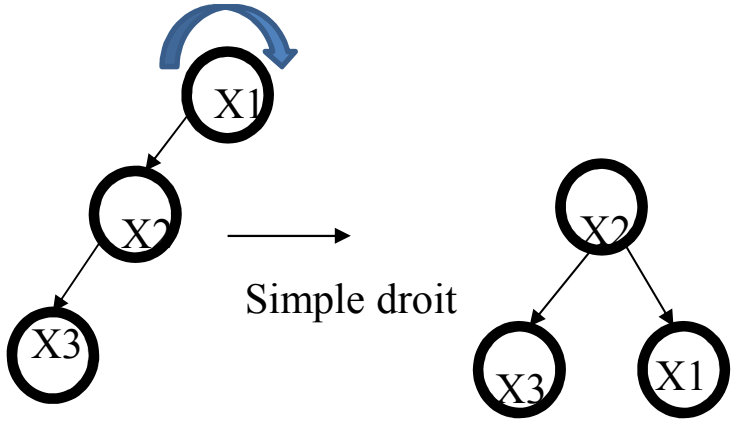
Arbres AVL

Algorithmique et Structures de Données Dynamiques (ALSDD)
1ière année CPI

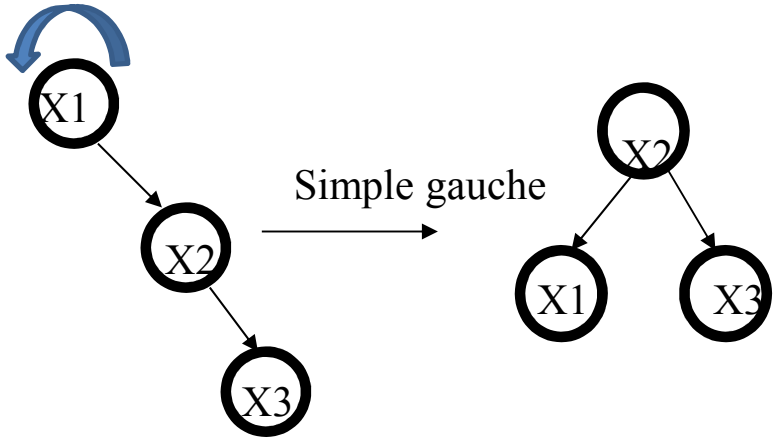
Dr. Boulakradeche Mohamed (MCB) & Dr. Kermi Adel (MCB)

Aff_fg(N,Ngd)
Aff_fd(Ng,N)
Aff_fd(parentN,Ng)

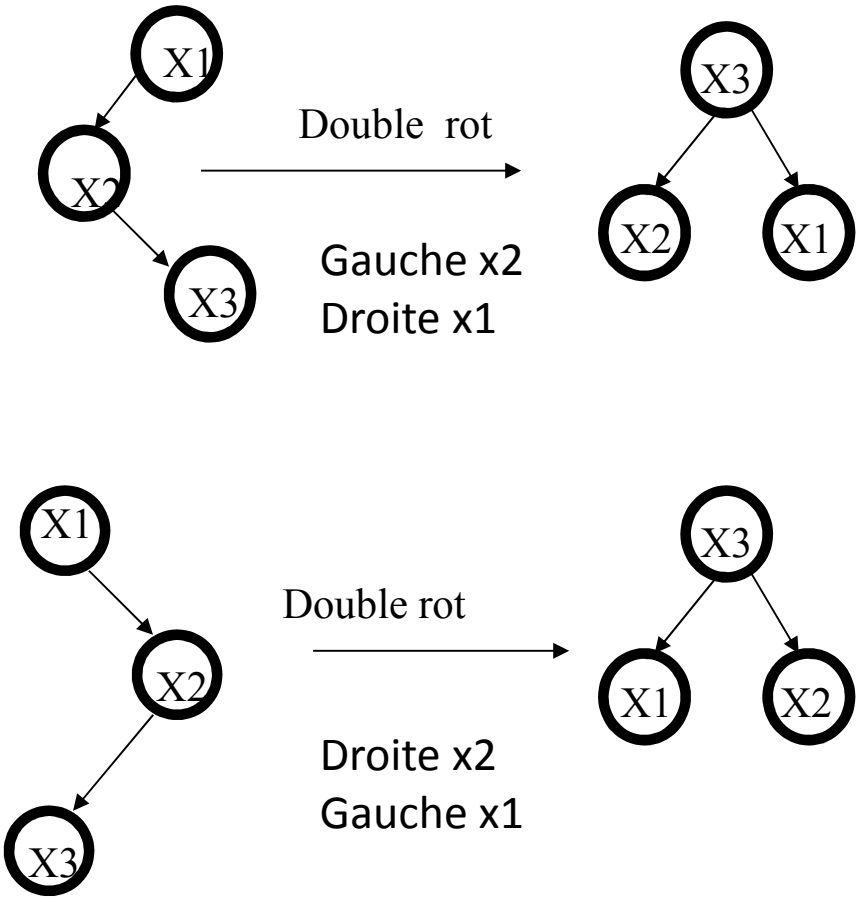


Aff_fg(X1,nil)
Aff_fd(X2,X1)

