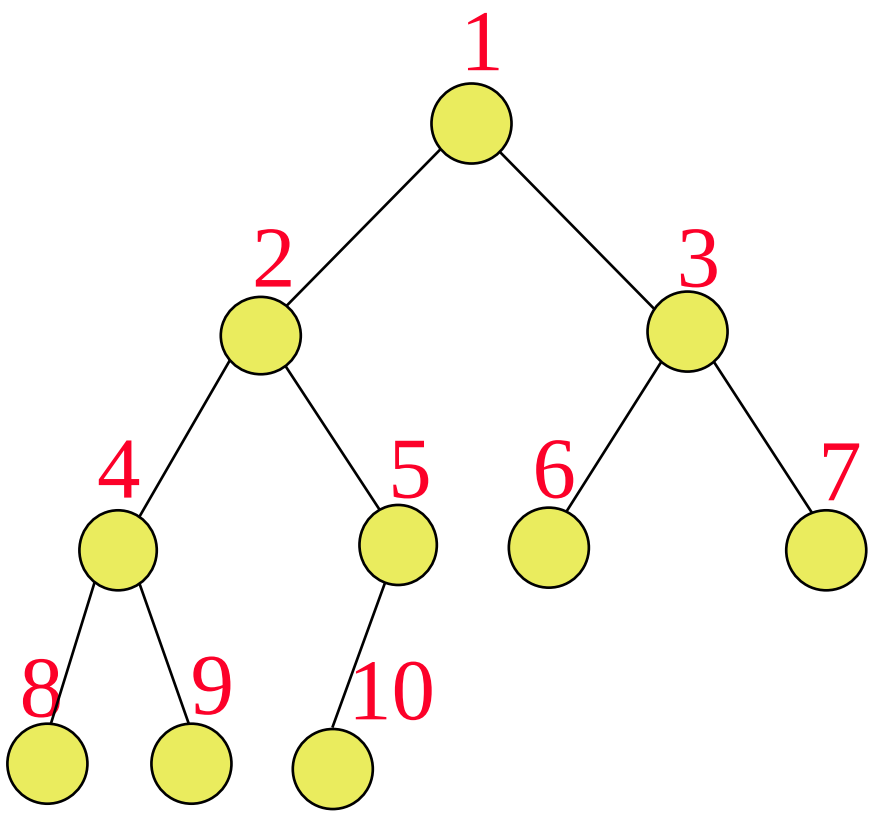
Parcours d'arbres



Préfixe : 1, 2, 4, 8, 9, 5, 10, 3, 6, 7
(VGD)

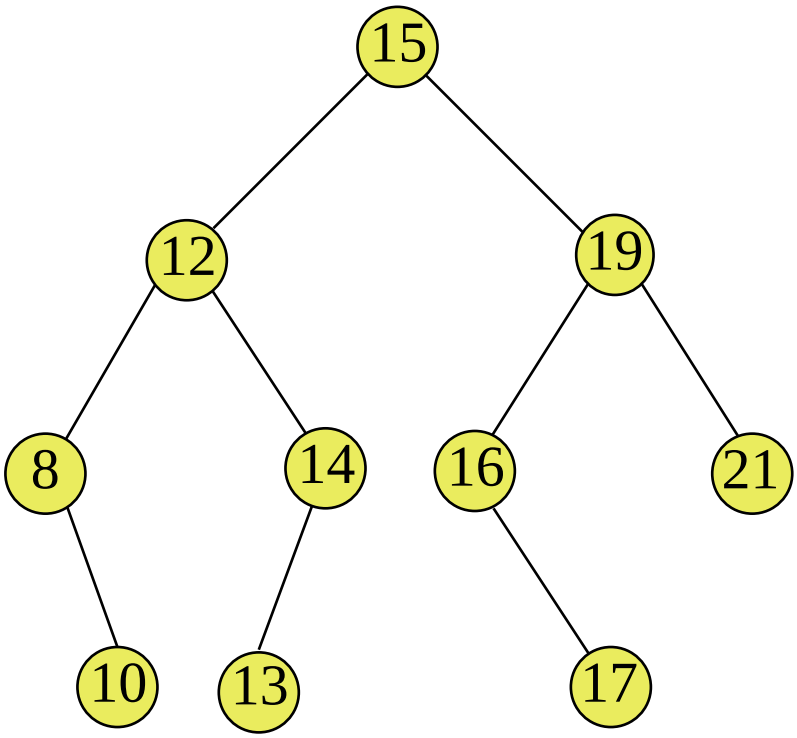
Infixe : 8, 4, 9, 2, 10, 5, 1, 6, 3, 7
(GVD)

