SD – Seminar 11
05.01.2021

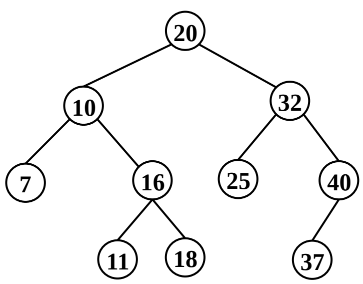
ARBORI BINARI DE CAUTARE

Implementarea cu structuri inlantuite

```
struct nod {
        int inf
        nod * stg
        nod * drp
}
ArbBinCautare alias pentru nod *.
```

Pr. 1 Sa se scrie un subprogram care afiseaza valorile k dintr-un arbore binar de cautare cu proprietatea: k1 <= k <= k2, unde k1 si k2 sunt parametri de intrare ai subprogramului. Care este complexitatea subprogramului propus?

<u>*Exemplu:*</u>



k1 = 17 si k2 = 35 → 18, 20, 25, 32

k1 = 1 si k2 = 100 → 7, 10, …, 40 (toate vf.)

k1 = 19 si k2 = 21 → 20

k1 = 50 si k2 = 100 → nimic

I Parcurgere + si verificarea conditiei → O(n), n = nr de vf din arbore; $\Omega(n)$ → $\Theta(n)$

II Apel conditionat :

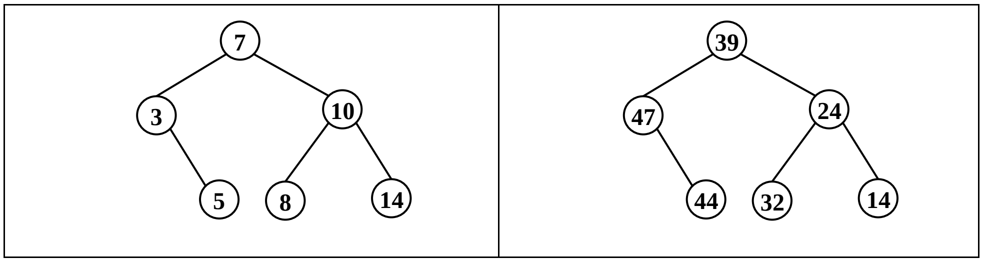
```
procedure subinterval( nod * t,  int k1, int k2 )
begin
        if t != NULL then { \\ parcurgere preordine
                if ( k1 <= t->inf  and t->inf <= k2) then print t->inf
                if (t->inf >= k1) then subinterval (t->stg, k1, k2)
                if (t->inf <= k2) then subinterval (t->drp, k1, k2)
        }

end
```

$\Omega(h)$, O(n)

? putem exprima complexitatea procedurii si in functie de k1 si k2 ?

Pr. 2 Scrieti o procedura care aduna la fiecare nod dintr-un arbore binar de cautare valorile mai mari decat valoarea curenta.




```
suma <- 0
transfromaArbore(t)

procedure transformaArbore( nod *t ) \\  complexitate:  O(n),  Ω(n)
begin
        if t != NULL then {
                transformaArbore(t->drp)
                suma  <- t->inf + suma;   t->inf <- suma
                transformaArbore(t->stg)
        }
end
```

```
  2                              1
1   3                              2
                                     3
```

h= O(log n)   h= O(n)

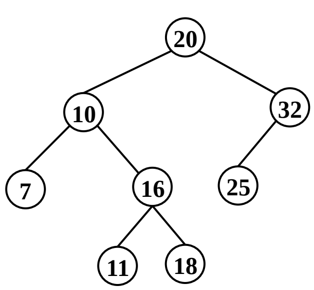
Arbori binari de cautare echilibrati: arbori AVL

Def. Un arbore de cautare este AVL echilibrat daca pentru orive varf v, avem:

$$|h(v\text{->}stg) - h(v\text{->}drp)| <= 1$$

Def. Factorul de echilibrare al unui nod v: h(v->stg) – h(v->drp)

Lema. Daca t este abrore binar de cautare AVL-echilibrat cu n noduri interne, atunci $h(t) = \Theta(\log n)$.

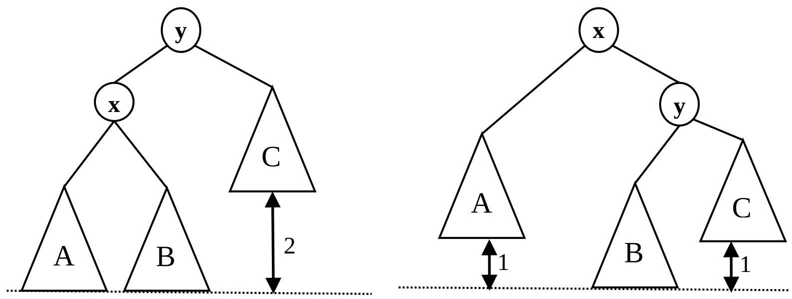


Echilibrarea se face prin rotatii:

