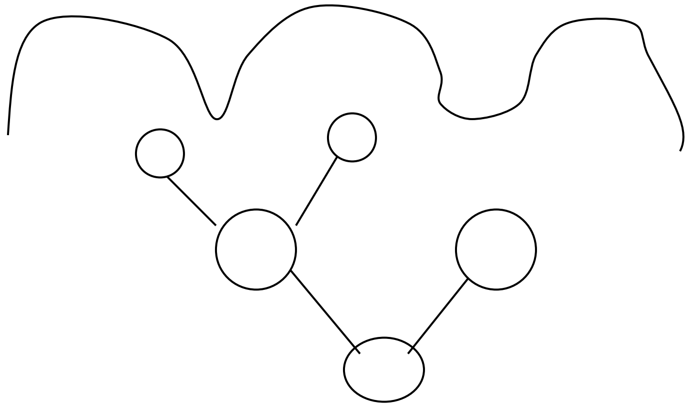
M. Caramihai, © 2020

STRUCTURI DE DATE & ALGORITMI

CURS 8

Arbori. Arbori binari



Structuri de date si arbori

Listele inlantuite sunt **structuri lineare** – si in general sunt dificil de utilizat in organizarea reprezentarii **ierarhice** a obiectelor.

Cozile si stivele permit reprezentari ierarhice dar au dezavantajul de a fi pe **o singura dimensiune**.

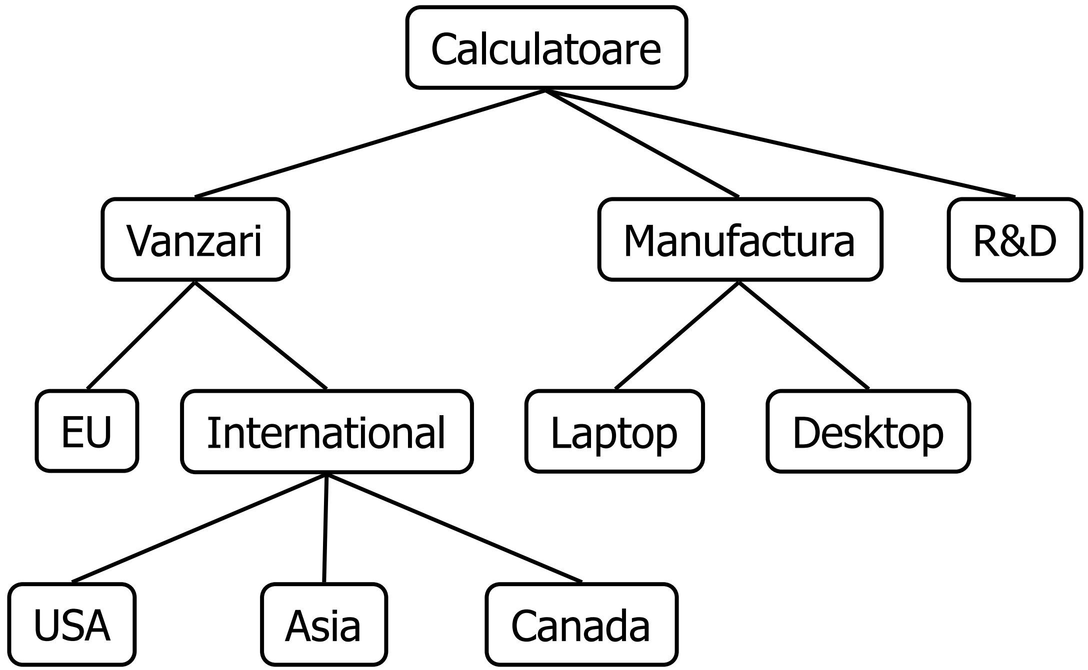
Solutie: structuri de date de tip **arbore** (format din **noduri** si **arce**)

Terminologie

Radacina: nodul radacina, primul, cel mai de sus
Grad: nr de laturi ce pornesc dintr'un nod
Frunza: orice nod fara copii
Nod intern: nod ce nu este radacina sau frunza
Nod parinte: nod cu succesor
Nod copil: nod cu predecesor
Nod frate: 2 sau mai multe noduri cu acelasi parinte
Nod ascendent: orice nod aflat intre radacina si nodul analizat
Nod descendent: orice nod aflat intre nodul analizat si frunza
Drum: o secventa de noduri dintr'un arbore
Nivelul unui nod: distanta de la radacina la un nod
Inaltimea (unui arbore): nivelul de dimensiune maxima
Subarbore: orice structura de conexiuni din cadul unui arbore, mai jos de radacina

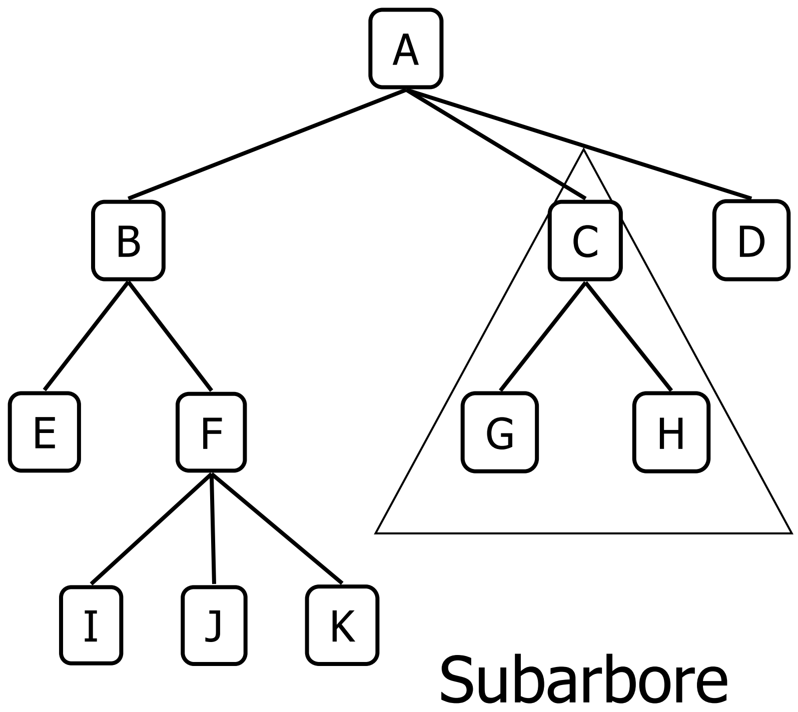
Ce este un arbore ?