Suffixe : 8, 9, 4, 10, 5, 2, 6, 7, 3, 1
(GDV)

```
typedef struct Noeud
{
  Element       contenu;
  struct Noeud   *filsG;
  struct Noeud   *filsD;
}*Arbre;


void ParcoursPrefixe(Arbre a)
{
  if (a != NULL)
  {
    printf("%3d", a->contenu);
    ParcoursPrefixe(a->filsG);
    ParcoursPrefixe(a->filsD);
  }
}
```

```
void ParcoursInfixe(Arbre a)
{
  if (a != NULL)
  {
    ParcoursInfixe(a->filsG);
    printf("%3d", a->contenu);
    ParcoursInfixe(a->filsD);
  }
}

void ParcoursSuffixe(Arbre a)
{
  if (a != NULL)
  {
    ParcoursSuffixe(a->filsG);
    ParcoursSuffixe(a->filsD);
    printf("%3d", a->contenu);
  }
}
```

Arbres de recherche



Propriété de base : Pour chaque noeud de valeur v, les noeuds du sous-arbre gauche ont une valeur < v et ceux du sous-arbre droit ont une valeur > v.

Exercices

(1) Montrer que le parcours infixe ordonne les nœuds par valeur croissante.

(2) Montrer que si un nœud a deux fils, son successeur dans l'ordre infixe n'a pas de fils gauche et son prédécesseur n'a pas de fils droit.

(3) Montrer que le successeur du nœud n est le sommet le plus à gauche dans le sous-arbre droit issu de n.

Recherche

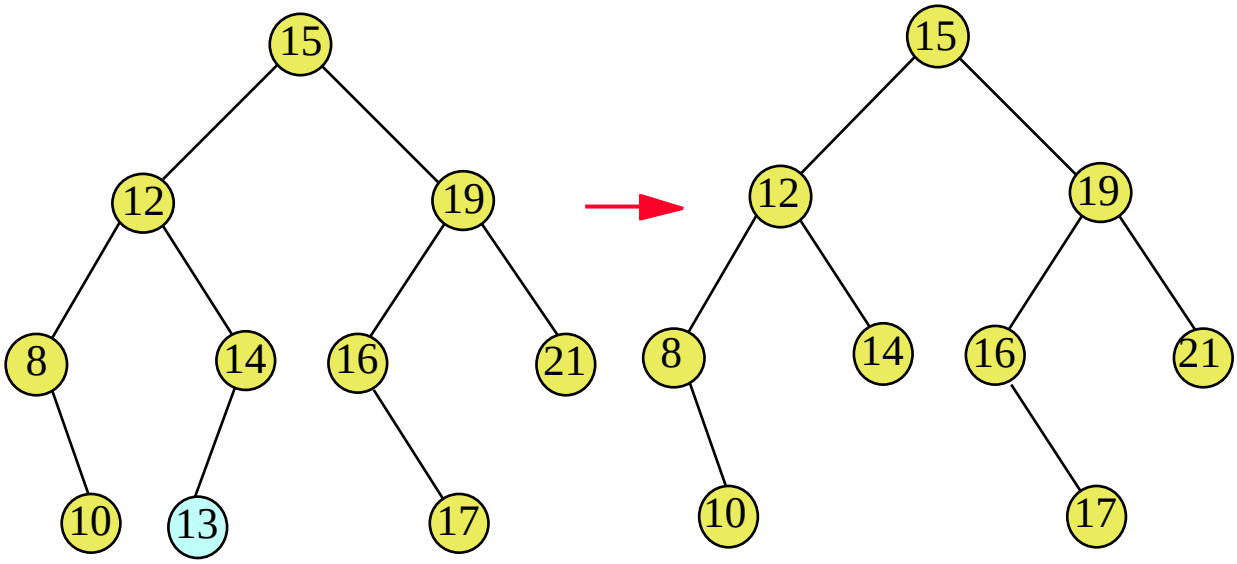
```
Arbre Recherche(Element v, Arbre a)
{
   if (a == NULL || v == a->contenu)
      return  a;
   if (v < a->contenu)
      return  Recherche(v, a->filsG);
   return  Recherche(v, a->filsD);
}
```