SIMPLE:
- la dreapta
- la stanga

DUBLE:
- la dreapta
- la stanga

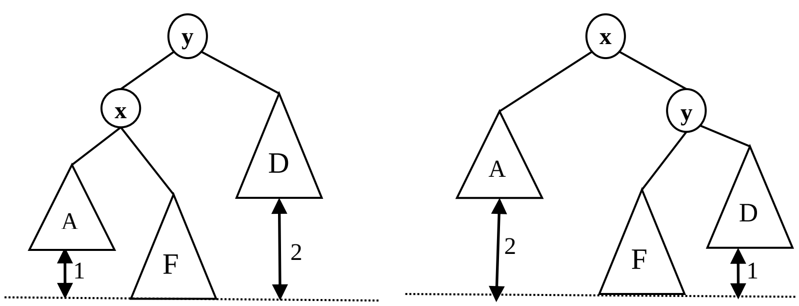
Rotatie simpla la dreapta



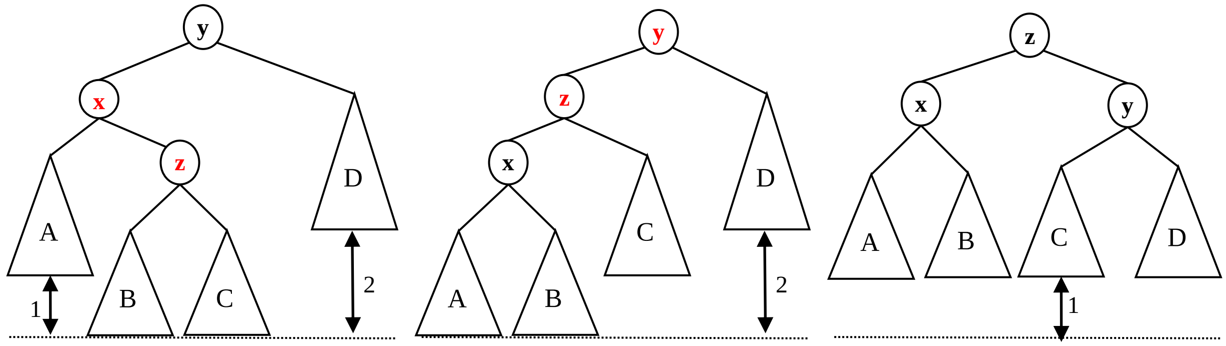

```
function rotatieSimplaDreapta( nod* t) \\ t este adresa nodului y . Complexitate:  O(1)
begin
    aux <- t->stg
    t->stg <- aux->drp
    aux->drp  <- t
    return aux
end
```

Rotatie dubla la dreapta

(Cand rotatiile simple nu sunt suficiente)



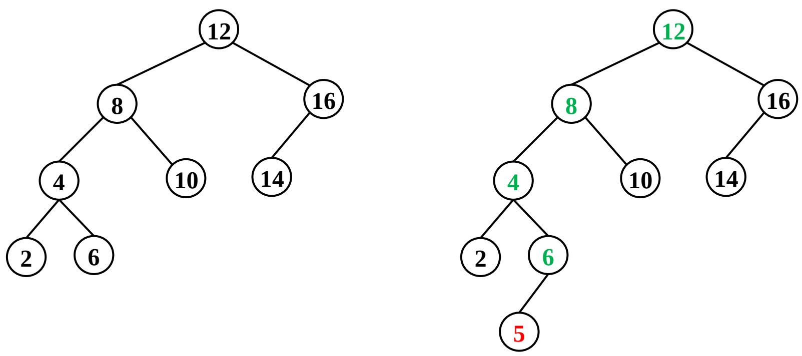
- rotatie simpla stanga cu x-y, urmata de o rotatie simpla dreapta cu z-y



function rotatieDublaDreapta(nod * t) \\ t este adresa radacinii y. Complexitate: O(1)
begin
  t->stg <- rotatieSimplaStanga(t->stg)
  return rotatieSimplaDreapta(t)
end

Exemplu de inserare intr-un arbore AVL echilibrat

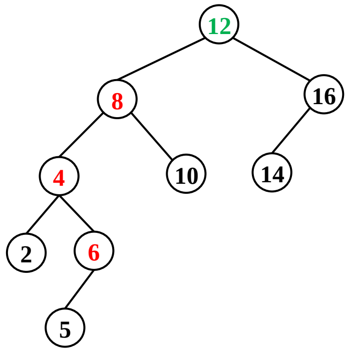
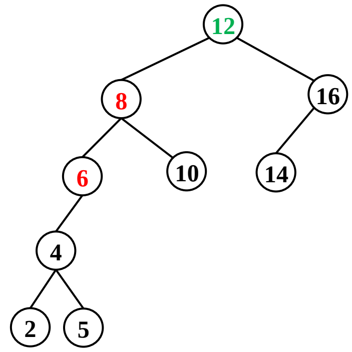
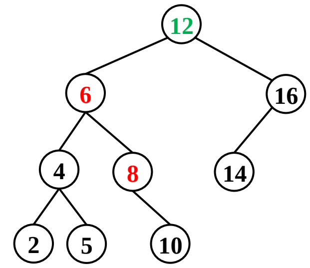
Inseram noua cheie **5** in urmatorul arbore binar de cautare AVL echilibrat.



Se memoreaza pe o stiva de noduri, drumul de la radacina la nodul inserat.
Stiva: [12, 8, 4, 6

Iterativ, se scoate un nod de pe stiva, se verifica factorul de echilibrare. In cazul in care apare un nod critic (factorul de echilibrare este 2 sau -2), atunci se aplica rotatiile corespunzatoare pentru a re-echilibra subarborele cu radacina in nodul critic.

fe(6) = 1
fe(4) = -1
fe(8) = 2 !!! – rotatie dubla la dreapta cu nodul 8

Stiva: [12
fe(12) = 1

Pr. 3 Scrieti o functie care verifica daca un arbore binar de cautare este AVL echilibrat.

function esteAVL(t)
begin

end

Pr. 4 Scrieti o functie care verifica daca exista o pereche de valori intr-un arbore binar de cautare echilibrat care sa aiba suma egala cu x. Complexitate timp a functiei trebuie sa fie O(n), unde n este numarul de noduri din arbore. Se poate folosi O(log n) spatiu de memorie suplimentar.