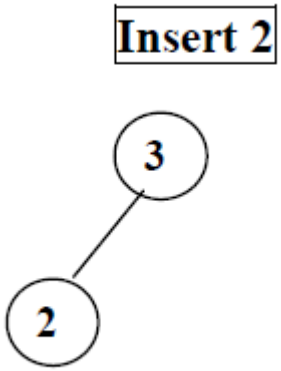
Aff_fd(N,Ndg)
Aff_fg(Nd,N)
Aff_fg(parentN,Nd)



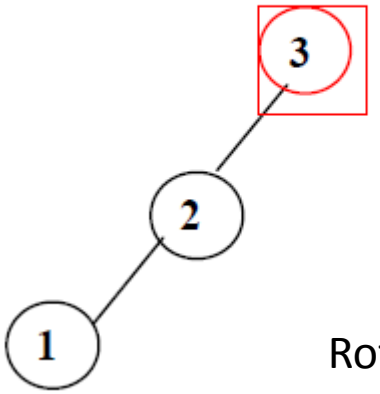
Aff_fd(X1,nil)
Aff_fg(X2,X1)


X1
X2
X3
Double rot
Gauche x2
Droite x1
X3
X2
X1
X1
X2
X3
Double rot
Droite x2
Gauche x1
X3
X1
X2

Insert 3

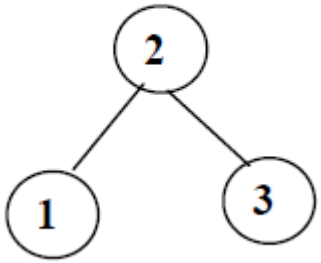

Insert 2
3
2

Insert 1 (non-AVL)

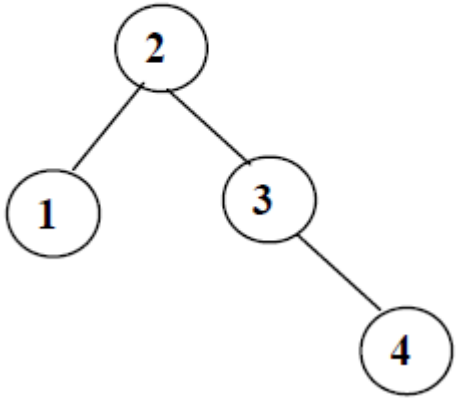


Rotation droite de 3

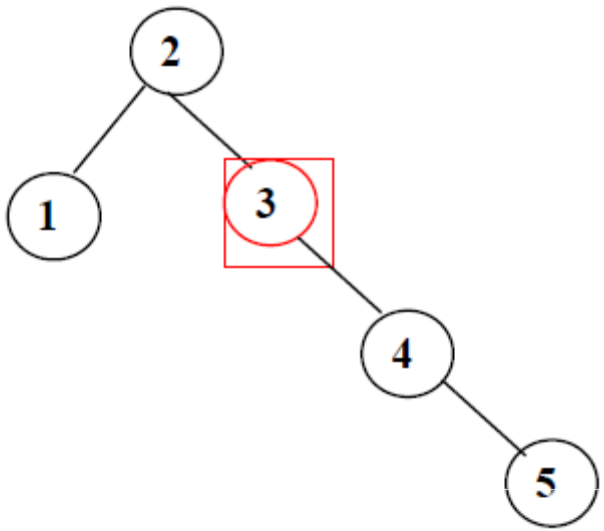
AVL



Insert 4

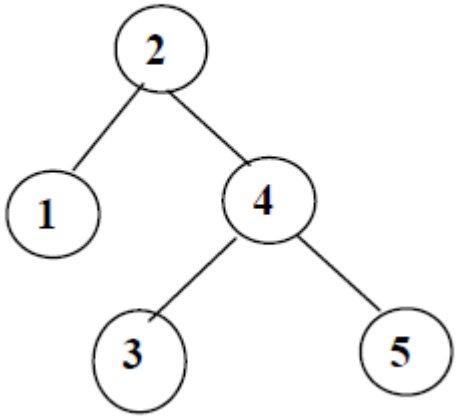


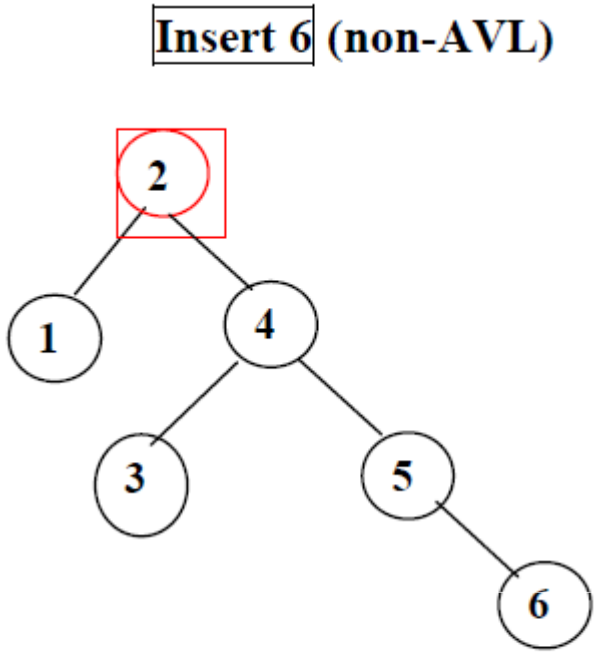
Insert 5 (non-AVL)



