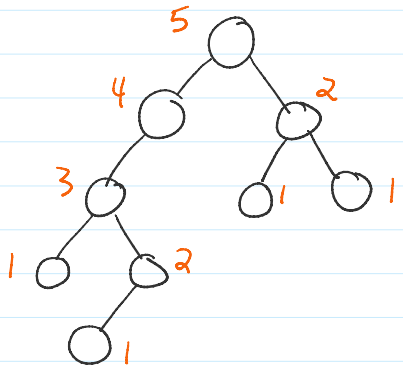
Arbres AVL

28 octobre 2024 19:24

Hauteur d'un noeud v = haut. du sous-arbre enraciné en v.



orange = haut

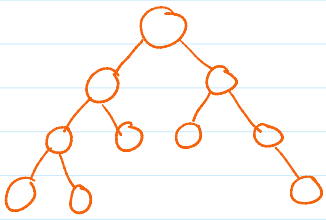
haut(nullptr) = 0

← non-balancé

Arbre AVL : arbre bien balancé

Pour tout noeud v, $|x|$: val abs. de x

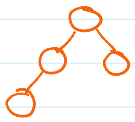
$$|\,haut(v \to gauche) - haut(v \to droit)\,| \leq 1$$



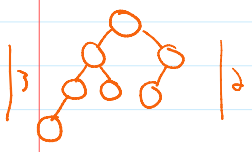
- Hauteur d'un arbre AVL avec n noeuds ?
  - L h est $O(\log n)$ pas trivial

Argument : pour une hauteur h,
f(h) = # min de noeuds d'un AVL de hauteur h.

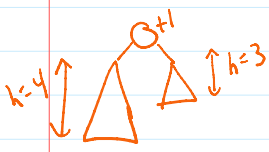
h	f(h)
1	1
2	2



3 4



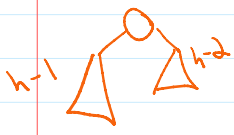
4 7



5 12

6 20

...



h $1+f(h-1)+f(h-2)$

$f(h) = 1 + f(h-1) + f(h-2)$

$\geq \varphi^h$ φ = ratio d'or $\frac{1+\sqrt{5}}{2} \approx 1.618$

$\geq 1.618^h$ (preuve par induction)

$f(h) \geq f(h-1) + f(h-2)$

$\geq \varphi^{h-1} + \varphi^{h-2}$

$= \varphi \cdot \varphi^{h-2} + \varphi^{h-2}$

$= (\varphi - 1)\varphi^{h-2}$

$\geq \varphi^2 \cdot \varphi^{h-2} = \varphi^h$

- Donc un arbre AVL de hauteur h a au moins 1.618^h noeuds

$$n \geq 1.618^h$$

$$\log n \geq \log 1.618^h$$

$$\log n \geq h \cdot \log 1.618$$

$$\log n \geq h \cdot c$$

$$\frac{\log n}{c} \geq h \Rightarrow h \text{ est } O(\log n)$$

Arbre AVL : insertion $O(\log n)$
suppression $O(\log n)$
recherche $O(\log n)$

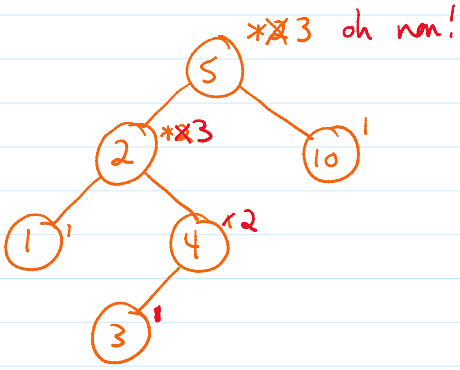
Défi : après une insertion (ou suppression),
s'assurer que l'arbre est bien balancé.

