Structuri de Date

Anul universitar 2019-2020

Prof. Adina Magda Florea

Curs Nr. 7

Arbori de căutare echilibrați – Part I
(balanced search trees)

- **Arbori AVL**
- **Arbori roșu și negru**
- **Arbori 2-4**

Arbori binari de cautare echilibrati in inaltime

- **Înălțimea unui arbore cu radacina N**

  h(N) = numărul de legături de-a lungul celei mai lungi căi de la **N** la o frunză
- **Factorul de echilibru** (balanced factor BF)
- **BF**(N) = **h**(Rt(N))- **h**(Lt(N))
- **Arbore binar echilibrat in inaltime cu radacina N** daca:
  - **Rt(N) si Lt(N) sunt arbori binari echilibrati**
  - **BF(N) este in {-1,0,+1}**
- Definitia se poate extinde si la arbori multicai
- Arbori AVL sunt arbori echilibrati in inaltime

Arbori binari de cautare echilibrati in inaltime

- Deci intr-un arbore AVL BF este 1,0 sau -1 pt orice nod
- BF(N) < 0 – left-heavy
- BF(N)>0 – right-heavy
- BF(N)=0 – balanced

Self-balancing trees

- Height balaced trees (AVL, Red-Black)
- Weight balanced trees

1. Arbori binari de căutare AVL

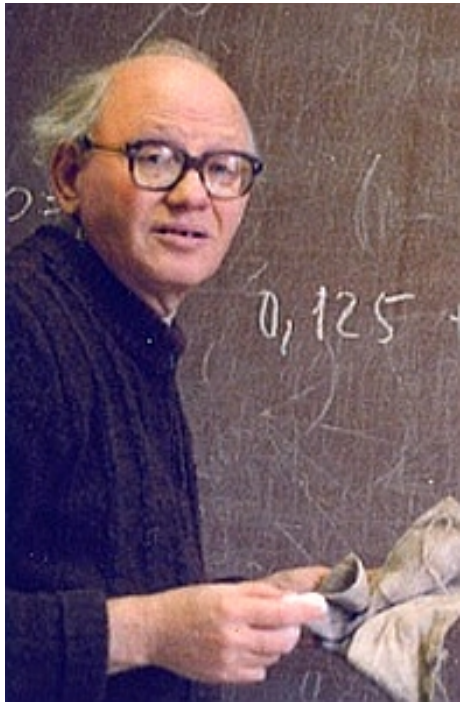
- Arbori cu **n chei** și **înălţimea O(log n)**