Rotation gauche de 3

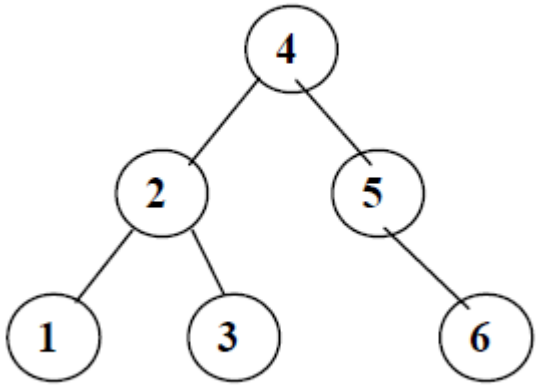
AVL

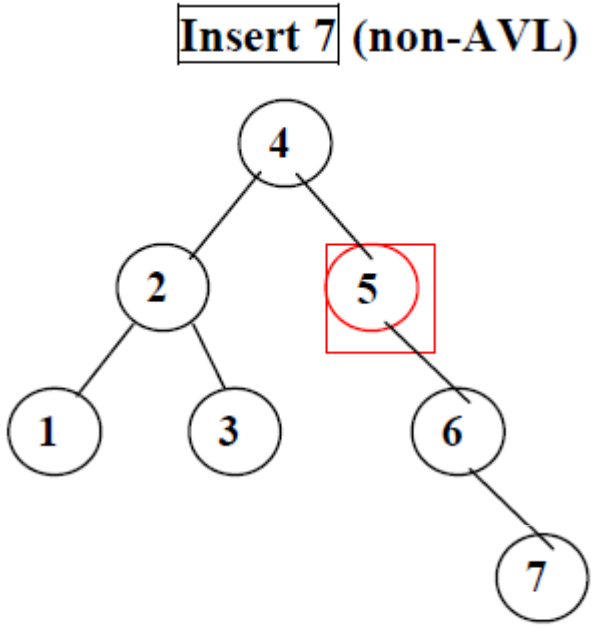




Rotation Gauche de 2

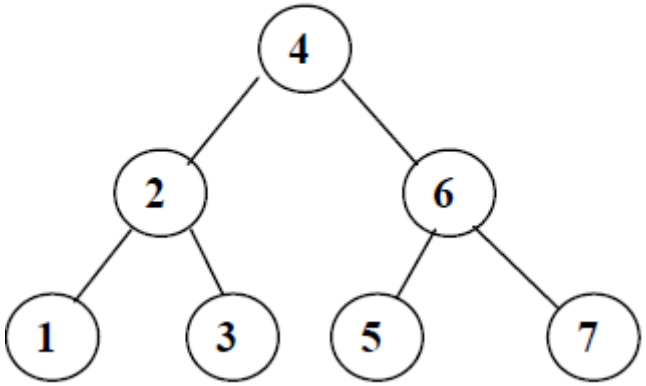
AVL



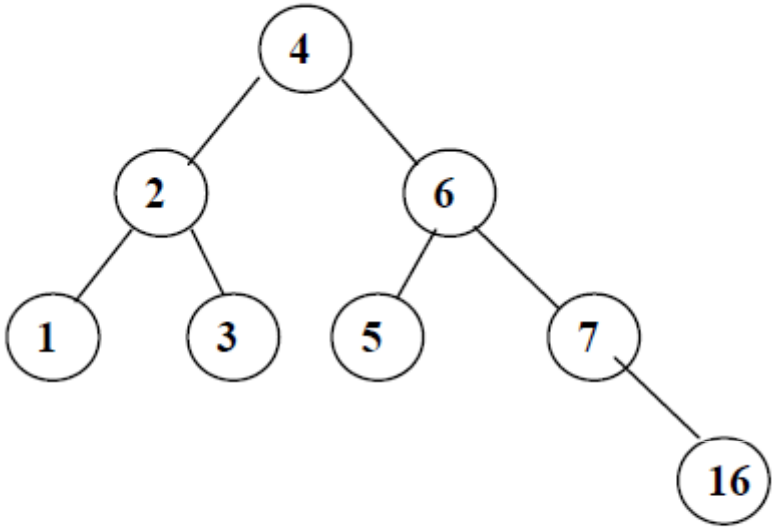


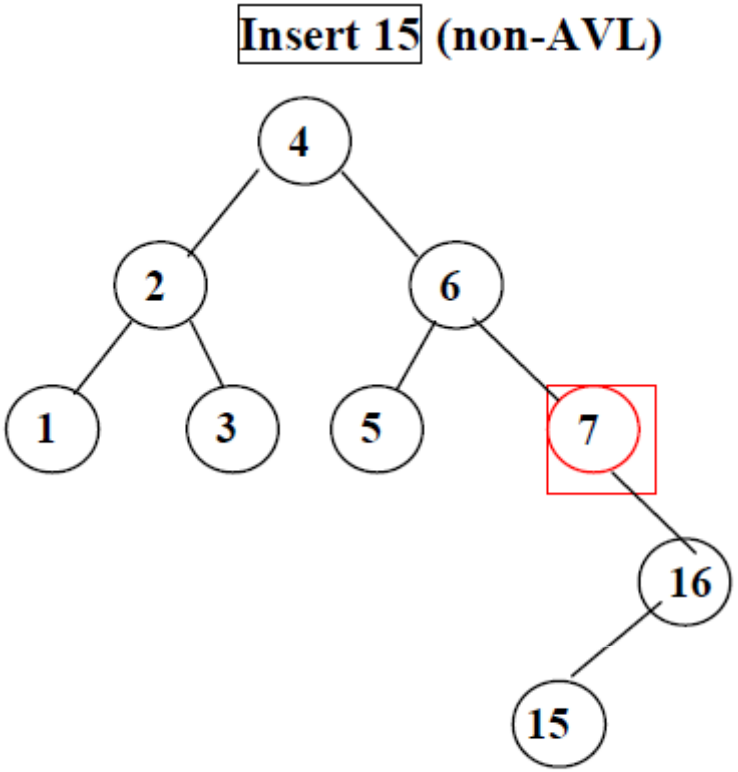
Rotation Gauche de 5

AVL

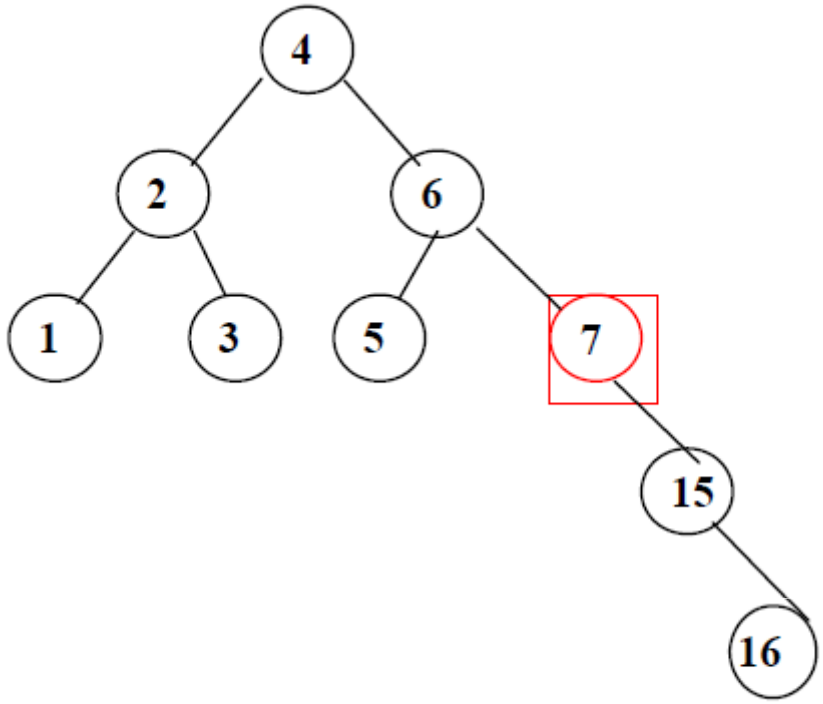


Insert 16