① Chaque noeud connaît sa hauteur

```
struct Noeud{
    Noeud* gauche, droit, parent;
    TYPE val;
    size_t haut;   //a maintenir
    ...
};
```

② À l'insertion, mettre à jour les hauteurs
+ détecter déséquilibrement

insert 5-2-10-1-4-3




```
void inserer( Noeud* v, TYPE& val){

    if(val < v->val){
        ...
    }
    else (val > v->val){
        ...
    }
    //insertion faite, descendants à jour
```

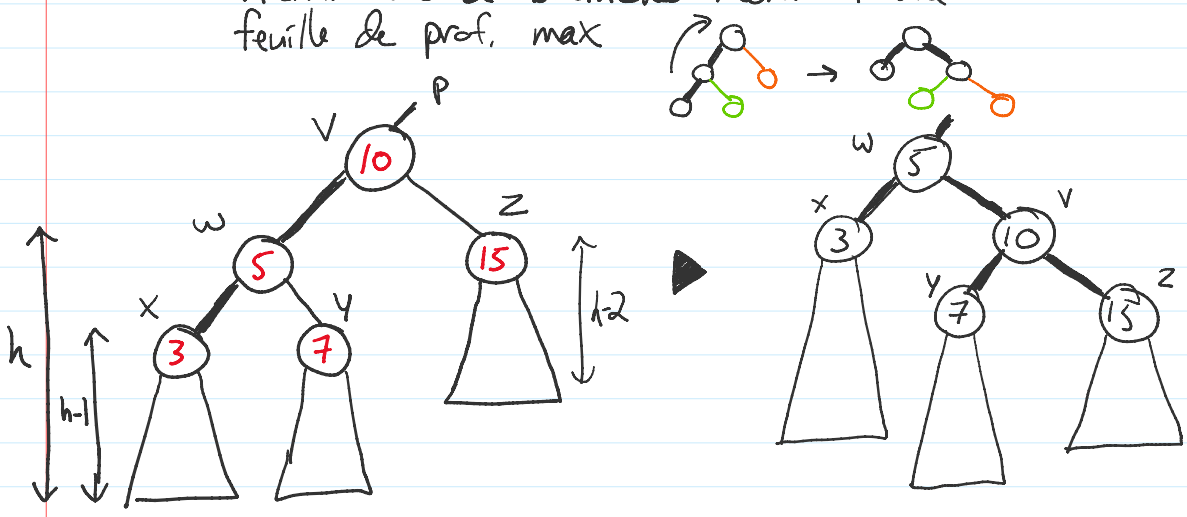
```
    }
    //insertion faite, descendants à jour
    v->hauteur = 1 + max(getH(v->gauche), getH(v->droit));

    if ( abs( getH(v->gauche) - getH(v->droit)) > 1)
    {
        //rebalancer : rotation(v)
    }
}
```

Pour rebalancer, il faut modifier le sous-arbre en v.

On applique une rotation.

Rotation : s'applique sur 2 arêtes consécutives.

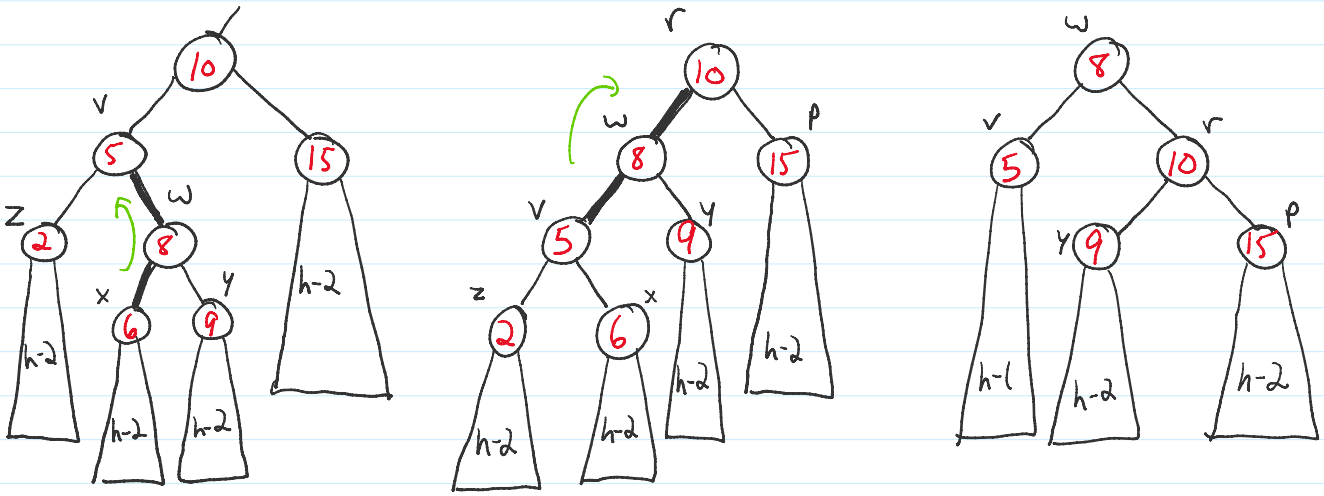
Prendre les 2 branches menant à une feuille de prof. max




```
rotation_gg(v){
    w = v->gauche
    x = w->gauche
    y = w->droit
    z = v->droit
    p = v->parent
```

```
y = w → droit
z = v → droit
p = v → parent

if (v == p → gauche)
    p → gauche = w
else
    p → droit = w
w → parent = p
w → droit = v , v → parent = w
v → gauche = y, y → parent = v
v → droit = z,  z → parent = v

v → haut = max(y → haut, z → haut) + 1
w → haut = max(x → haut, v → haut) + 1
}
```