- Un arbore: model abstract al unei structuri ierarhice.
- Un arbore este format din noduri legate intre ele prin relatii “parinte-copil”
- Aplicatii:
  - Scheme organizatorice
  - Sisteme de fisiere
  - Medii de programare
- Toate nodurile (cu exceptia unuia singur) are un parinte unic.



Exemplificare

- Radacina: nod unic fara parinte
- Nod intern: nod cu cel putin un copil (A, B, C, F)
- Nod extern: nod fara copil (E, I, J, K, G, H, D)
- Ascendentii unui nod: parinte, bunic, strabunic, …
- Descendentii unui nod: copil, nepot, etc.
- Subarbore: un arbore format dintr’un nod oarecare si descendentii sai



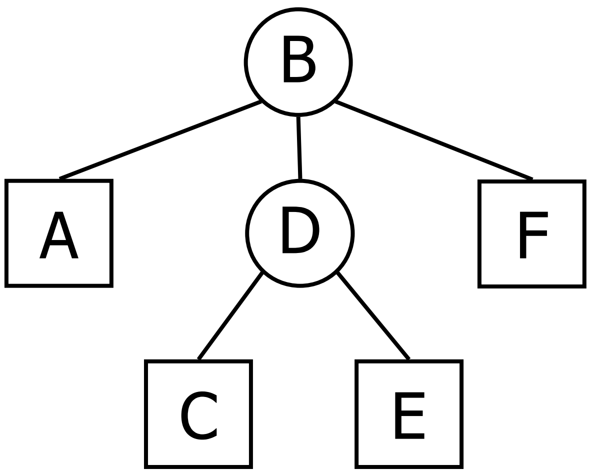
Recursivitatea arborilor

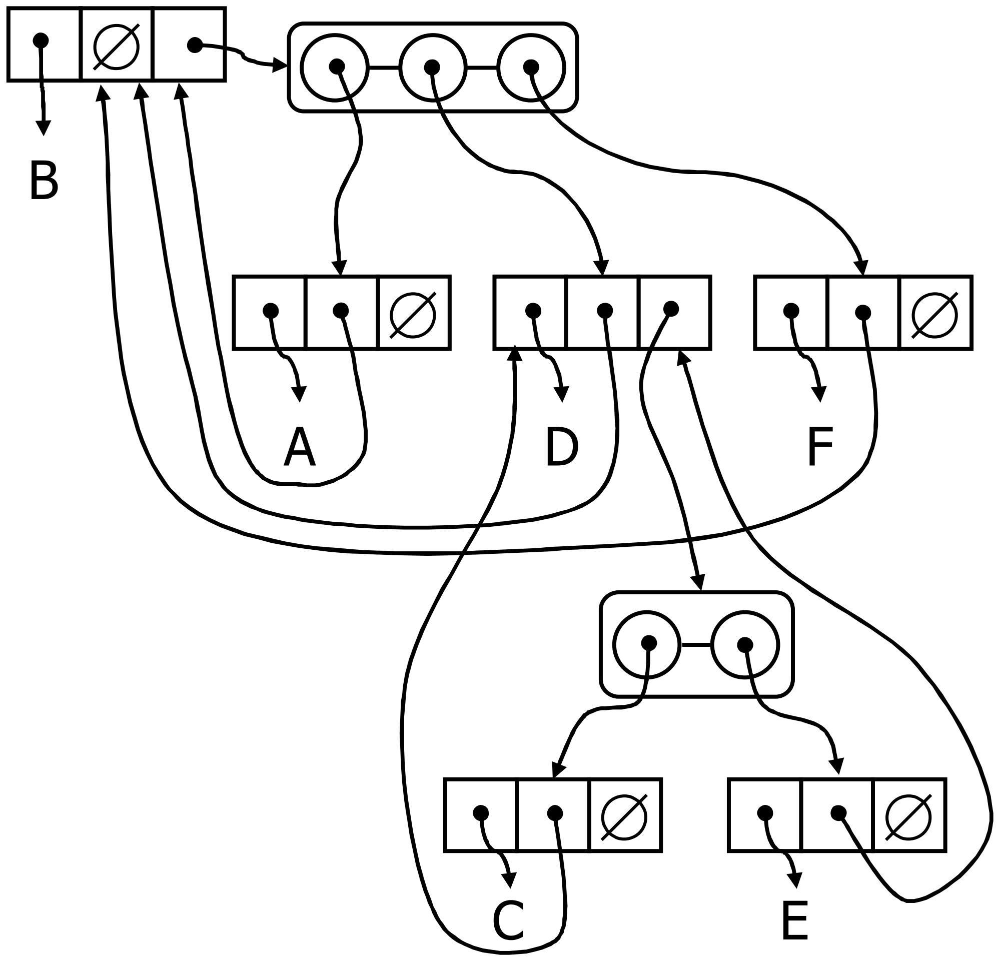
Un arbore poate fi definit recursiv in felul urmator::

1. O structura de date fara elemente (goala) reprezinta un arbore gol.
2. Daca *t*1, *t*2, …, *tk* sunt arbori disjuncti, atunci structura a carei radacina are ca si copii radacinile lui *t*1, *t*2, …, *tk* este de asemenea un arbore
3. Arborii pot fi generati numai prin regulile 1 si 2.

Arbori: Structuri inlantuite

- Un nod este reprezentat de un obiect ce memoreaza:
  - Element
  - Nod parinte
  - Secventa de noduri copil