Arbori binari de căutare AVL

- Numiţi după inventatorii lor

Georgy Adelson-Velsky şi **Evgenii Landis**

1922-2014 1921-1997



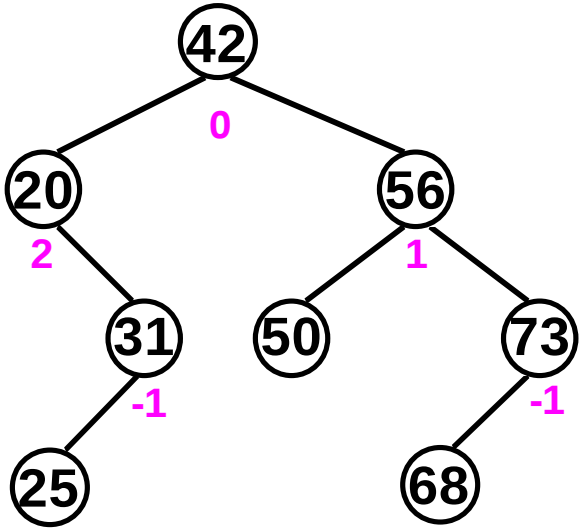
Arbori binari de căutare AVL

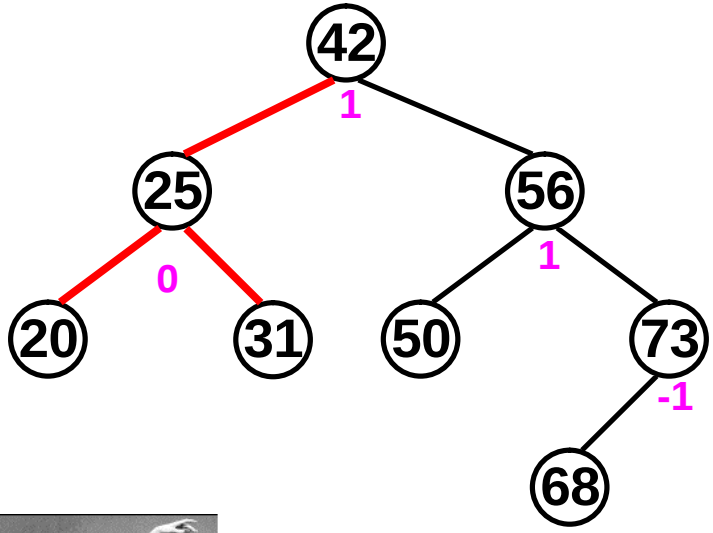
- **Înalțimea unui arbore AVL cu n noduri este O(log n)**
- Intr-un arbore AVL orice subarbore este AVL

- Căutarea, inserarea și eliminarea se realizează în timp O(log *n*)
- Căutarea se face la fel ca în orice arbore binar de cautare
- Inserarea și eliminarea pot distruge proprietatea de arbore AVL

Arbori binari de căutare AVL

BF(N)





Arbori binari de căutare AVL

- Inserarea sau eliminarea se face normal apoi se restaureaza conditia AVL prin re-echlibrare
- Dupa operatie se verifica BF pt nod parinte si stramosi.
- Printr-o inserarea BF poate ajunge max +2 sau min -2
- La fel printr-o eliminare
- Re-echilbrarea se face prin rotiri

Arbori binari de căutare AVL

- **4 cazuri de rotire posibile**, cu Z arborele asupra caruia s-a facut operatia, X parintele lui
- **Situatia Right-Right**
  - Z este Rt a lui X si Z nu este left-heavy

    (BF(N) ≥ 0)
  - **Rotire simpla Left** (stanga)
- **Situatia Left-Left**
  - Z este Lt a lui X si Z nu este right-heavy

    (BF(N) ≤ 0)
  - **Rotire simpla Right** (dreapta)

Arbori binari de căutare AVL

- **Situatia Right-Left**
  - Z este Rt a lui X si Z este left-heavy

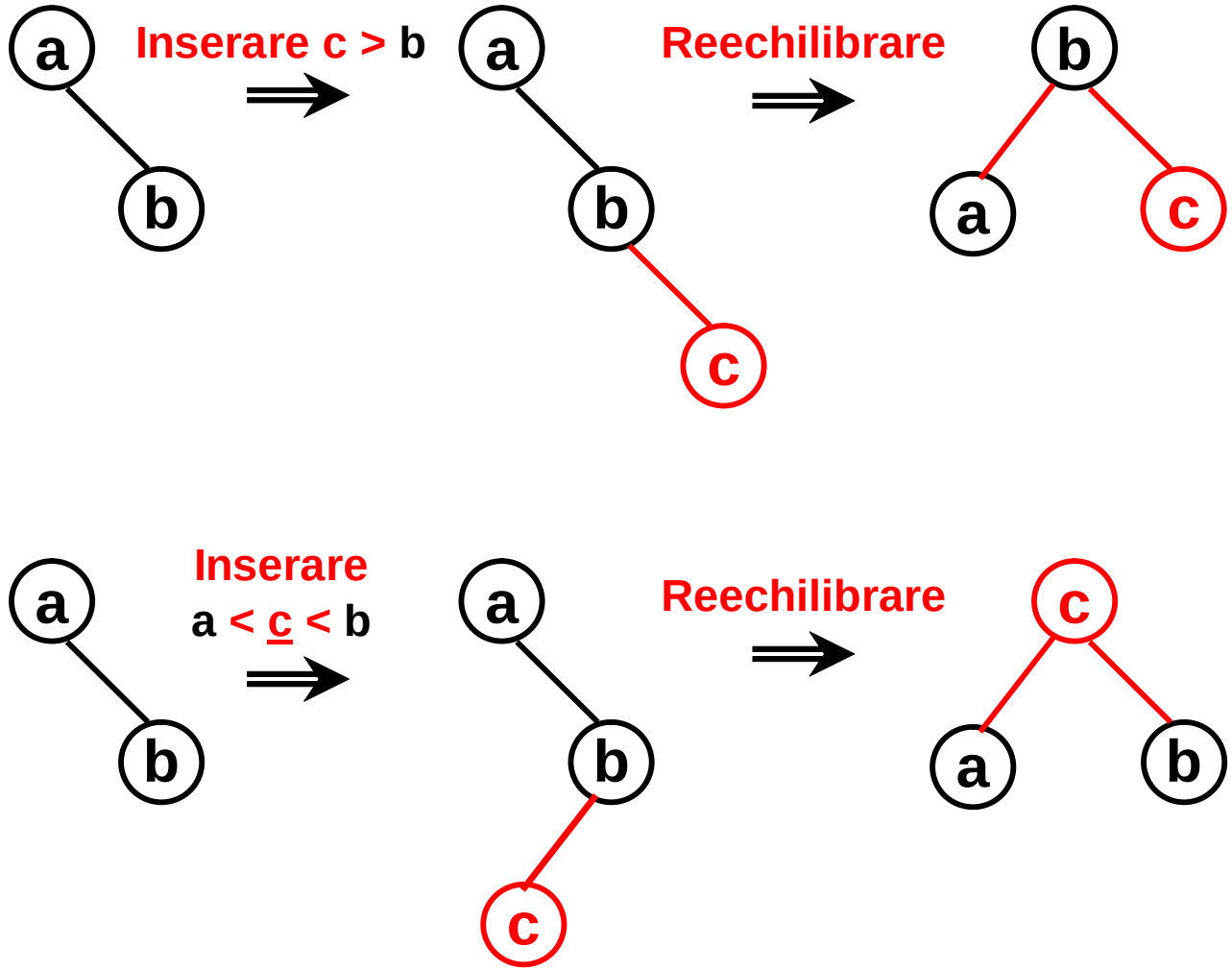
    (BF(N) = -1)
  - **Rotire dubla Right-Left** (dreapta-stanga) =