Rotation g-d : il faut 2 rotations :
1 rotation + basse et 1 gg/dd



Temps d'une insertion :

Analyse 1 : 1) parcourir $O(h) = O(\log n)$ noeuds par trouver où insérer

2) insérer $O(1)$

3) remonter $O(h) = O(\log n)$ noeuds pour faire 1 ou 2 rotations par noeud.

1 rotation : $O(1)$

Total : $1) + 2) + 3) = O(\log n + 1 + \log n)$

$= O(\log n)$

Analyse 2 : Temps = nb-appels-rec × temps par appel

$h = O(\log n)$; $O(1)$ car rotation constant

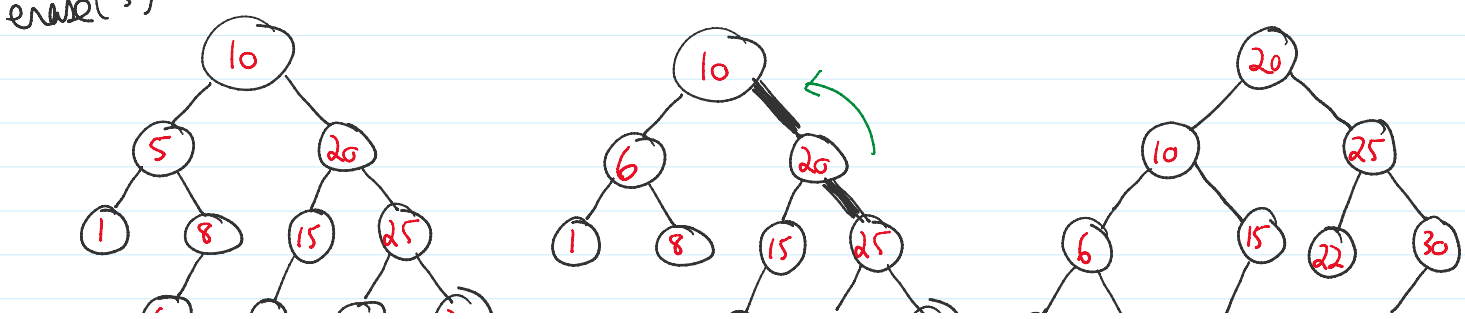
$\rightarrow O(\log n)$

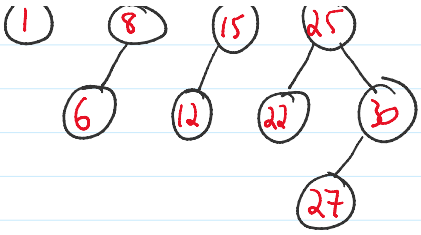
- Supprimer un noeud (erase(val))

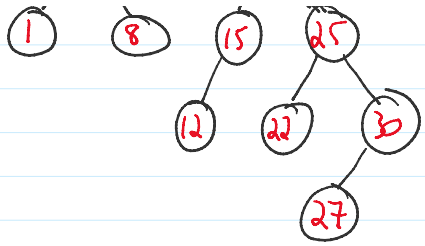
1) Supprimer comme dans un ABR.
   - soit v le noeud supprimé

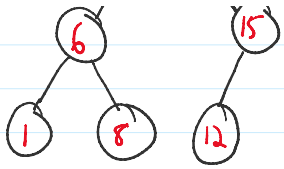
2) Pour chaque w de v à la racine
   - voir s'il y a déséquilibrement
     si oui, rotations

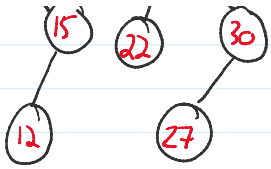
erase(5)




1
8
15
25
6
12
22
30
27


1
8
15
25
12
22
30
27


6
1
8


15
12
22
30
27