Ajout d'un élément

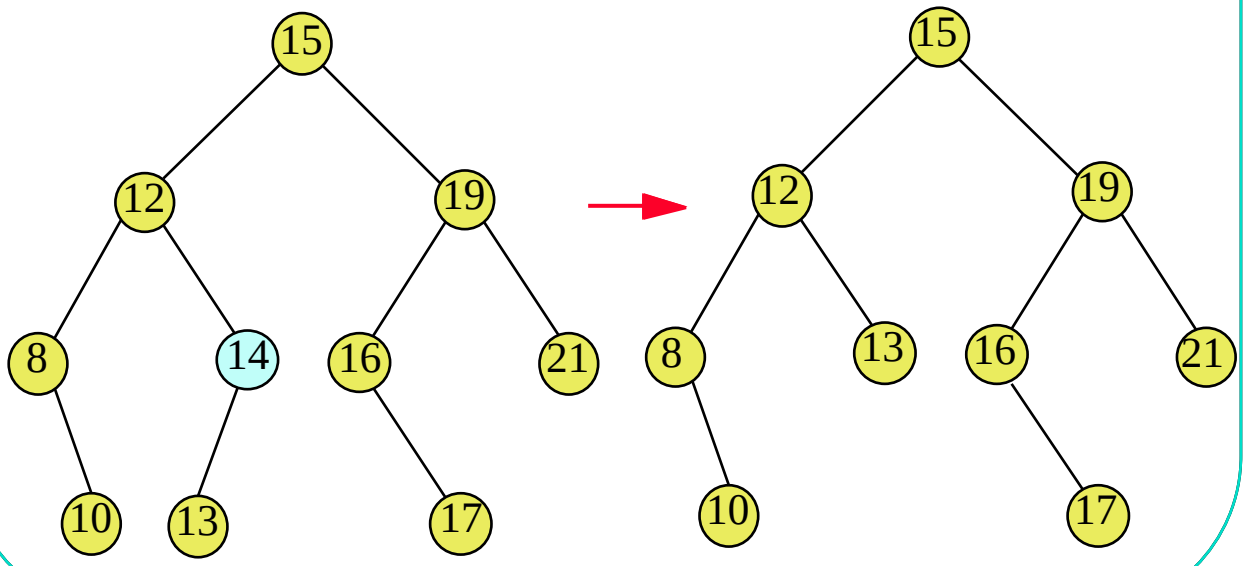
```
Arbre NouvelArbre(Element v, Arbre a,
Arbre b)
{
  Arbre     c;

  c = (Arbre)malloc (sizeof (struct  Noeud ));
  c->contenu = v;
  c->filsG = a;
  c->filsD = b;
  return  c;
}

void  AjouterArbre(Element v, Arbre *ap)
{
    Arbre a = *ap;

    if  (a == NULL )
        a = NouvelArbre(v, NULL , NULL );
    else if  (v <= a->contenu)
        AjouterArbre(v, &a->filsG);
    else
        AjouterArbre(v, &a->filsD);
    *ap = a;
}
```

Suppression

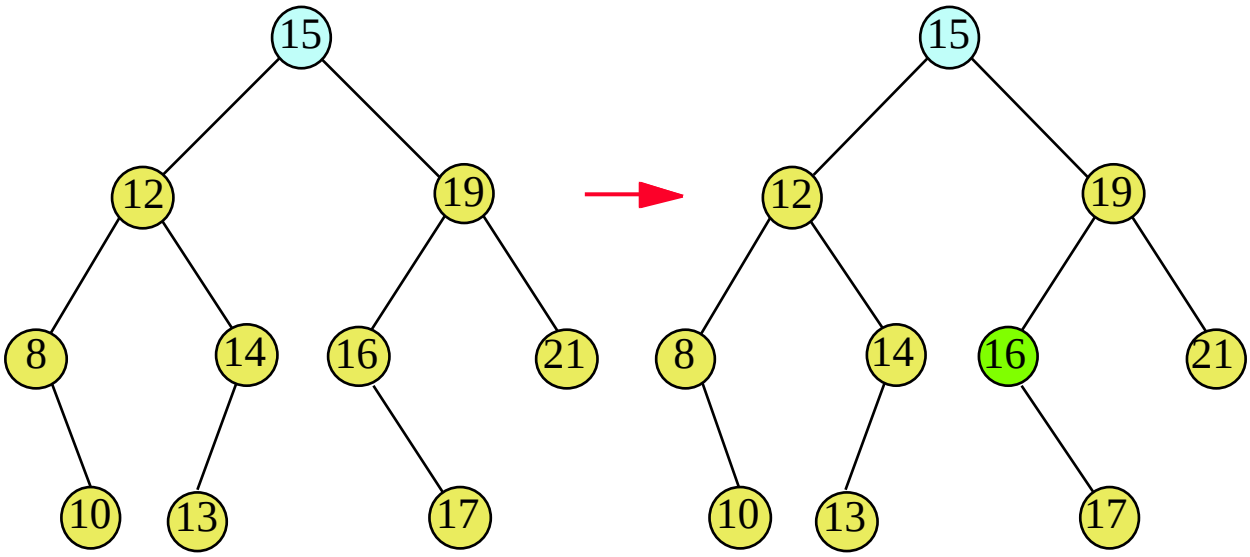
(a) le noeud est une feuille : on le supprime