    Rotire Right dupa Z + rotire Left dupa X
- **Situatia Left-Right**
  - Z este Lt a lui X si Z este right-heavy

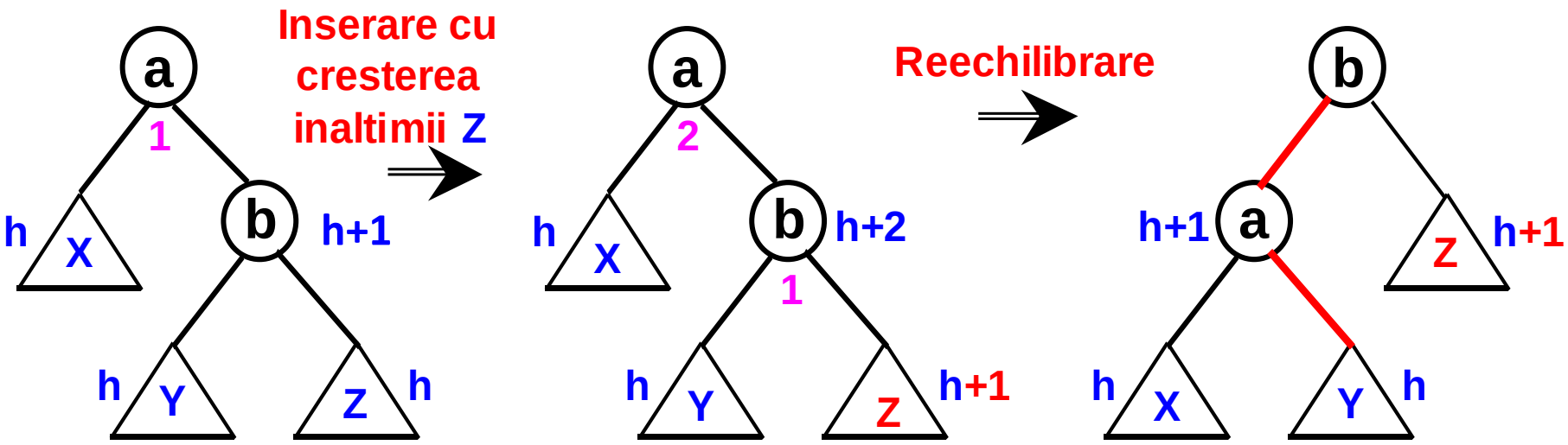
    (BF(N) = 1)
  - **Rotire dubla Left-Right** (stanga-dreapta) =