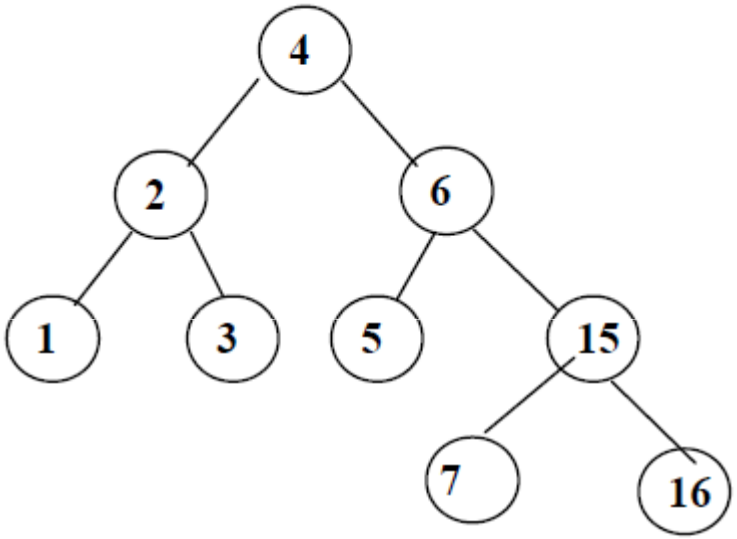


Double Rotation Droite 16 puis gauche 7

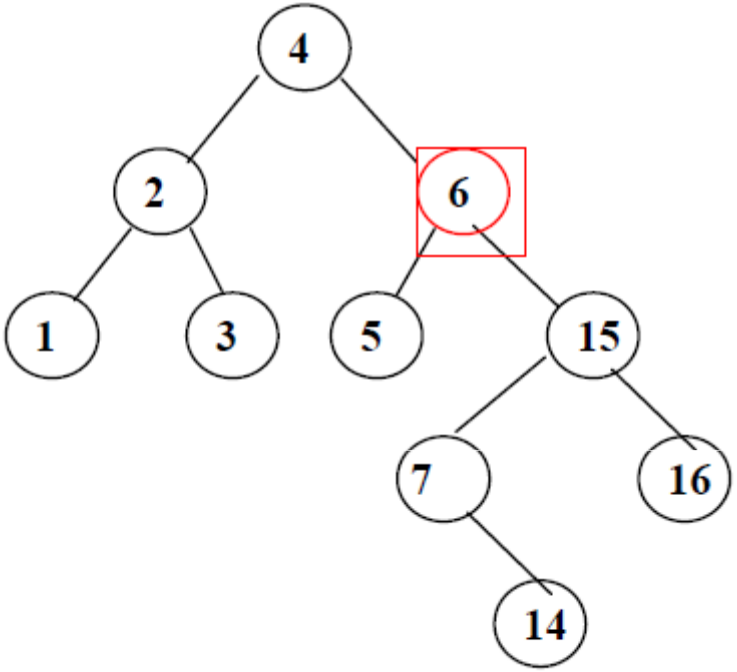


Apres RD 16

Apres RG 7 AVL



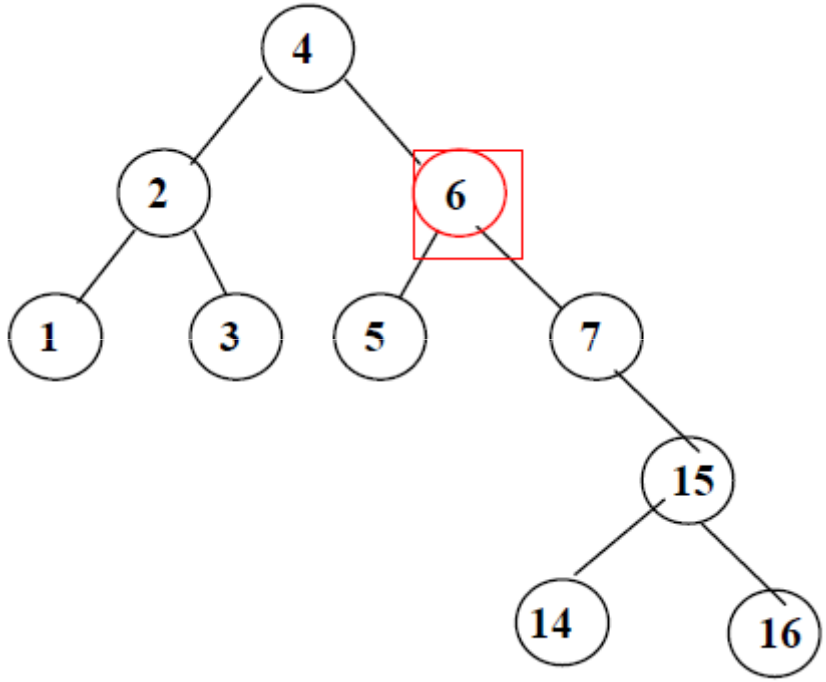
Insert 14 (non-AVL)



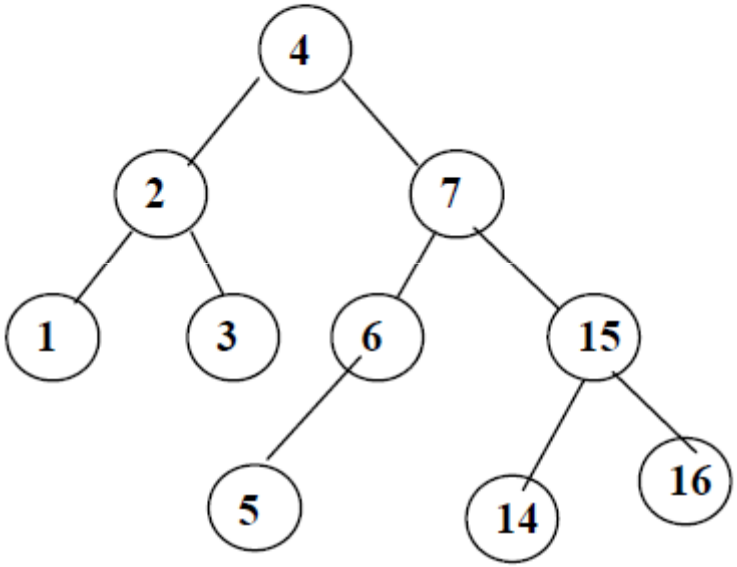
Double Rotation

RD 15 puis RG 6

Après RD 15



Après RG 6



Fonction d'insertion avec équilibrage

```
fonction inserer(x:entier,  a:ptrnoeud):ptrnoeud;
debut
    si a = nil alors inserer:=creernoeud(nil,x,nil)
    sinon   si x< info(a) alors aff_fg(a,inserer(x,fg(a)))
                sinon
                  si x>info(a) alors aff_fd(a,inserer(x,fd(a))) finsi
            finsi
    finsi
    equilibrer(a)
fin;
```