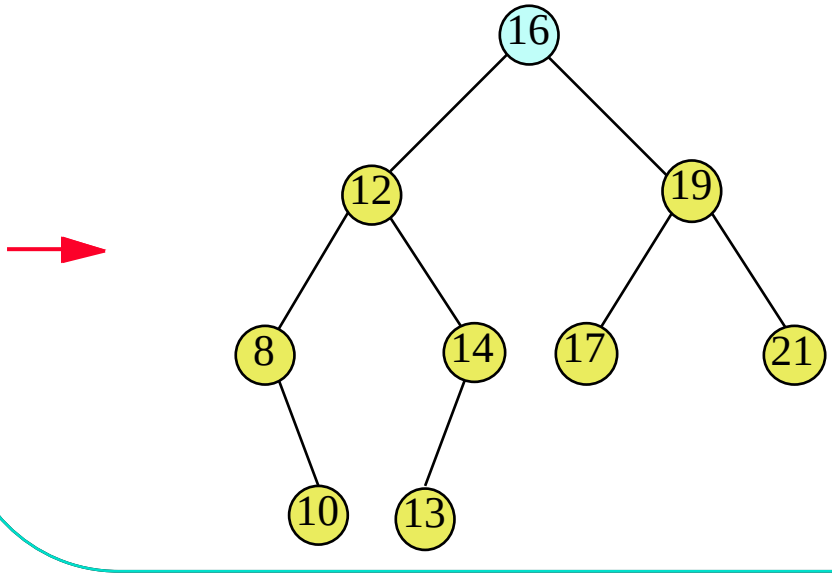


(b) le noeud est un a un seul fils : on la supprime



(c) le noeud s a deux fils : on supprime son successeur t (qui n'a pas de fils gauche), mais on remplace la valeur de s par celle de t.





```
void SupprimerArbre(Element v, Arbre *ap)
{
   Arbre a = *ap;

   if (a == NULL)
      *ap = a;
   else if (v < a->contenu)
      SupprimerArbre(v, &a->filsG);
   else if (v > a->contenu)
      SupprimerArbre(v, &a->filsD);
   else if (a->filsG == NULL)
      *ap = a->filsD;
   else if (a->filsD == NULL)
      *ap = a->filsG;
   else (*ap)->contenu =
                  SupprimerSuccesseur(&a);
}

Element SupprimerSuccesseur(Arbre *ap)
{
   Arbre a = *ap;
   Element v;

   if (a->filsG == NULL)
   {
      v = a->contenu;
      a = NULL;
      return v;
   }
   return SupprimerSuccesseur(&(a->filsG));
}
```

Temps de calcul

- O(log n) si l'arbre est équilibré
- O(n) si l'arbre est filiforme

--> D'où l'intérêt des arbres équilibrés!

- Arbre AVL (Adel'son-Vel'skii et Landis) : pour tout noeud, la différence de hauteur entre les sous-arbres gauche et droit est égale à -1, 0 ou 1.

Exemple : les tas

Hauteur d'un arbre AVL : O(log n)

```
typedef  struct  NoeudAVL
{
   int                balance;
   Element            contenu;
   struct  NoeudAVL   *filsG;
   struct  NoeudAVL   *filsD;
}*ArbreAVL;
```

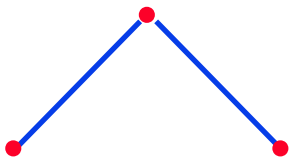
Théorème. Soit un arbre AVL de hauteur h à n sommets. Alors

$$\log_2(1 + n) \leq 1 + h \leq 1{,}44 \log_2(1 + n)$$

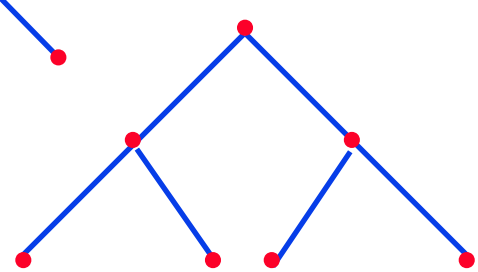
Preuve. Pour une hauteur h donnée, le nombre maximum de sommets est atteint pour l'arbre complet à $2^{h+1} - 1$ sommets.

Hauteur 0 :

Hauteur 1 :



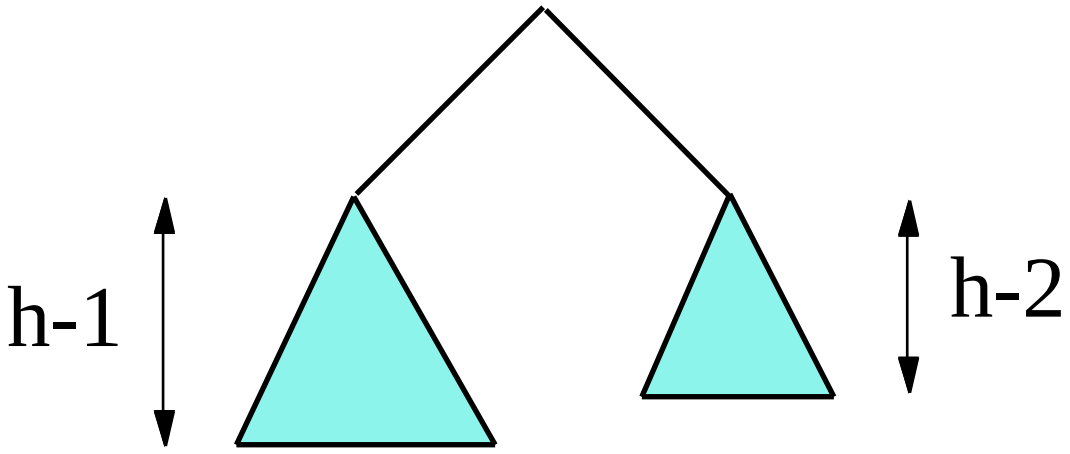
Hauteur 2 :



