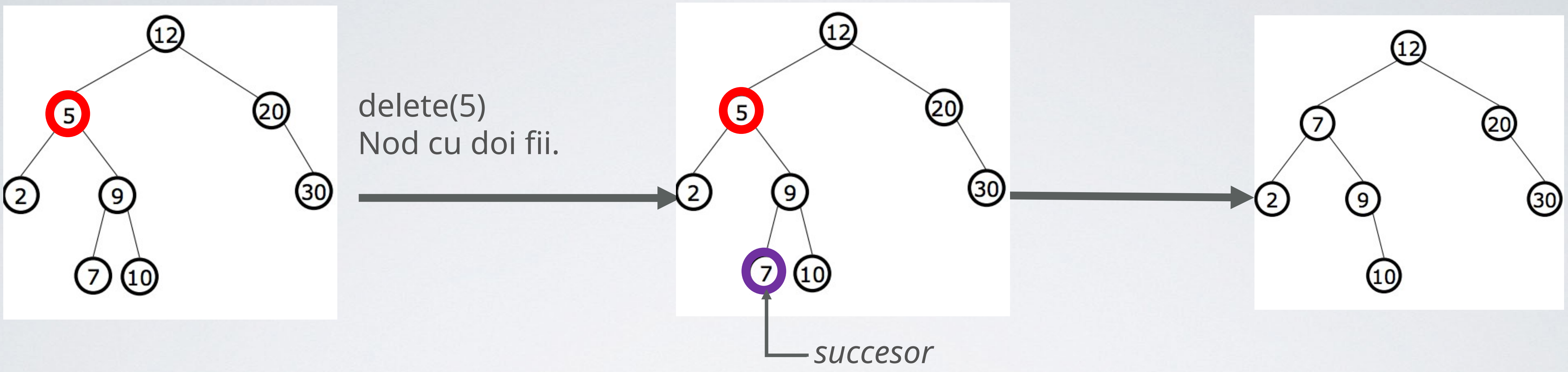
delete(15)
Nod cu un singur fiu.



EXEMPLU STERGERE: CAZURI 1&2



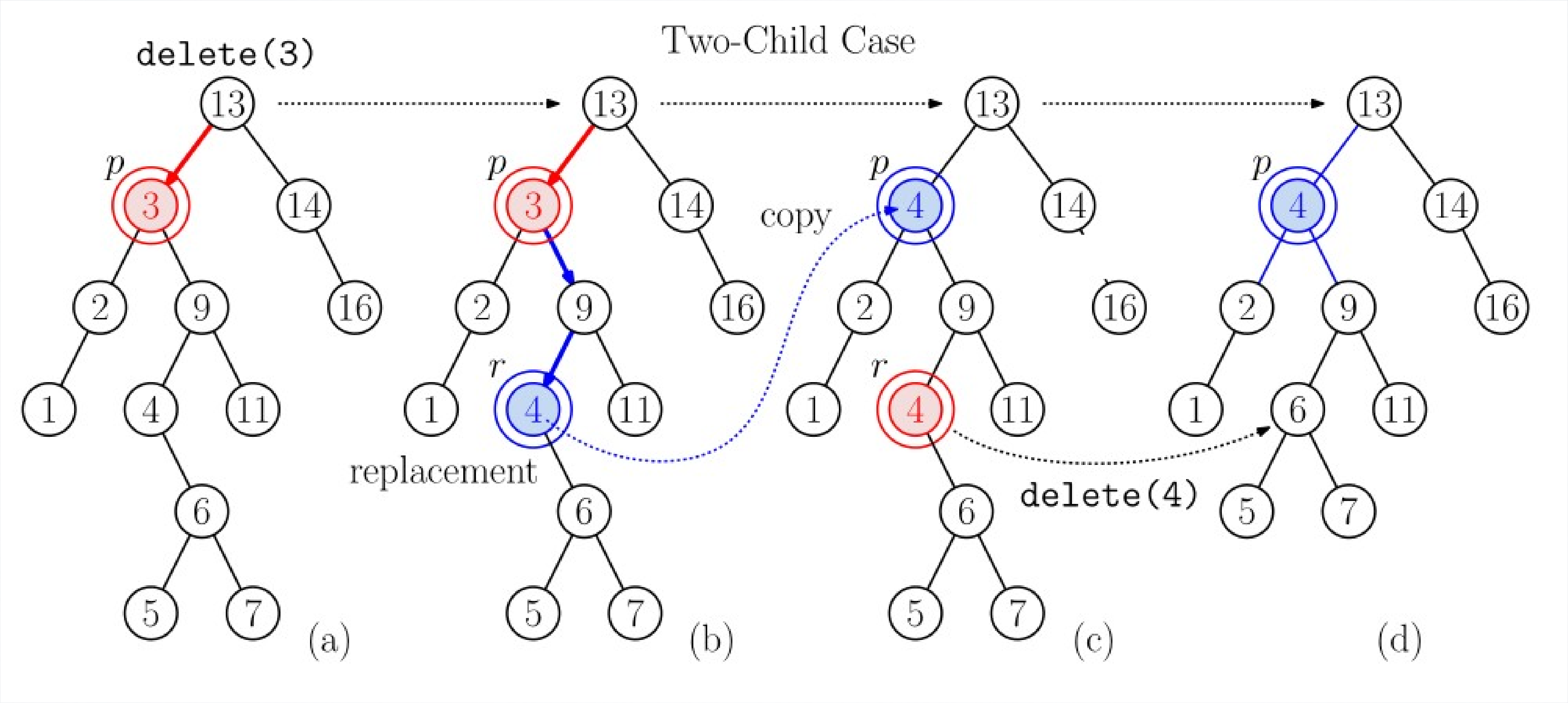
EXEMPLU STERGERE: CAZUL 3



Cum putem pastra proprietatea de ABC?