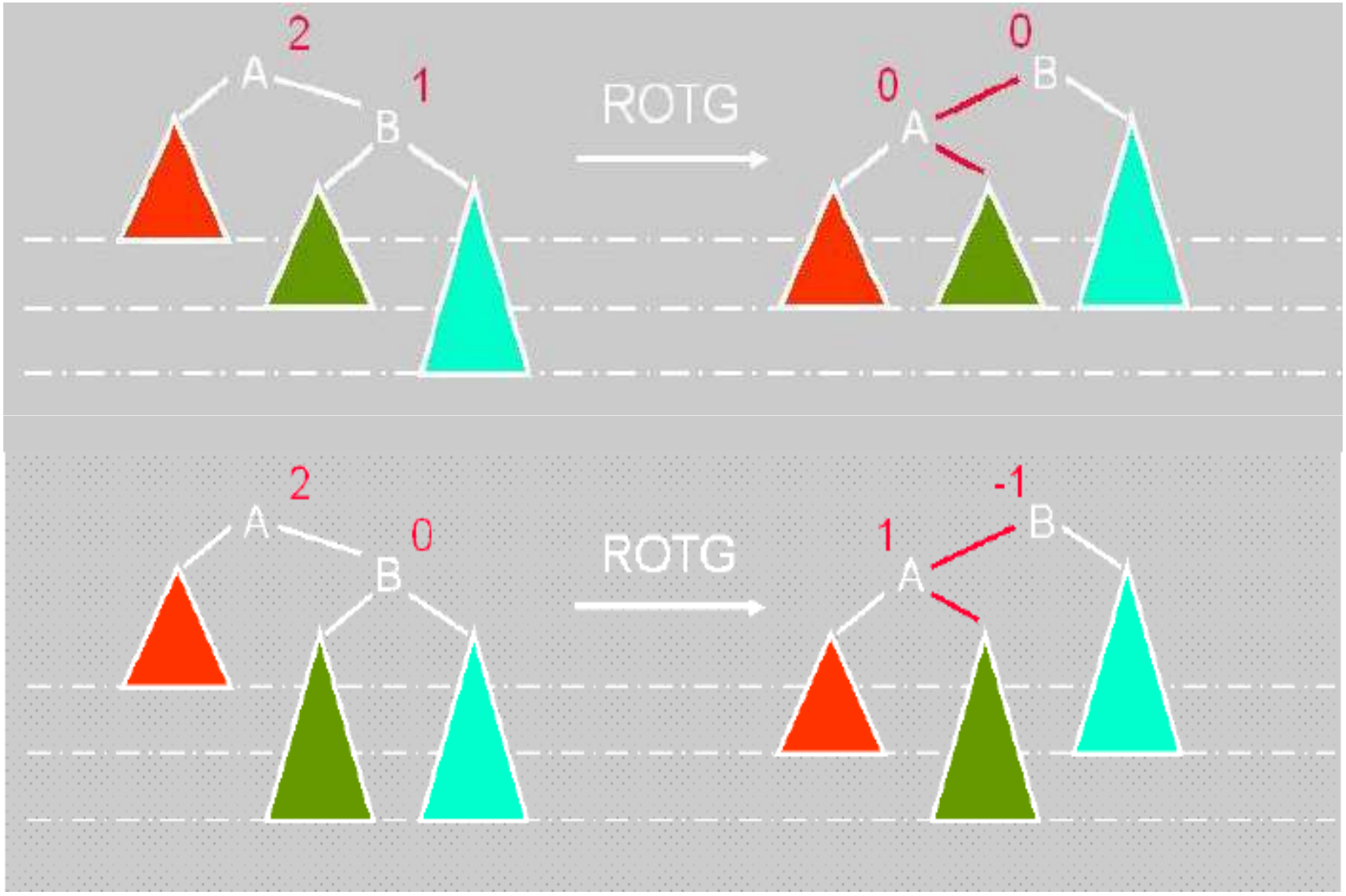
Procédure equilibrer(a:ptrnoeud)

```
Si |h(Fg)-h(Fd)| <= 1, ne rien faire
   Sinon
    Si h(Fg)-h(Fd) = 2
       Si h(Fg(Fg)) > h(Fg(Fd)) Alors rd(N)
       Sinon rg(Fg) puis rd(N)
     Sinon (h(Ng)-h(Nd) = -2)
       Si h(Fd(Fd)) > h(Fd(Fg)) Alors rg(N)
       Sinon rd(Nd) puis rg(N)
   Fsi
Fsi
```

Après insertion si non AVL une restructuration suffit.

Restructuration = une rotation simple ou une double rotation

Rotation à gauche :



Algorithme d'une Rotation à Gauche :

```
Fonction ROTG (A : ptr(Nœud)) : ptr(Nœud)
Var B : ptr(Nœud) ; x, y : entier;
Début
        B := FD(A);
        x := bal(A);  y := bal(B);
        FD(A) := FG(B);
        FG(B) := A;  /* rotation */
        FD(PÈRE(A)) := B; /* si PÈRE(A) <> NIL */
        A.bal := x – min (y , 0) + 1; ou  A.bal := bal(A);
        B.bal := max (x + 2, x+y+2, y+1); ou B.bal := bal(B);
        ROTG  := B
Fin
```

```
Aff_fd(N, FG(FD(N)))
Aff_fg(FD(N),N)
Aff_fg(PÈRE(N),FD(N))
```

```
Fonction bal (A : ptr(Nœud)) : entier
Début
        bal := profondeur(FG(A)) – profondeur(FD(A))
Fin
```

Fonction de Rotation à Gauche en C:

```
arbre ROTG(arbre A) {
        arbre B; int a,b;
        B = A->d;
        a = A->bal;  b = B->bal;
        A->d = B->g;
        B->g = A;  /* rotation */
        A->bal = a-max(b,0)-1;
        B->bal = min(a-2,a+b-2,b-1);
        return B;
}
```