    Rotire Left dupa Z + rotire Right dupa X

Arbori binari de căutare AVL - inserare

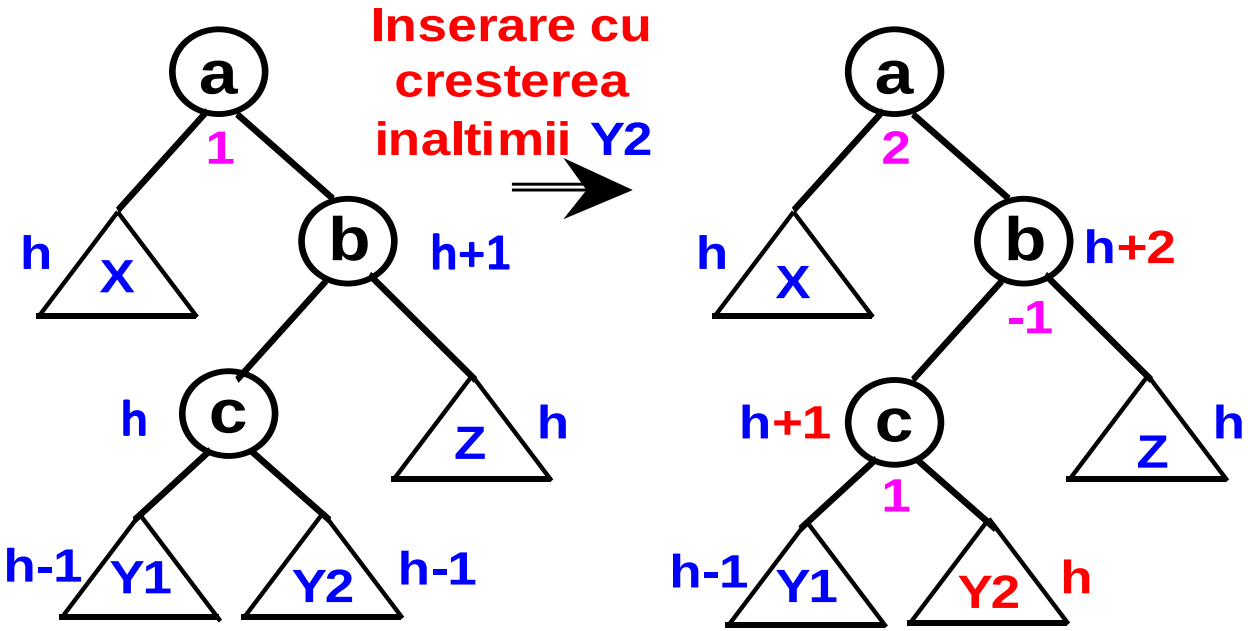


Arbori binari de căutare AVL - inserare

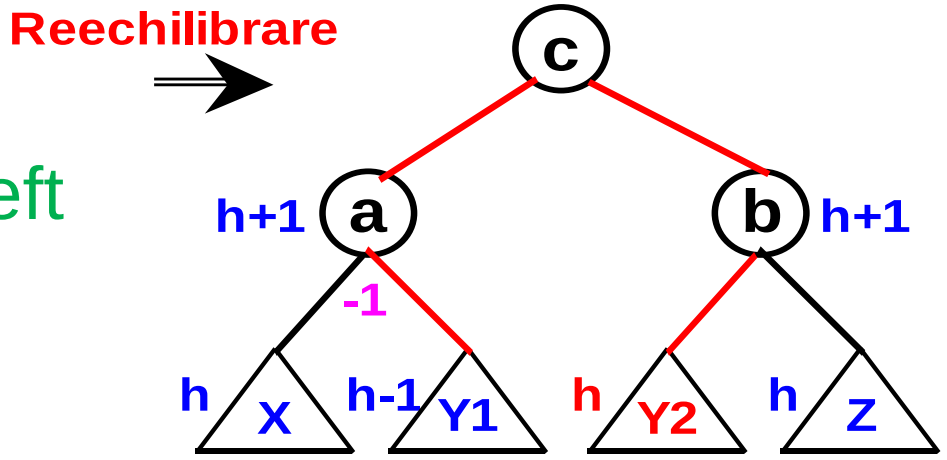


Rotire simplă Left

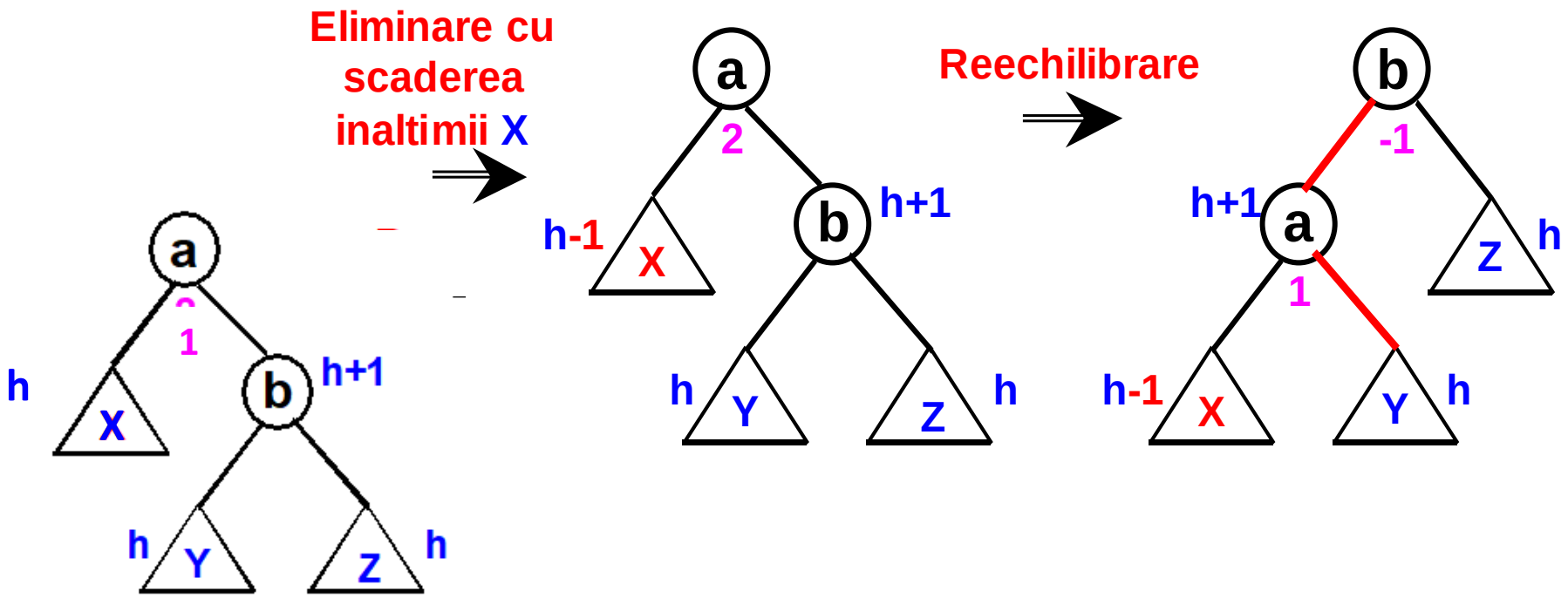
Arbori binari de căutare AVL - inserare



Rotire dublă Right-Left

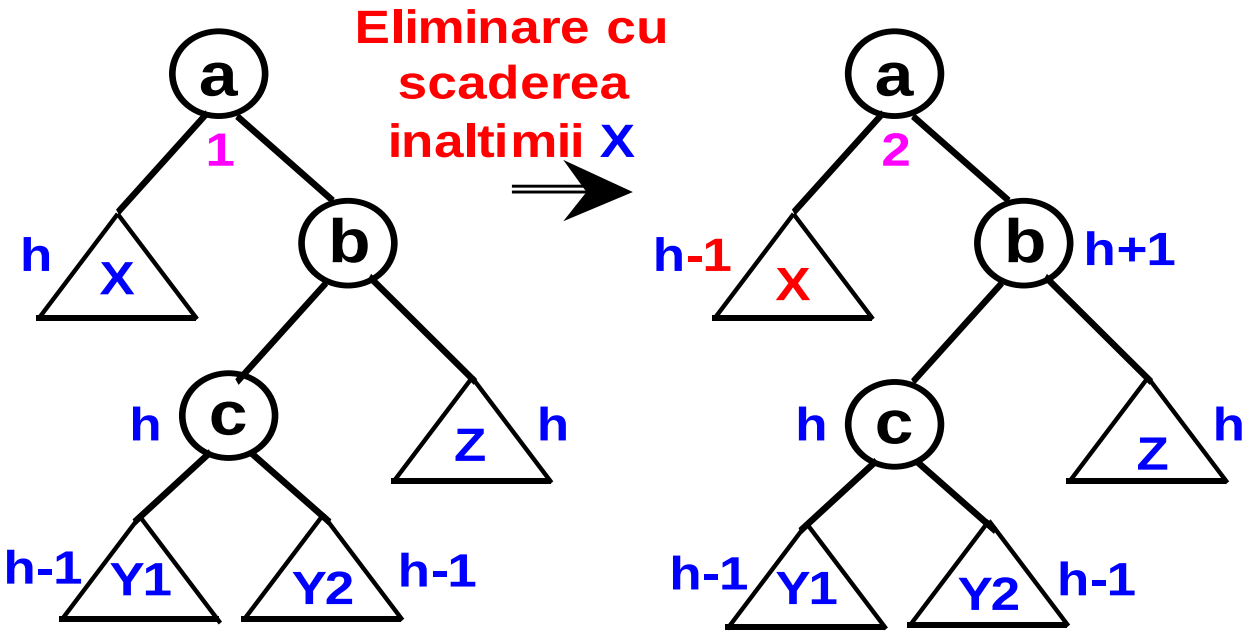


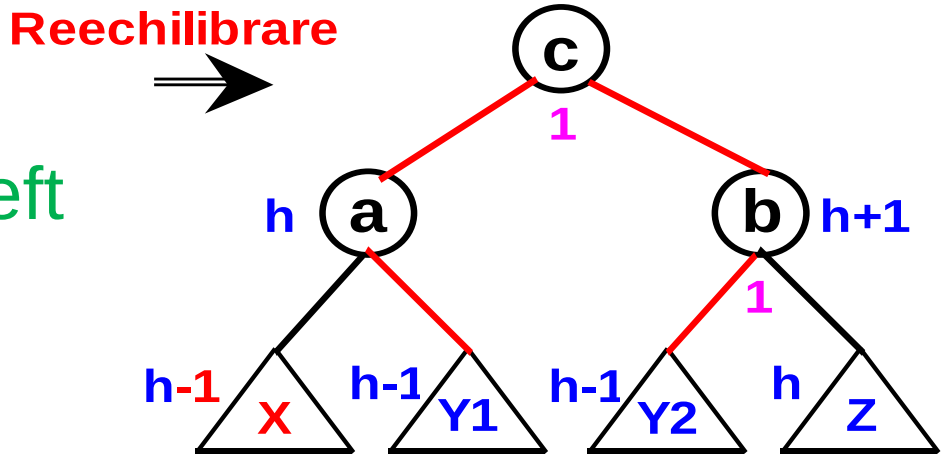
Arbori binari de căutare AVL - eliminare



Rotire simplă Left

Arbori binari de căutare AVL - eliminare





Rotire dublă Right-Left

Reprezentare arbori AVL

```
typedef struct node *AVLink;

typedef struct node {
          Item    elem;
          int     h;
          AVLink  lt, rt; } AVLNode;