Inlocuim cu o valoare intre cei 2 copii!
- succesorul: findMin(node.right)

EXEMPLU STERGERE: CAZUL 3



FUNCTIA DE STERGERE: CAZURI REVIZITATE

Caz 1 – nodul de sters (p) nu are copii – il eliminam, si inlocuim legatura parintelui spre el sa pointeze NIL.
Caz 2 – nodul *p* are 1 copil – „urcam” acel copil in arbore, modificand legatura parintelui lui *p* sa pointeze catre copilul lui p.
Caz 3 – nodul p are 2 copii – gasim *r*, succesorul lui *p* (care sigur se gaseste in subarborele drept al lui *p*), si inlocuim pe *p* cu *r* in arbore. Apelam recursiv stergerea pt nodul r .

FUNCTIA DE STERGERE: PSEUDOCOD

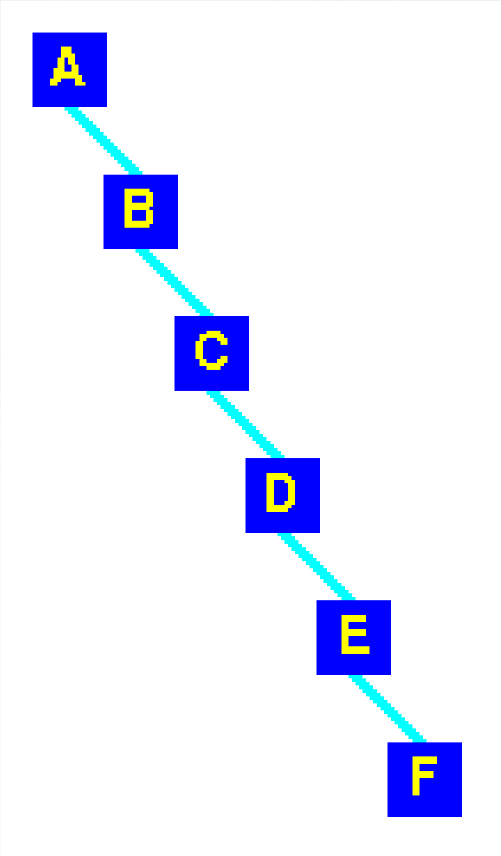
```
BSTNode delete(Key x, BSTNode p) {
    if (p == null)                                    // fell out of tree?
        throw KeyNotFoundException;                   // ...error - no such key
    else {
        if (x < p.data)                               // look in left subtree
            p.left = delete(x, p.left);
        else if (x > p.data)                          // look in right subtree
            p.right = delete(x, p.right);
                                                      // found it!
        else if (p.left == null || p.right == null) { // either child empty?
            if (p.left == null) return p.right;       // return replacement node
            else                return p.left;
        }
        else {                                        // both children present
            r = findReplacement(p);                   // find replacement node
            copy r's contents to p;                   // copy its contents to p
            p.right = delete(r.key, p.right);         // delete the replacement
        }
    }
    return p;
}
```

```
BSTNode findReplacement(BSTNode p) {
    BSTNode r = p.right;
    while (r.left != null) r = r.left;
    return r;
}
```

PERFORMANTA OPERATIILOR

- Cautare $c\,h$
- Inserare $c\,h$
- Stergere $c\,h$
- $h = log\ n$? (cazul mediu, da)
- Aparent eficient!
- Sa se construiasca un arbore cu caracterele:

  A B C D E F
- Dezechilibrat - cazul defavorabil O(n)



COMPARAREA PERFORMANTEI

	Arrays	Linked List	Trees
	Simplu **Inflexibil**	**Simplu** **Flexibil**	**Relativ simplu,** **Flexibil**
Insert	**O(1)** **O(n)** *inc sort*	**O(1)** *sort -> no adv*	
Delete	**O(n)**	**O(1) -** *any* **O(n) -** *specific*	
Search	**O(n)** **O(logn)** *binary search*	**O(n)** *(no bin search)*	**O(log n)**

PERFORMANTA OPERATIILOR

- Cum putem obtine garantia h ~ log n
  - constructia initiala
    - cheile ordonate (crescator, descrescator) ?
    - mediane?
    - inserari/stergeri ulterioare nu garanteaza mentinerea proprietatii
  - noduri inserate in ordine aleatoare
    - conditie de echilibru, care
      - asigura inaltimea e O(log n)
      - usor de intretinut la inserari/stergeri
- in curand….

APLICATII PRACTICE ALE ARBORILOR BINARI DE CAUTARE

- Randare 3D
- Indexarea bazelor de date

... dar cu garantii asupra timpilor operatiilor (i.e. ABC echilibrati)

EXERCITII ARBORILOR BINARI DE CAUTARE

1. Determinati parcurgerea in inordine si postordinepentru un arbore care are parcurgerea in preordine: A, B, C,-,-, D,-,-, E,-, F,-,-

Literele corespund nodurilor, - corespund la NULL

EXERCITII ARBORILOR BINARI DE CAUTARE

2. Se dau următoarele parcurgeri ale unui arbore binar:

- Preordine (Root-Left-Right): 10, 5, 1, 7, 40, 50
- Inordine (Left-Root-Right): 1, 5, 7, 10, 40, 50

Reconstruiti arborele binar

REFERINTE

- Th. Cormen et al – “Introduction to Algorithms”, 3rd ed: sect. 10.4, ch. 12
- S. Skiena – “The Algorithm Design Manual”: sect 3.4