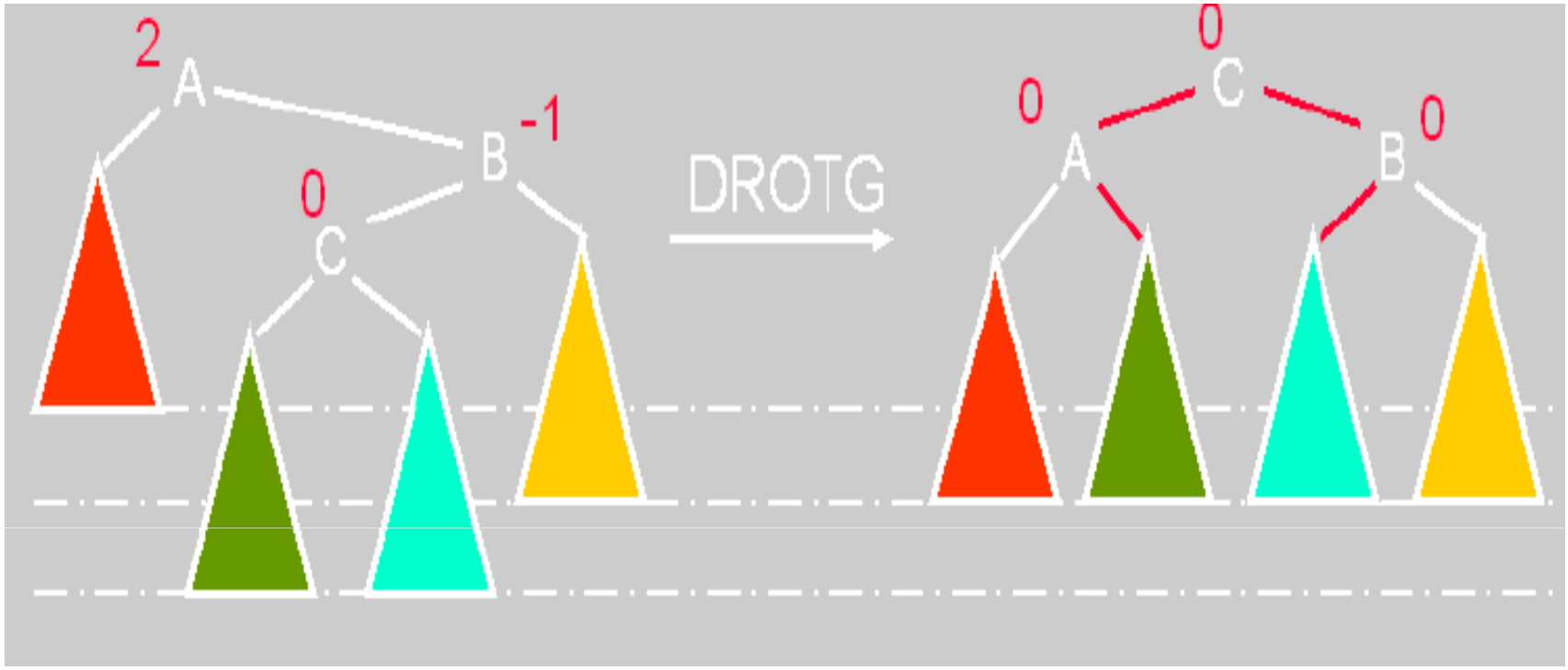
Algorithme d'une Rotation à Droite :

```
Fonction ROTD (A : ptr(Nœud)) : ptr(Nœud)
Var B : ptr(Nœud) ; x, y : entier;
Début
        B := FG(A);
        x := bal(A);  y := bal(B);
        FG(A) := FD(B);
        FD(B) := A;  /* rotation */
        FG(PÈRE(A)) := B; /* si PÈRE(A) <> NIL */
        A.bal := x – max(y , 0) – 1; ou A.bal := bal(A);
        B.bal:= min(x – 2, x+y-2, y-1); ou B.bal := bal(B);
        ROTD  := B
Fin
```

Fonction de Rotation à Droite en C:

```
arbre ROTD(arbre A) {
        arbre B; int a,b;
        B = A->g;
        a = A->bal;  b = B->bal;
        A->g = B->d;
        B->d = A;  /* rotation */
        A->bal = a-min(b,0)+1;
        B->bal = max(a+2,a+b+2,b+1);
        return B;
}
```

Double Rotation Gauche :



Fonction DROTG Algo.:

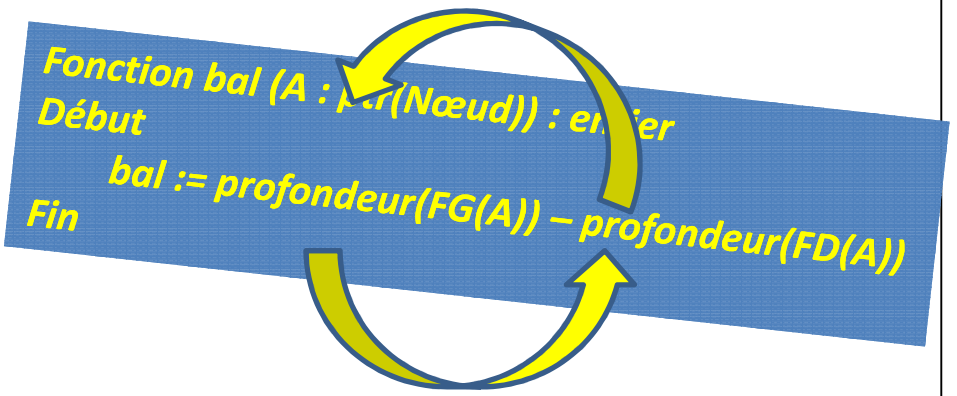
```
Fonction DROTG (A : ptr(Nœud)) : ptr(Nœud)
Début
        FD(A) := ROTD (FD(A));
        DROTG  := ROTG (A);
Fin
```