Donc $n \leq 2^{h+1} - 1$ et $\log_2(1 + n) \leq 1 + h$

Soit N(h) le nombre minimum de sommets d'un arbre AVL de hauteur h. On a

$$N(0) = 0 \qquad N(1) = 2$$
$$N(h) = 1 + N(h-1) + N(h-2)$$



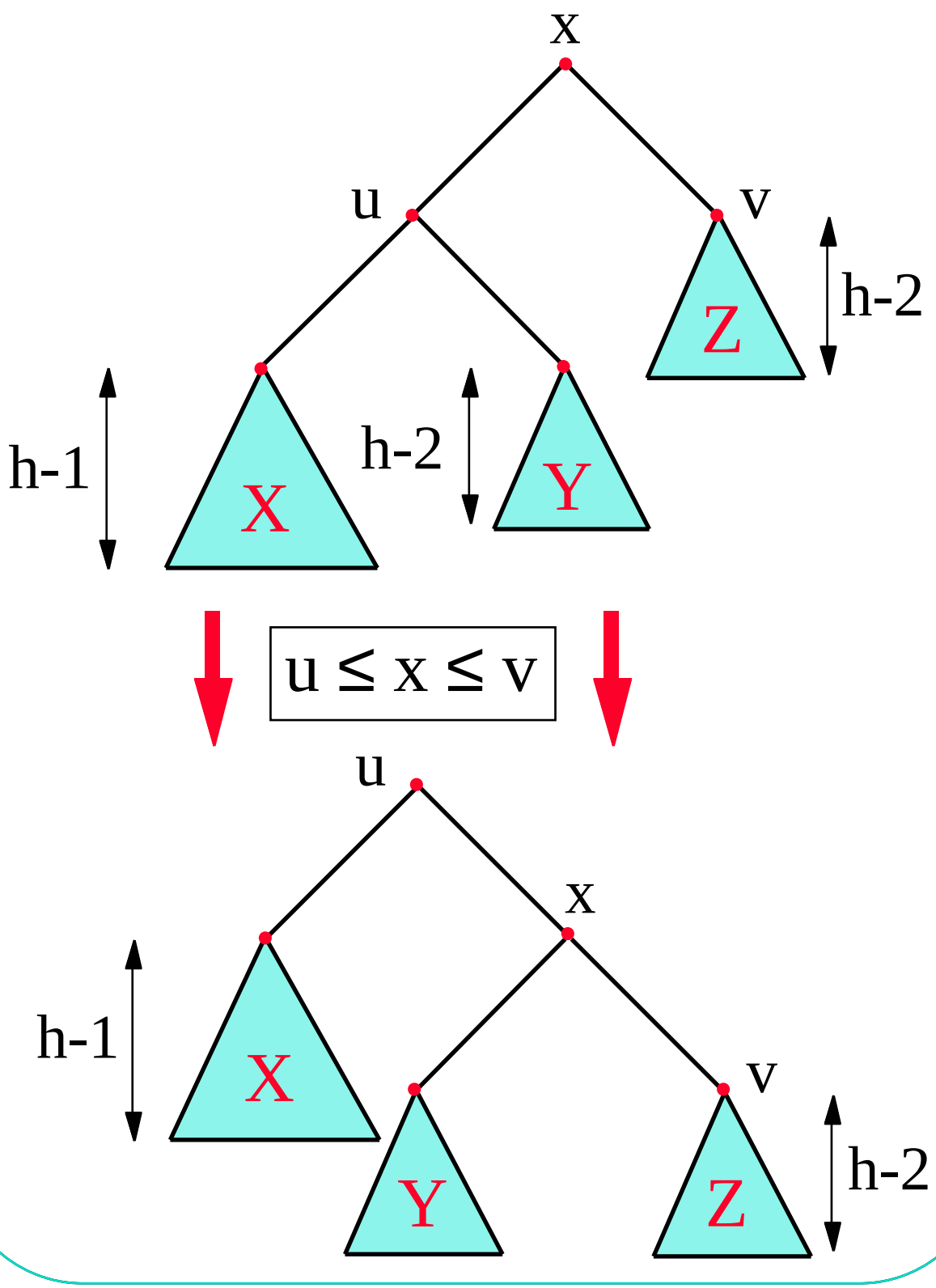
Donc $F(h) = N(h) + 1$ vérifie