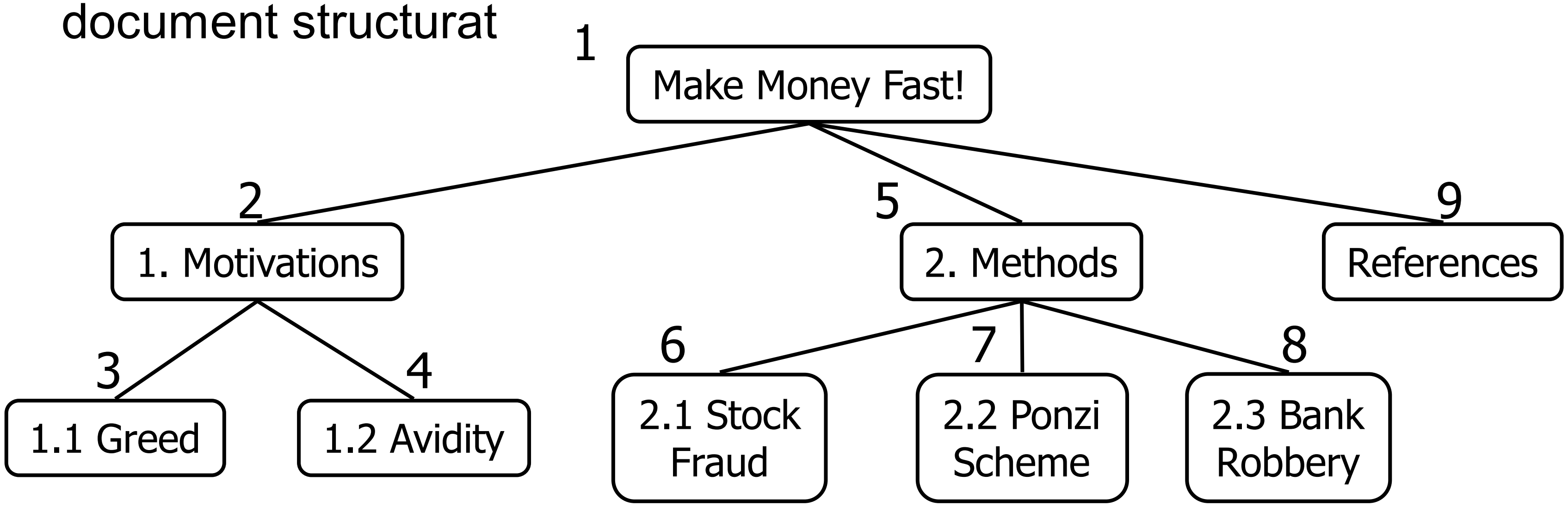
Structuri de date de tip arbore

- Nodurile se abstractizeaza prin pozitionare
- Metode generice:
  - integer size()
  - boolean isEmpty()
  - Iterator elements()
  - Iterator positions()
- Metode *accessor*:
  - position root()
  - position parent(p)
  - positionIterator children(p)
- Metode de interogare:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Metoda de actualizare (*update*):
  - object replace (p, o)
- Alte metode pot fi definite in cadrul structurii de date ce implementeaza arborele

Preorder Traversal (PT)

- O metoda transversala permite vizitarea nodurilor unui arbore intr'o maniera sistematica
- Prin PT, un nod este vizitat inaintea descendentilor sai.
- Aplicatie: tiparirea unui document structurat

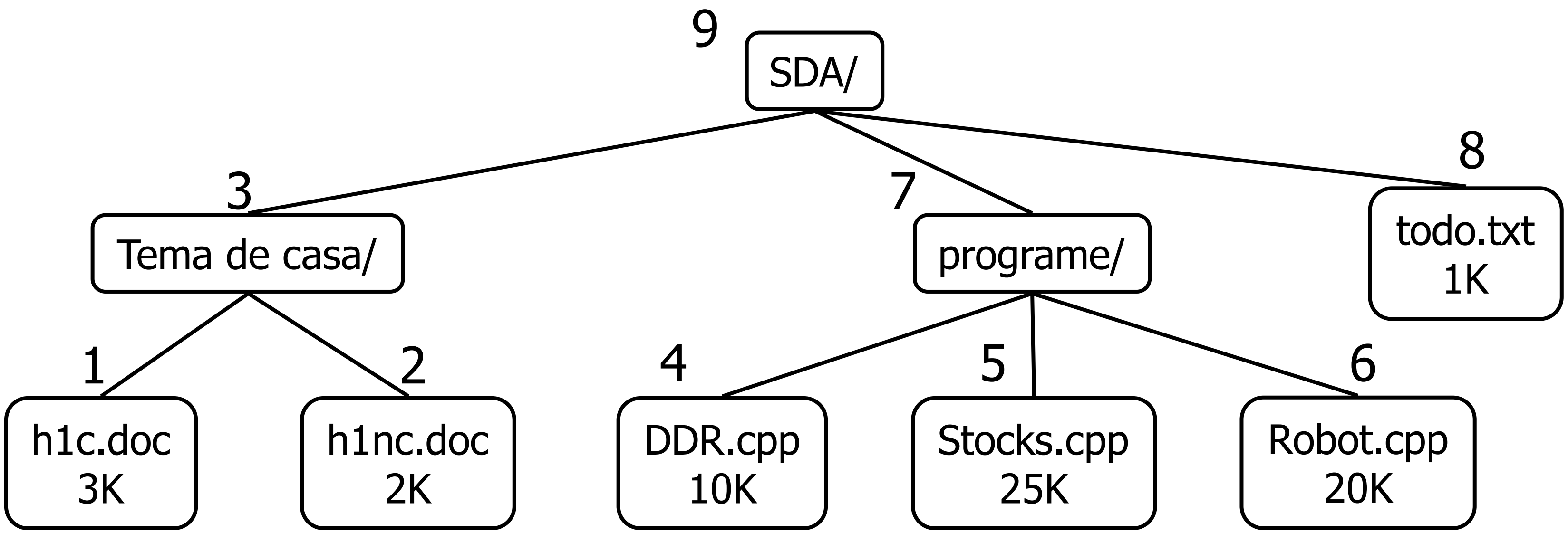
Algorithm ***preOrder***(***v***)
  visit(***v***)
  for each ***child w of v***
    preorder (***w***)



Postorder Traversal (PoT)

- In PoT, un nod este vizitat dupa descendentii sai
- Aplicatie: sa se calculeze spatiul ocupat de fisiere intr'un director / subdirectoarele sale.

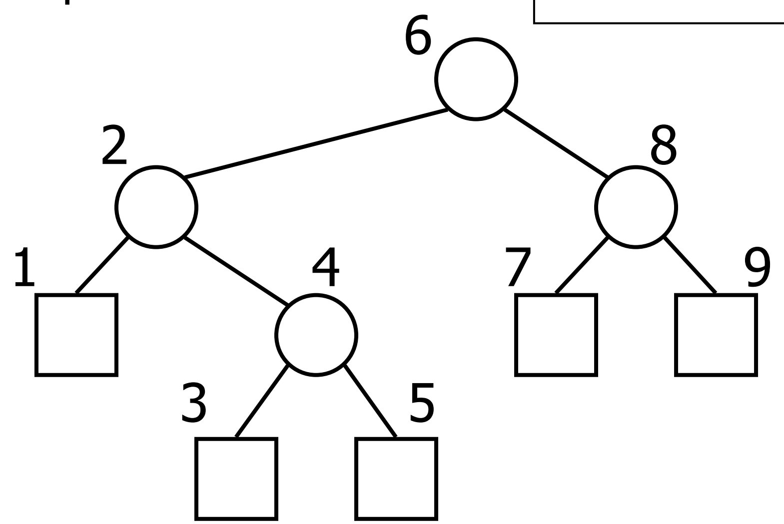
Algorithm ***postOrder*****(*****v*****)**
  for each ***child w of v***
    postOrder **(*****w*****)**
  visit**(*****v*****)**