Fonction DROTG en C:

```
arbre DROTG(arbre A)  {
        A->d = ROTD(A->d);
        return ROTG(A);
}
```

Algo. Fonction Equilibrage d'un arbre A:

```
Fonction EQUILIBRER ( A : ptr(Nœud)) : ptr(Nœud)
Début
    SI  (bal (A) = -2)  ALORS      /* bal(A) retourne le facteur de déséquilibre du nœud A */
        SI (bal (FD (A))  <=  0) ALORS
            EQUILIBRER := ROTG (A)
        SINON
            FD(A) :=  ROTD(FD(A));
            EQUILIBRER :=  ROTG(A);
        FINSI
    SINON
        SI  (bal(A) = 2)  ALORS
            SI (bal(FG(A)) >= 0)  ALORS
                EQUILIBRER := ROTD(A)
            SINON
                FG(A) := ROTG(FG(A));
                EQUILIBRER := ROTD(A);
            FINSI
        SINON   EQUILIBRER := A;
        FINSI
    FINSI
Fin
```



Algo. Fonction Equilibrage d'un arbre A (bis):

```
Fonction EQUILIBRER ( A : ptr(Nœud)) : ptr(Nœud)      /* AUTRE FONCTION  */
Début
   SI  (bal (A) = -2)  ALORS     /* bal(A) retourne le facteur de déséquilibre du nœud A */
      SI (bal (FD (A))  <=  0) ALORS
         EQUILIBRER := ROTG (A)
      SINON
         FD(A) :=  ROTD(FD(A));
         EQUILIBRER :=  ROTG(A);
         // ou bien EQUILIBRER := DROTG(A);
      FINSI
   SINON
      SI  (bal(A) = 2)  ALORS
         SI (bal(FG(A)) >= 0)  ALORS
            EQUILIBRER := ROTD(A)
         SINON
            FG(A) := ROTG(FG(A));
            EQUILIBRER := ROTD(A);
            // ou bien EQUILIBRER := DROTD(A);
         FINSI
      SINON   EQUILIBRER := A;
      FINSI
   FINSI
Fin
```

```
Fonction bal (A : ptr(Nœud)) : entier
Début
      bal := profondeur(FG(A)) – profondeur(FD(A))
Fin
```

Fonction Equilibrage d'un arbre A en C:

```
arbre EQUILIBRER(arbre A) {
   if (A->bal == 2)
         if (A->d->bal >= 0)
            return ROTG(A);
         else {
            A->d = ROTD(A->d);
            return ROTG(A);
         }
   else if (A->bal == -2)
         if (A->g->bal <= 0)
            return ROTD(A);
         else {
            A->g = ROTG(A->g);
            return ROTD(A);
         }
   else return A;
}
```

Fonction d'insertion « AJOUTER » en C :

```
(arbre,int) AJOUTER(element x, arbre A) {
  if (A == NULL) {
        créer un nœud A;
        A->g = NULL;
        A->d = NULL;
        A->elt = x;  A->bal = 0;
        return (A,1);
  }
  else if (x == A->elt)
        return (A,0);
  else if (x > A->elt)
        (A->d,h) = AJOUTER(x,A->d);
  else
        (A->g,h) = AJOUTER(x,A->g); h = -h;
  if (h == 0)
        return (A,0);
  else {
        A->bal = A->bal + h;
        A = EQUILIBRER(A);
        if (A->bal == 0) return (A,0);
        else return (A,1);
  }
}
```

/* retourne l'arbre modifié et la différence de hauteur : 0 ou +1 */

Fonction de suppression « ENLEVER » en C :

```
(arbre,int) ENLEVER(element x, arbre A) {
  if (A == NULL)
        return (A,0);
  else if (x > A->elt)
        (A->d,h) = ENLEVER(x,A->d);
  else if (x < A->elt)
        (A->g,h) = ENLEVER(x,A->g); h = -h;
  else if (A->g == NULL)
        /* free */ return (A->d,-1);
  else if (A->d == NULL)
        /* free */ return (A->g,-1);
  else {
        A->elt = min(A->d);
        (A->d,h) = OTERMIN(A->d);
  }
  if (h == 0)
        return (A,0);
  else {
        A->bal = A->bal + h;
        A = EQUILIBRER(A);
        if (A->bal == 0)
          return (A,-1);
        else
          return (A,0);
  }
}
```

/* retourne l'arbre modifié et la différence de hauteur : -1 ou 0 */

```
(arbre,int) OTERMIN(arbre A) {
  if (A->g == NULL) {
       min = A->elt;
       /* free */ return (A->d,-1);
  } else
       (A->g,h) = OTERMIN(A->g); h = -h;
  if (h == 0)
       return (A,0);
  else {
       A->bal = A->bal + h;
       A = EQUILIBRER(A);
       if (A->bal == 0)
         return (A,-1);
       else
         return (A,0);
  }
}
```

/* retourne l'arbre modifié et la différence de hauteur : -1 ou 0 */