NF16 - Parcours d'arbre itératif

```
Parcours_it(x : Noeud ; type : entier)
    N := 1                          Type 1 : Préfixe
    P := creerPile()                Type 2 : Infixe
    Répéter                         Type 3 : Postfixe
        Si N = 1
            Tant que x ≠ Nil
                Si type = 1
                    afficher(x)
                empiler(P, (x, 1))
                x := SAG(x)
        Si P ≠ ∅
            (x, N) := dépiler(P)
            Si N = 1
                Si type = 2
                    afficher(x)
                empiler(P, (x, 2))
                x := SAD(x)
            Si N = 2
                Si type = 3
                    afficher(x)
```

Chaque sommet rencontré 3 fois :

- Lors de la descente, après son père et avant son fils gauche
- Lors de la remontée par la gauche : après avoir dépilé le fils gauche et avant d'empiler le fils droit
- Lors de la remontée par la droite : après avoir dépilé le fils droit et avant de dépiler le père

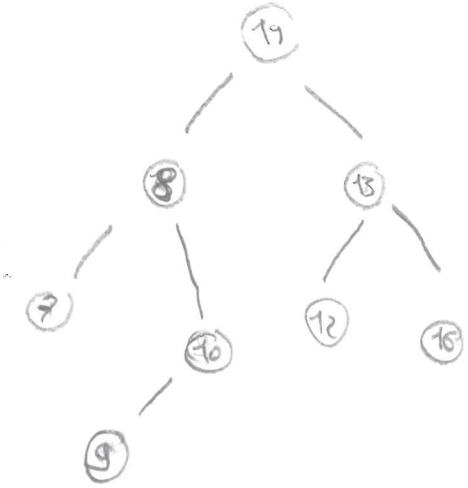
NF16 - Parcours d'arbre itératif

```
Parcours_it (x : Noeud ; type : entier)
    N := 1                      Type 1 : Préfixe
    P := creerPile()            Type 2 : Infixe
    Répéter                     Type 3 : Postfixe
        Si N = 1
            Tant que x ≠ Nil
                Si type = 1
                    afficher(x)
                empiler(P, (x,1))
                x := SAG(x)
        Si P ≠ ∅
            (x, N) := dépiler(P)
            Si N = 1
                Si type = 2
                    afficher(x)
                empiler(P, (x,2))
                x := SAD(x)
            Si N = 2
                Si type = 3
                    afficher(x)
```

Chaque sommet rencontré 3 fois :

- Lors de la descente, après son père et avant son fils gauche
- Lors de la remontée par la gauche : après avoir dépilé le fils gauche et avant d'empiler le fils droit
- Lors de la remontée par la droite : après avoir dépilé le fils droit et avant de dépiler le père

NF16-TD9 - Prefixe



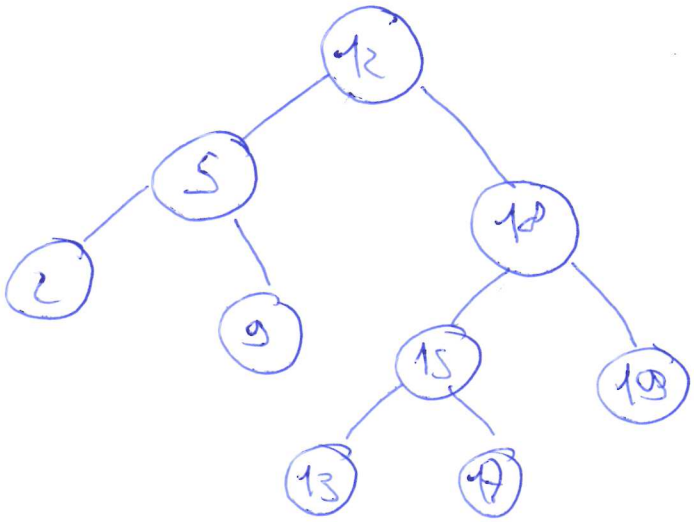

```
Prefixe(x)
  P: Pile
  Si x ≠ nil
    empiler(P, x)
  FinSi
  Tantque (pile_vide(P) = faux)
    x := depiler(P)
    parcourir(x)
    Si fils_droit(x) ≠ nil
      empiler(P, fils_droit(x))
    FinSi
    Si fils_gauche(x) ≠ nil
      empiler(P, fils_gauche(x))
    FinSi
  Fin Tantque
Fin Prefixe
```

NF16 – Parcours d'arbre récursif

```
Prefixe(x)
    Si x ≠ Nil
        afficher(x)
        prefixe(SAG(x))
        prefixe(SAD(x))
```

```
Postfixe(x)
    Si x ≠ Nil
        postfixe(SAG(x))
        postfixe(SAD(x))
        afficher(x)
```

```
Infixe(x)
    Si x ≠ Nil
        infixe(SAG(x))
        afficher(x)
        infixe(SAD(x))
```