Inorder Traversal (IT)

- Un nod este vizitat dupa vizitarea subarborele stang si inainte de vizitarea subarborelui drept.
- Aplicatie: sa se deseneze un AB
  - x(v) = inorder rank of v
  - y(v) = depth of v

Algorithm ***inOrder*(*v*)**
  if ***hasLeft*** **(*v*)**
    inOrder **(*left*** **(*v*))**
  ***visit*(*v*)**
  if ***hasRight*** **(*v*)**
    inOrder **(*right*** **(*v*))**



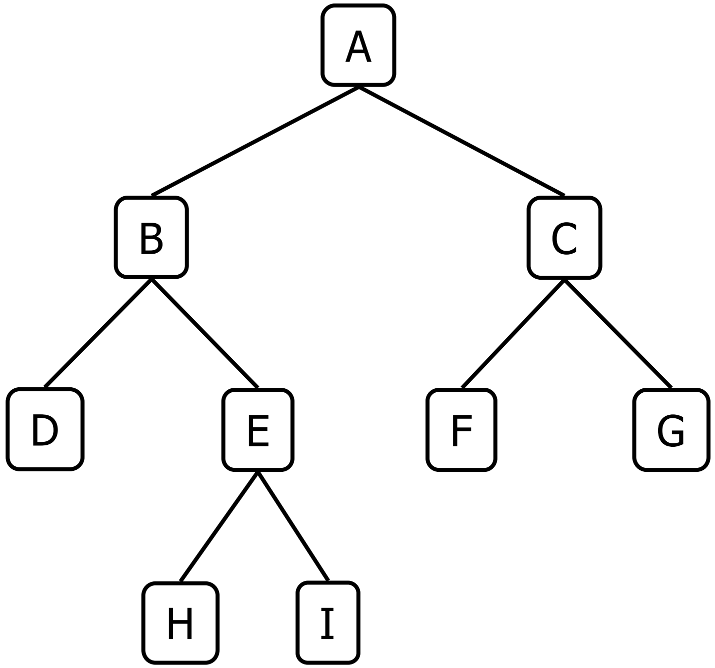
Arbori binari (1)

- Un arbore binar (AB) este un arbore cu urmatoarele proprietati:
  - Orice nod interior are maximum 2 descendenti
  - Descendentii unui nod formeaza o pereche ordonata

- Descendentii unui nod interior: copil stanga / dreapta

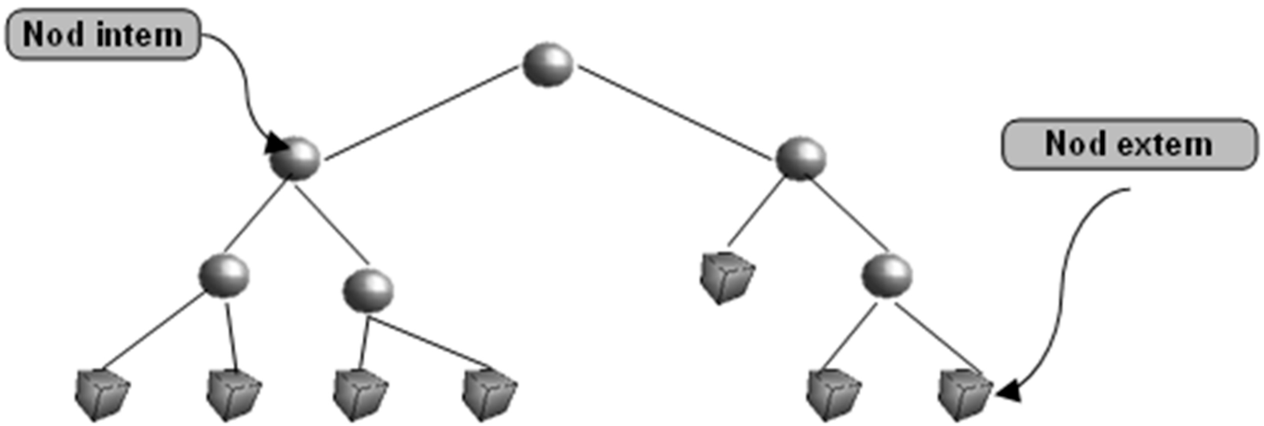
- Definitii recursive ale AB:
  - Un arbore cu un singur nod, sau
  - Un arbore a carui radacina are o pereche ordonata de descendenti, fiecare dintre ei fiind la randul lui un AB.

- Aplicatii:
  - Expresii aritmetice
  - Procese de decizie
  - cautare



Arbori binari (2)

- Deoarece in cazul AB toate nodurile trebuie sa aiba un acelasi numar de descendenti, au fost introduse urmatoarele denumiri:
  - Nod intern: are doi descendenti
  - Nod extern (frunza): nu are nici un descendent



Arbori binari (3)

Intr-un AB, daca toate nodurile la toate nivelurile (cu exceptia ultimului) au doi descendenti, atunci vor exista 2^0 noduri la nivel 1 (radacina), 2^1 noduri la nivel 2, … 2^i noduri la nivel i+1.
In acest caz este vorba de un **AB complet (ABC)**.

In ABC, toate nodurile non-terminale vor avea descendenti si toate frunzele (corespunzatoare) vor fi la un acelasi nivel..

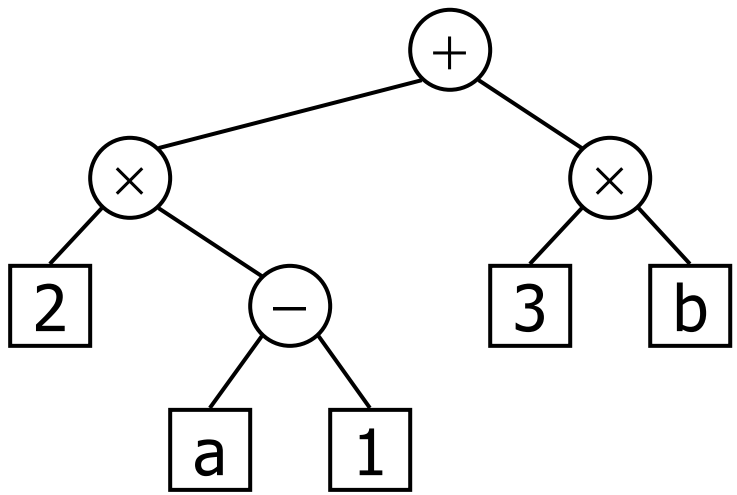
Numarul total de noduri in ABC cu k nivele va fi:

$$\underbrace{1+2+\cdots+2^{k-2}}_{\text{noduri non-terminale}} + \underbrace{2^{k-1}}_{\text{noduri terminale (frunze)}} = 2^k - 1$$