$$F(0) = 2 \qquad F(1) = 3$$
$$F(h) = F(h-1) + F(h-2)$$

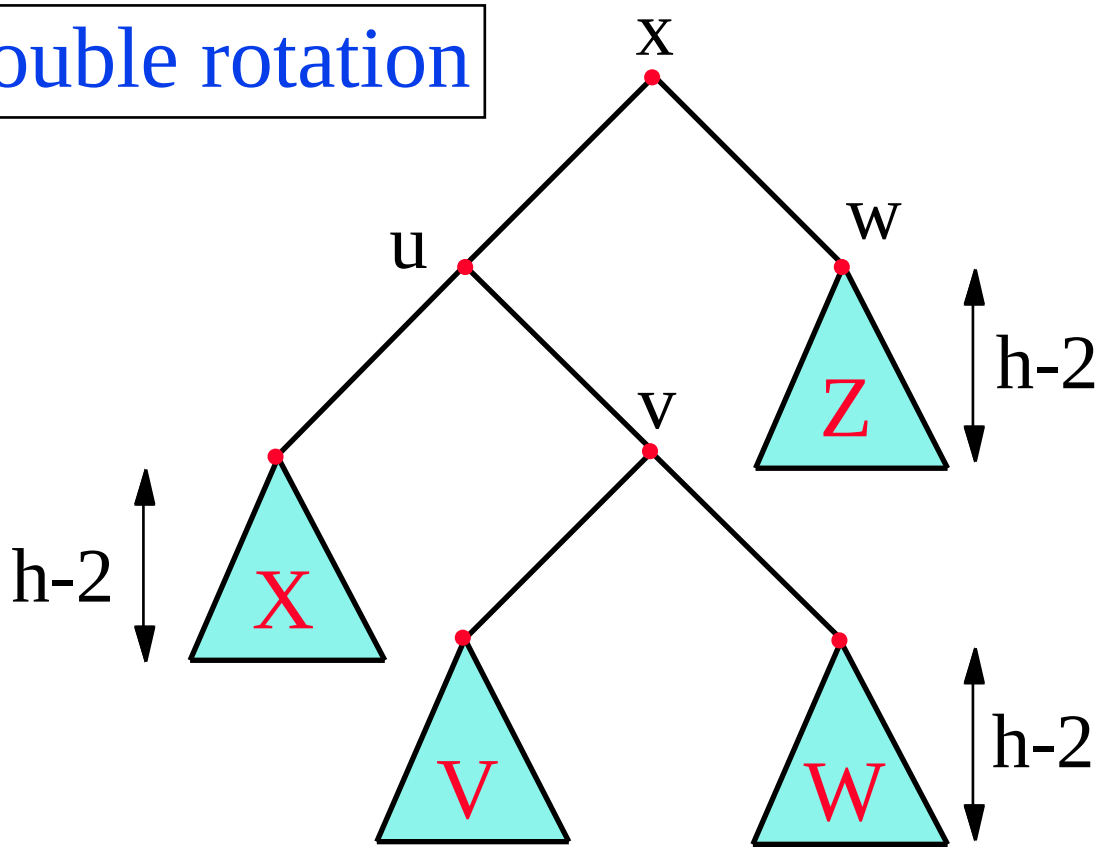
d'où $F(h) = F_{h+3} =$

$$\frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{h+3} - \left(\frac{1-\sqrt{5}}{2}\right)^{h+3}\right]$$