/* optional se poate si
     AVLink  parent;  */
```

Inserare AVL

```
AVLink insert( Item x, AVLink t )
{ return insert1( x, t, NULL ); }

AVLink insert1( Item x, AVLink t, AVLink
  parent )
{  AVLink rotated_t;
    if( t == NULL )
    { t = (AVLink) malloc(sizeof(AVLNode));
      t->elem = x;  t->h = 0;
      t->lt = t->rt = NULL;
      return t;
     }
```

```
if (x < t->elem)
{ t->lt = insert1(x,t->lt,t);
   if(height(t->lt)–height(t->rt))== 2)
   {  if (x < t->lt->elem)
         rotated_t = s_rotate_right(t);
       else
         rotated_t = d_rotate_right(t);
       if (parent->lt == t)
         parent->lt = rotated_t;
       else
         parent->rt = rotated_t;
     }
 else
     t->h = max(height(t->lt), height(t->rt));
 }
else
  /* caz simetric pentru subarbore drept */
return t;
}
```

Rotire simplă dreapta

```
/* functia realizeaza rotirea intre X si copilul
   stang
   poate fi apelata numai daca X are un copil
   stang
   actualizeaza inaltimea si intoarce radacina */
AVLink s_rotate_right(AVLink X)
{ AVLink kZ;
   Z = X->lt;
   X->lt = Z->rt;
   Z->rt = X;
   X->h = max(height(X->lt),height(X->rt))+1;
   Z->h = max(height(Z->lt), Z->h )+1;
   return Z;
}
```

Rotire dublă Left Right

```
/* functia se poate apela daca X are un
   copil stang si copilul stang a lui X are un
   copil drept */

AVLink d_rotate_right(AVLink X)
{ /* roteste intre Z si Y */
  X->lt = s_rotate_left(X->lt);
  /* roteste intre X si Y */
  return(s_rotate_right(X));
}
```