Expresii aritmetice

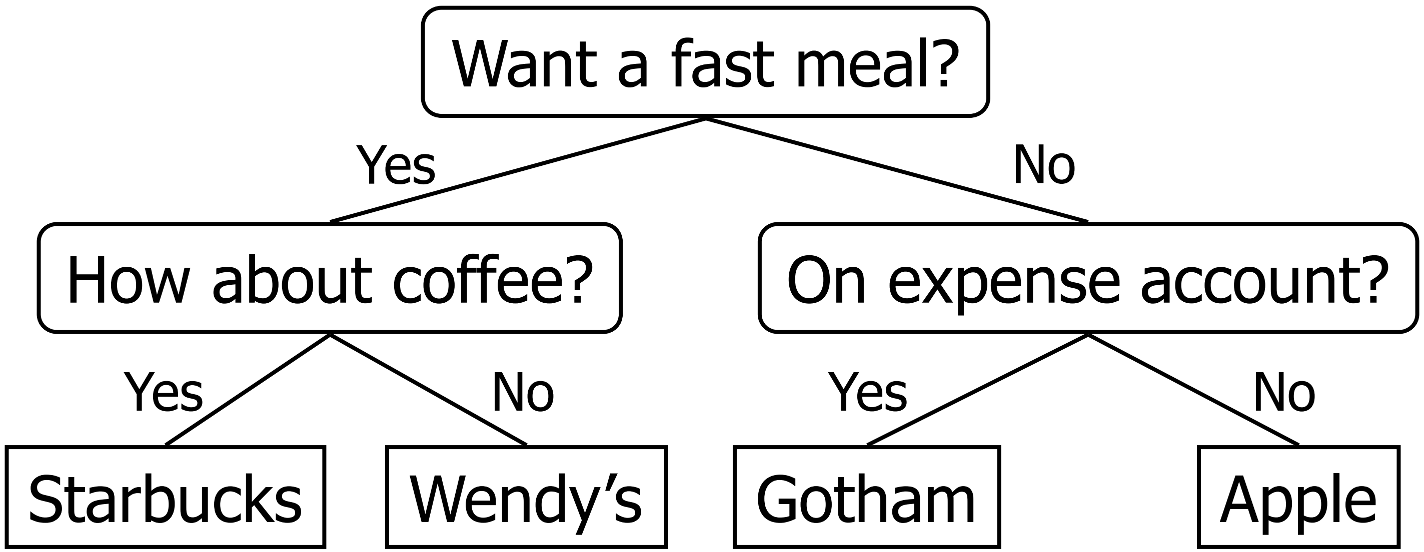
- AB asociat cu o expresie aritmetica
  - Nod intern: operatori
  - Nod extern: operanzi
- Exemplu: fie urmatoarea expresie aritmetica implementabila cu ajutorul unui AB:

$(2 \times (a - 1) + (3 \times b))$



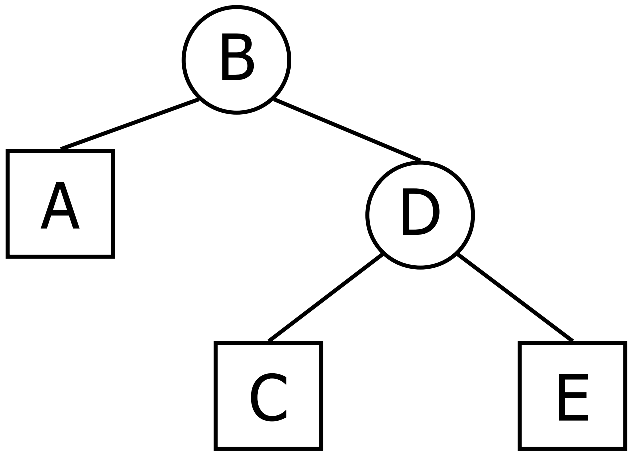
Arbore de decizie

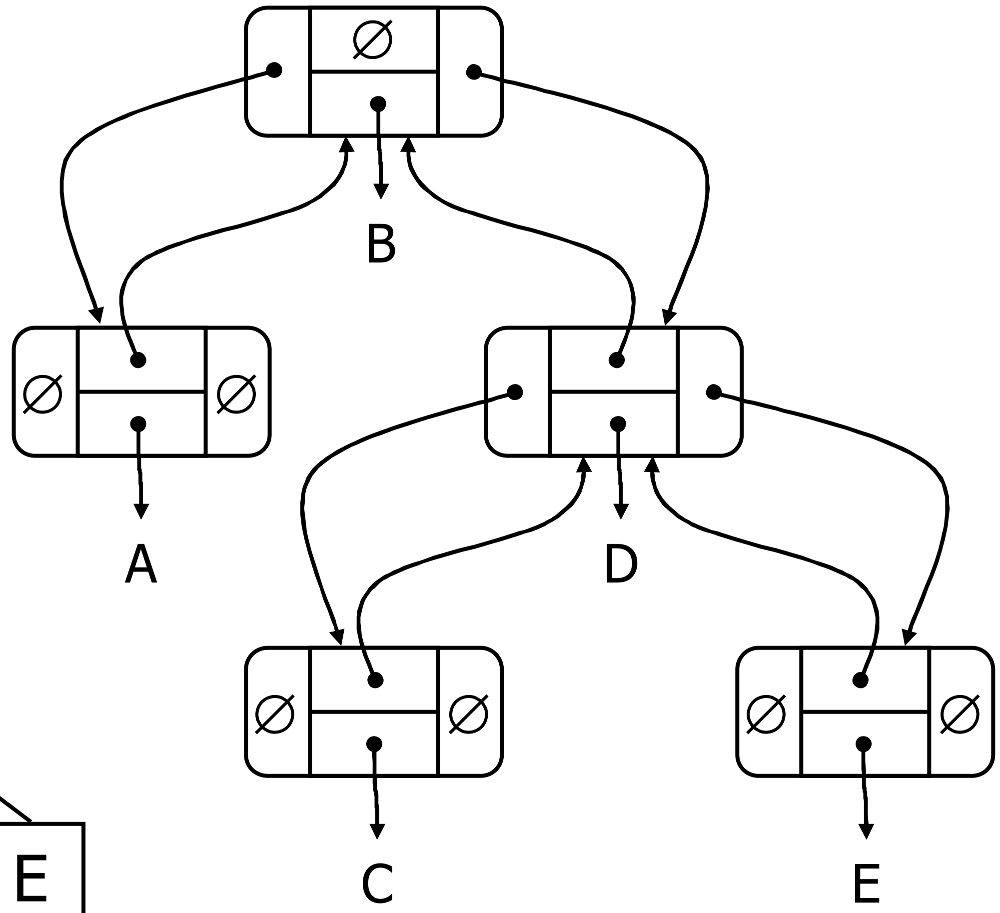
- AB asociat cu un proces de decizie:
  - Nod intern: intrebari cu respunsuri DA / NU
  - Nod extern: decizie
- Exemplu: micul dejun