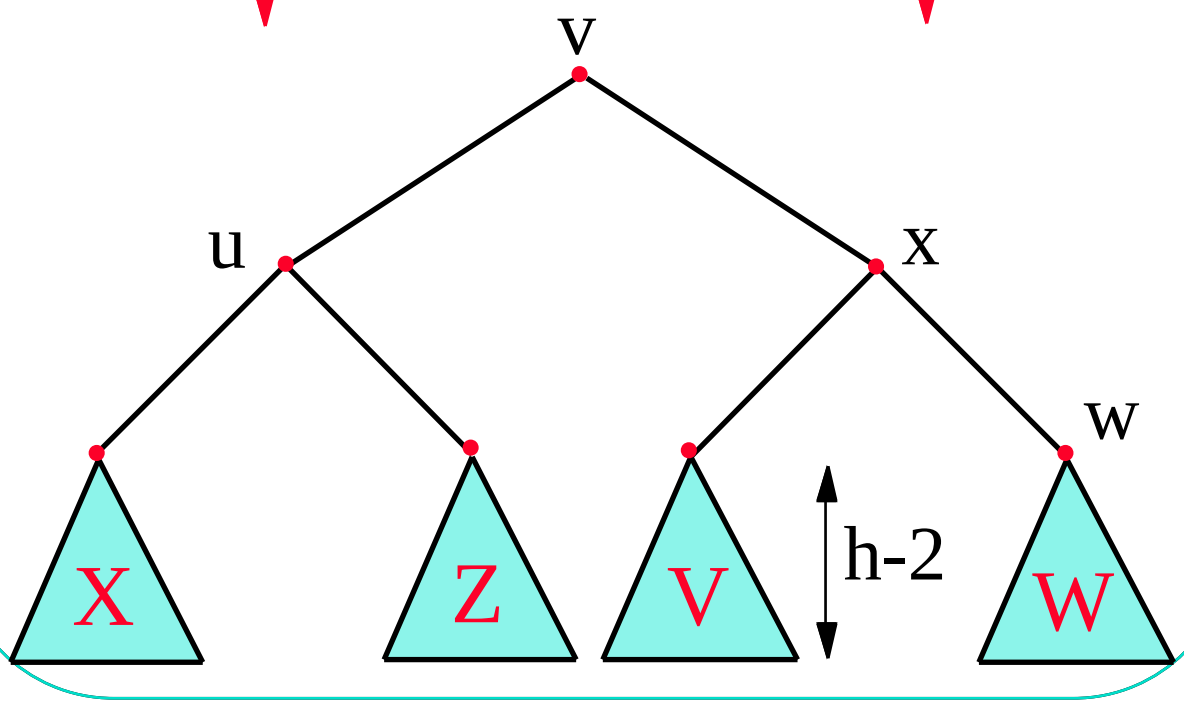
Rotation simple



Double rotation



$u \leq v \leq x \leq w$



Partitions

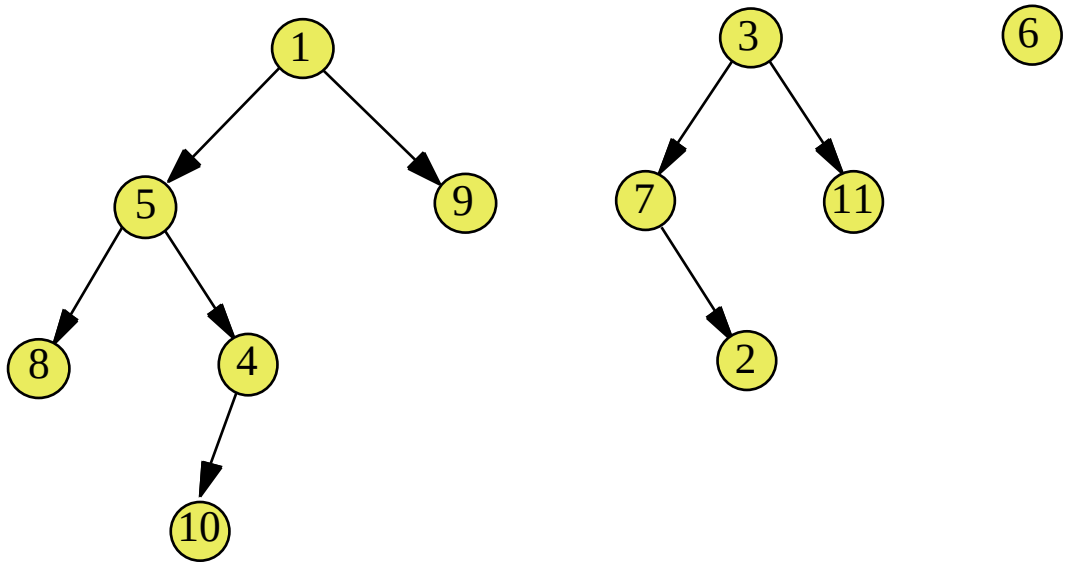
Données : une partition de l'ensemble {1, ..., K}

- Trouver la classe d'un élément
- Faire l'union de deux classes.

Une première solution : Représenter la partition par un tableau classe tel que classe[i] soit la classe de l'élément i.

Trouver : O(1)
Union : O(n)

Deuxième solution : utilisation d'une forêt.

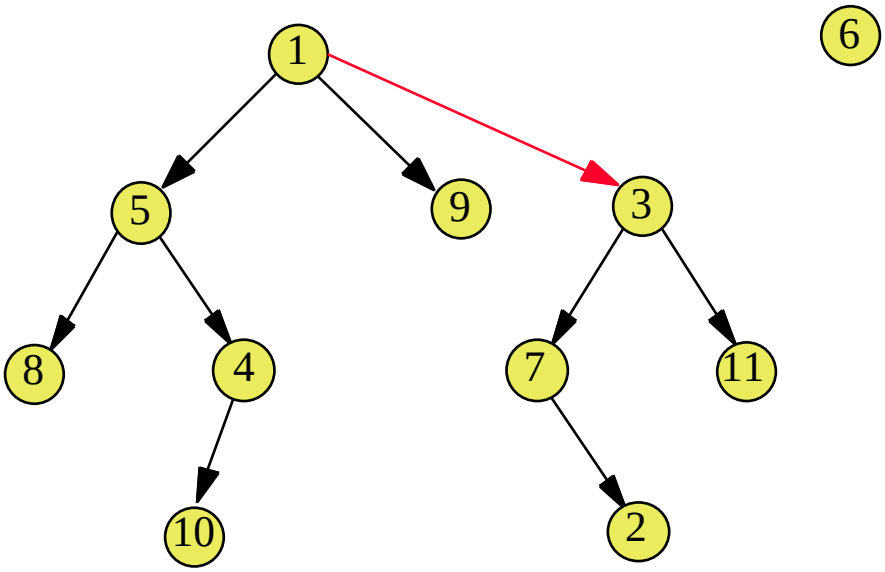


On représente la forêt par un tableau t tel que t[s] = père de s (si s est une racine, t[s] = s)

Trouver : O(n)
Union : proportionnel à la hauteur de l'arborescence

Union pondérée

Règle : Lors de l'union, la racine de l'arbre le moins haut devient fils de la racine de l'arbre le plus haut.