Structuri inlantuite pentru AB

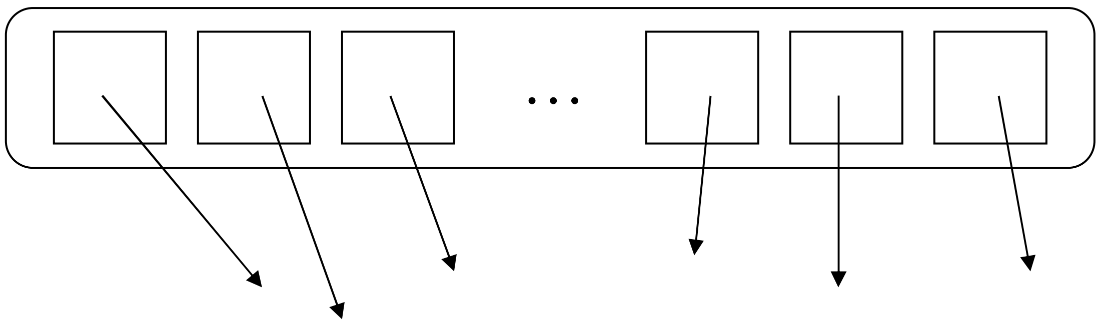
- Un nod este reprezentat de un obiect ce memoreaza:
  - Element
  - Nod parinte
  - Nod copil stanga
  - Nod copil dreapta

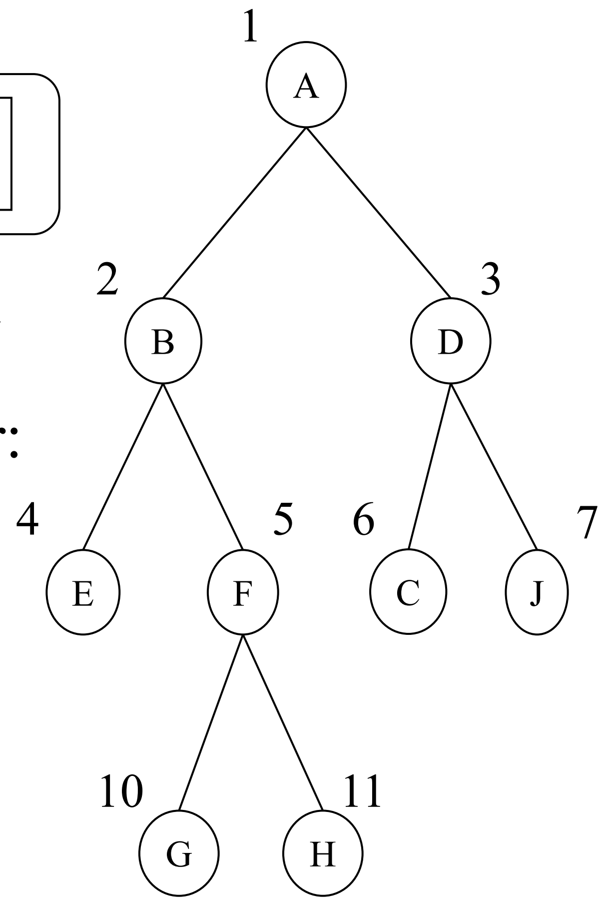




AB: reprezentare matriceala

- Nodurile sunt stocate intr'un vector



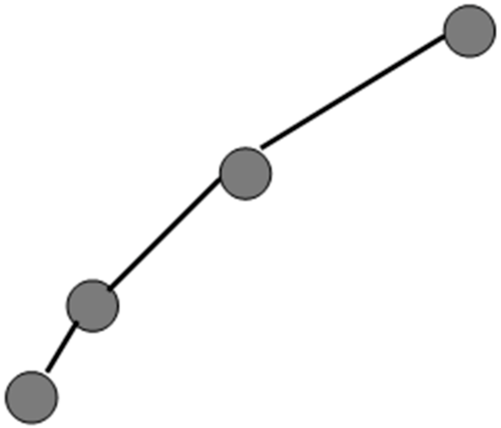


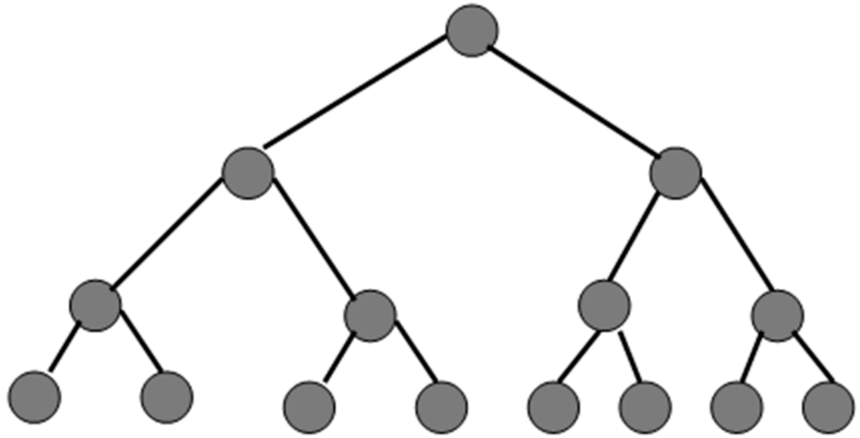
- fie rank(node) definit in felul urmator:
    i. rank(root) = 1
    ii. if node is the left child of parent(node), rank(node) = 2*rank(parent(node))
    iii. if node is the right child of parent(node), rank(node) = 2*rank(parent(node))+1

Numarul de noduri

- Intr-un AB cu inaltime h trebuie sa existe *minimum* cate un nod pe fiecare nivel

→ in acest caz numarul minim de noduri va fi h + 1





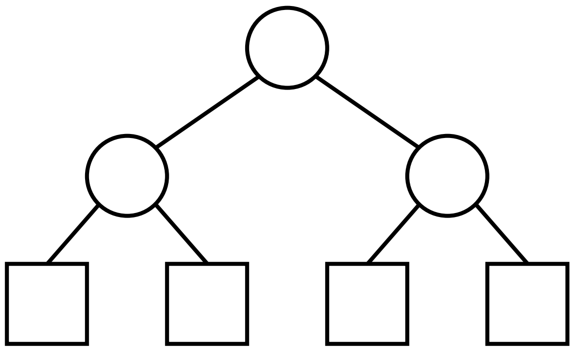
- Intr-un AB cu inaltime h numarul maxim de noduri va fi:

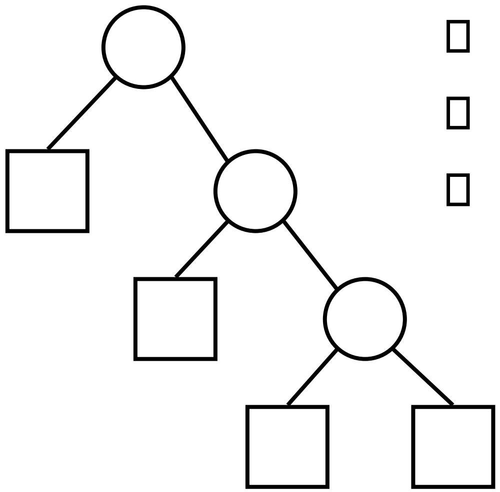
= 1 + 2 + 4 + 8 + … + 2h = 2h+1 - 1

Proprietatile AB

- Notatii
  - n numar noduri
  - e numar noduri externe
  - i numar noduri interne
  - h inaltime

- Proprietati:
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$





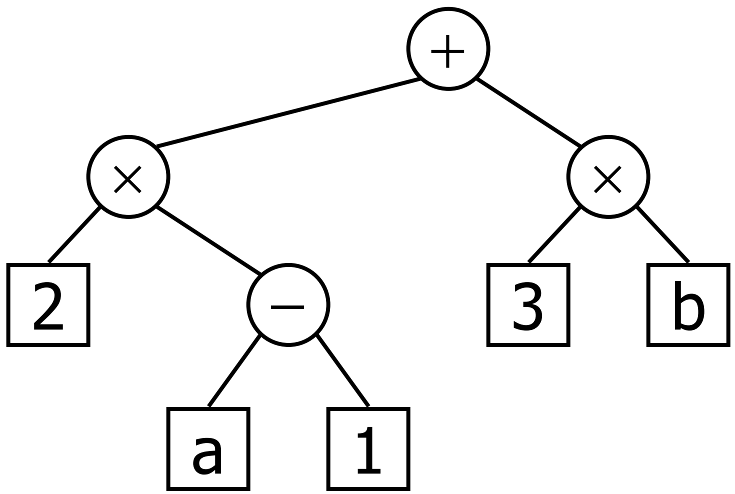
Structuri de date de tip AB

- Un AB extinde o structura de tip arbore, i.e., mosteneste toate metodele aferente unui arbore
- Metode aditionale:
  - position left(p)
  - position right(p)
  - boolean hasLeft(p)
  - boolean hasRight(p)
- Metodele aditionale pot fi definite in cadrul structurilor de date ce implementeaza AB

Tiparirea expresiilor aritmetice

Specializarea unui *inorder traversal*

- Print operand / operator cand nodul este vizitat
- print “(“ inainte de traversare subarbore stanga
- print “)“ inainte de traversare subarbore dreapta

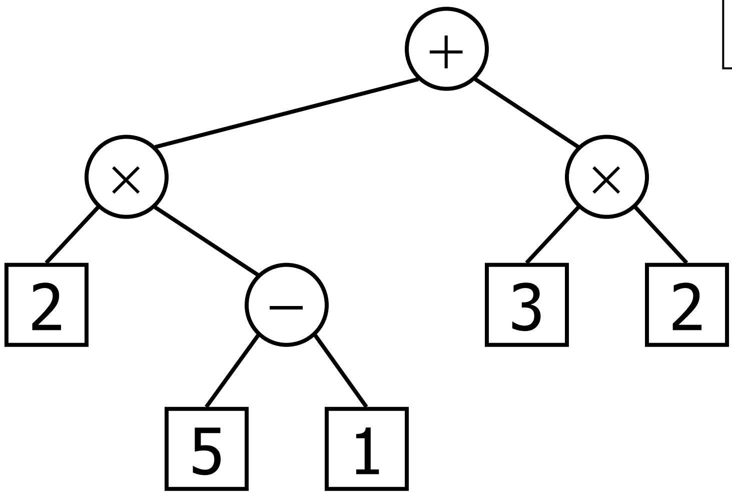


Algorithm ***printExpression***(***v***)
    if ***hasLeft*** (***v***)
        print(“(”)
        inOrder (***left***(***v***))
    print(***v.element*** ())
    if ***hasRight*** (***v***)
        inOrder (***right***(***v***))
        print (“)”)

$$((2 \times (a - 1)) + (3 \times b))$$

Evaluare expresii aritmetice

- Specializare unui *postorder traversal*
  - Metoda recursiva ce intoarce valoarea unui subarbore
  - Cand se viziteaza un nod intern, sunt combinate valorile din subarbore



Algorithm ***evalExpr*(*v*)**
    if ***isExternal*** **(*v*)**
        return ***v.element*** **()**
    else
        x ← ***evalExpr*(*leftChild*** **(*v*))**
        y ← ***evalExpr*(*rightChild*** **(*v*))**
        ◊ ← operator stored at ***v***
        return ***x*** **◊** ***y***

BST (Binary Search Tree)