On part de la partition de {1, ..., K} en K classes.

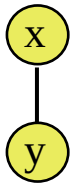
Notons $t_i(x)$ la taille (= nombre de noeuds) et $h_i(x)$ la hauteur du sommet x après la i-ème opération d'union pondérée. On a $t_0(x) = 1$ et $h_0(x) = 0$.

Lemme. On a $t_i(x) \geq 2^{h_i(x)}$ et $h_n(x) \leq \log_2(n + 1)$.

Preuve.
(a) Si $h_i(x) = 0$, alors $t_i(x) = 1$

(b) à suivre …

Lemme. On a $t_i(x) \geq 2^{h_i(x)}$ et $h_n(x) \leq \log_2(n+1)$.



Preuve (suite).

(b) Si $h_i(x) > 0$, soit y un fils de x. On a $h_i(y) = h_i(x) - 1$. Si y est devenu un fils de x lors de la j-ième union ($j \leq i$), on a $t_{j-1}(x) \geq t_{j-1}(y)$, d'où $t_j(y) = t_{j-1}(y)$ et $t_j(x) \geq 2t_j(y)$. Ensuite, la taille de y ne varie plus, mais celle de x peut croître. De même, la hauteur de y ne varie plus, donc $h_i(y) = h_{j-1}(y)$. Par hypothèse de récurrence, $t_{j-1}(y) \geq 2^{h_{j-1}(y)}$, donc $t_j(y) = t_{j-1}(y) \geq 2^{h_{j-1}(y)} = 2^{h_i(y)}$ et

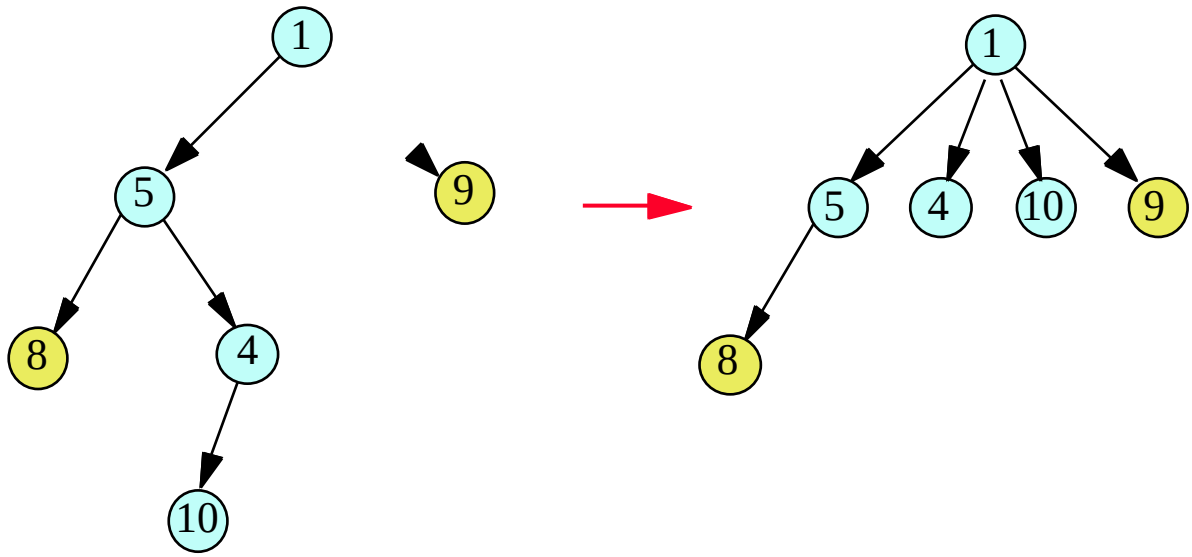
$$t_j(x) \geq t_j(x) \geq 2.t_j(y) \geq 2.2^{h_i(y)} = 2^{h_i(x)}$$

Comme $t_n(x) \leq n+1$, on a la seconde inégalité.

Corollaire. Une suite de $n-1$ "unions" et de m "trouver" se réalise en temps $O(n + m \log n)$.

Compression des chemins

On comprime un chemin en faisant de chaque nœud traversé un fils de la racine :



Prop. Une suite de n-1 "unions" et de m "trouver" ($m \geq n$) se réalise en temps $O(m.\alpha(n,m))$, où α est une sorte d'inverse de la fonction d'Ackermann. En pratique, $\alpha(n,m) \leq 2$.