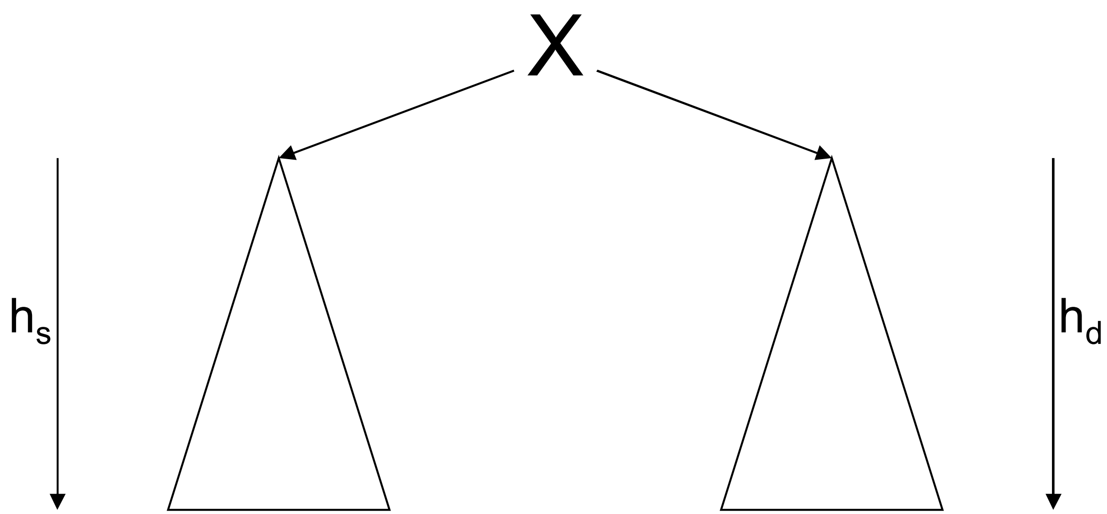
- **BST**: valoarea dintr’un nod este >= valoarea din copilul stang si <= valoarea din copilul drept

- Operatii in BST:
  - **Insert**: intodeauna o frunza
  - **Delete**:
    - Nod singur
    - Nod cu un copil
    - Nod cu doi copii: se inlocuieste nodul cu elemental minim din subarborele drept / elemental maxim din subarborele stang
  - **Find element** (min / max)

Arbori AVL (1)

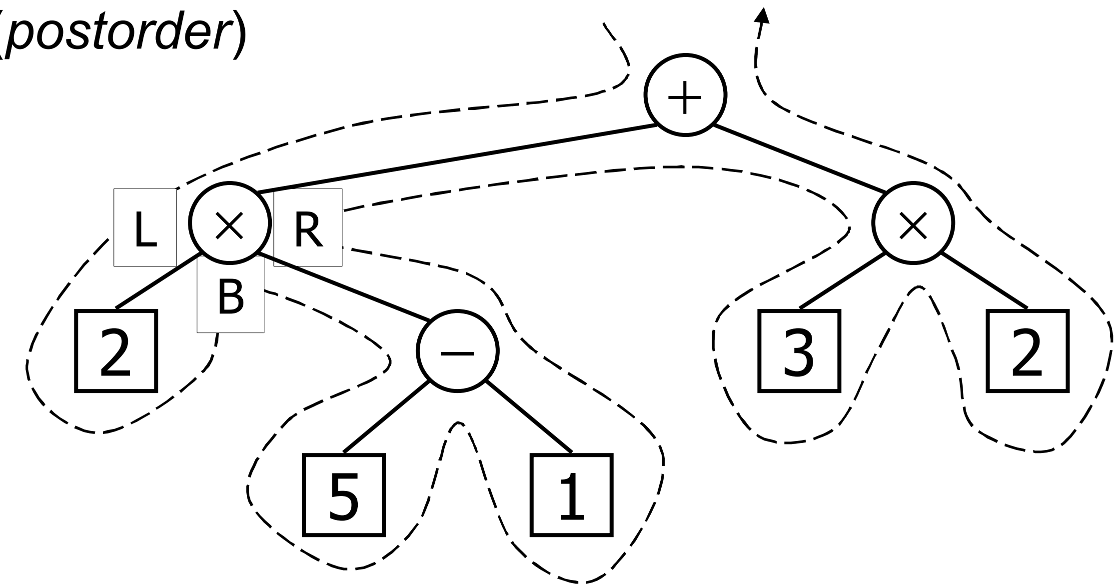
- **Arborii AVL** sunt arbori binari ordonati, care au in plus o proprietate de echilibru stabilita de **A**delson, **V**elski si **L**andis, de unde si denumirea de arbori **AVL**
- Proprietatea de echilibru e valabila **pentru orice nod** al arborelui si spune ca: “inaltimea subarborelui stang al nodului difera de inaltimea subarborelui drept al nodului prin cel mult o unitate”
- Cu alte cuvinte, pentru orice nod, cei doi subarbori au inaltimile sau egale, sau, daca nu, ele difera prin maxim o unitate in favoarea unuia sau altuia dintre subarbori

- $| h_s – h_d | \leq 1$, oricare ar fi nodul X apartinand arborelui



Euler Tour Traversal

- Metoda generica de traversare a unui AB
- Cazuri particulare: *preorder, postorder* si *inorder traversals*
- Parcurgerea arborelui se face vizitand fiecare nod de trei ori:
  - La stanga (*preorder*)
  - Din jos (*inorder*)
  - La dreapta (*postorder*)



Cautarea in AB (1)

- Complexitatea unei cautari in **AB** este masurata in **numarul de comparatii** facute in cursul procesului de cautare. Acest numar depinde de numarul de nosuri intalnite pe unicul drum ce uneste radacina de nodul destinatie (cautat).

- Prin urmare, complexitatea este data de lungimea drumului + 1.

- Ea depinde de **forma arborelui** si de **pozitia nodului destinatie in cadrul arborelui**.

Cautarea in AB (2)

Lungimea drumului intern (*internal path length,* IPL) este data de suma lungimilor drumurilor la toate drumurile, i.e. $\sum(i\text{-}1)l_i$ pentru toate nivelele i, unde l_i reprezinta numarul de noduri de la nivelul i.

Lungimea medie a drumului (intern) = IPL/n

In cazul cel mai nefavorabil, in care arborele devine o lista inlantuita:

$$path_{worst} = \frac{1}{n}\sum_{i=1}^{n}(i-1) = \frac{n-1}{2} = O(n)$$

Cautarea in ABC (1)

IPL pentru cazul cel mai bun < IPL al unui ABC cu o aceeasi lungime

Prin urmare, se aproximeaza *average path length* pentru cazul cel mai bun prin *average path length* al unui ABC cu o aceeasi inaltime, *h*.

Pentru ABC cu inaltimea h, IPL=

$$\sum_{i=1}^{h-1}(i \times 2^i) = (h-2)\times 2^h + 2$$

Cautarea in ABC (2)

Numarul total de noduri in ABC cu inaltimea este n=2^h-1.
Prin urmare,

$$path_{best} \leq \frac{IPL_{complete}}{n} = \frac{(h-2)2^h + 2}{2^h - 1} \approx h - 2 = \log_2(n+1) - 2$$

$O(path_{best}) = O(\log n)$

Cazul mediu se gaseste undeva intre (n-1)/2 si log(n+1)-2.

Intrebare: in cautarea unui nod (aflat intr'o pozitie medie) intr'un arbore, rezultatul va fi mai aproape de **O(*n*)** sau de **O(log*n*)** ?