Prefixe : 12 5 2 9 18 15 13 17 19

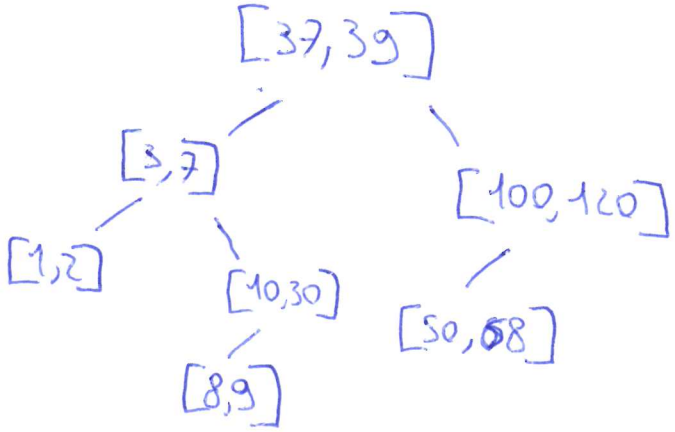
Postfixe : 2 9 5 13 17 15 19 18 12

Infixe : 2 5 9 12 13 15 17 18 19

NF16-TD8 - Exercice 3 - 1 - A)

Représentation sous la forme d'une ABR :

- Un noeud x de l'arbre A correspond à chaque intervalle I_x de réservation de la salle
- Un intervalle I_x est défini par ses valeurs $\min(I_x)$ et $\max(I_x)$. On choisit $\min(I_x)$ comme clé du noeud x.
- Si x' est un noeud du sous-arbre gauche $A_g(x)$ de x alors $\max(I_{x'}) < \min(I_x)$
- Si x' est un noeud du sous-arbre droit $A_d(x)$ de x alors $\min(I_{x'}) > \max(I_x)$



Suppression de [3,7] : On le remplace

- Soit par le min du sous-arbre droit enraciné en [3,7] → [8,9]
- Soit par le max du sous-arbre gauche enraciné en [3,7] → [1,2]

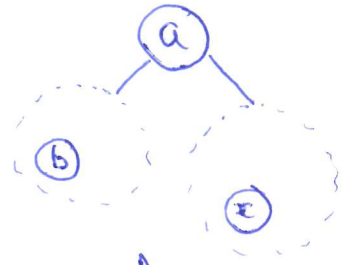
Arbre binaire de recherche - Rappels - Successeur / Predecesseur

Cas 1 : Le noeud n'a pas de fils droit (gauche)

- Fils gauche (droit) contient des éléments plus petits (grands) → pas le successeur (pred)
- Il faut remonter dans l'arbre pour trouver le successeur.

Deux cas : 1) Soit a un noeud de l'ABR tq $x \in SAD[a]$

Soit b un noeud de l'ABR tq $b \in SAG[a]$

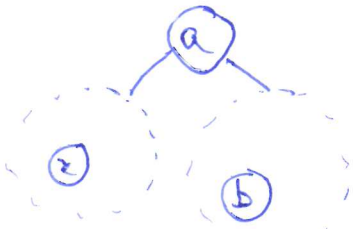


$cle[x] > cle[a] > cle[b]$ donc ni a, ni b ne sont le successeur de x.

a peut être le pred mais pas b

2) Soit a un noeud de l'ABR tq $x \in SAG[a]$

$b \in SAD[a]$



$cle[x] < cle[a] < cle[b]$ donc a peut être le successeur mais pas b

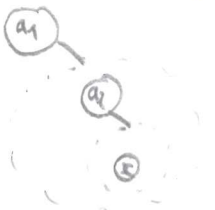
ni a ni b ne peuvent être le pred

↪ Le successeur est un noeud a de l'arbre tq $x \in SAG[a]$.

pred est un noeud

Parmi tq $x \in SAD[a]$

Soit a_1 et a_2 deux noeuds tq $x \in SAG_D[a_1]$ et $x \in SAG_D[a_2]$. On note a_1 le noeud le moins profond entre a_1 et a_2



$cle[a_1] > cle[a_2] > cle[x]$

$cle[a_1] < cle[a_2] < cle[x]$

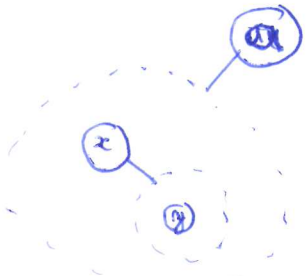
Pour trouver le successeur, on remonte l'arbre jusqu'à trouver le premier noeud ~~qui est fils gauche de son noeud parent~~.

tq $x \in SAG[a]$

$SAD[a]$

Cas 2 : Le noeud a un fils droit (gauche)

Soit y le noeud $\in SAD[x]$ de clé min



~~$cle[a] > cle[y]$~~

$cle[a] > cle[y] > cle[x]$.

$cle[a] < cle[y] < cle[x]$

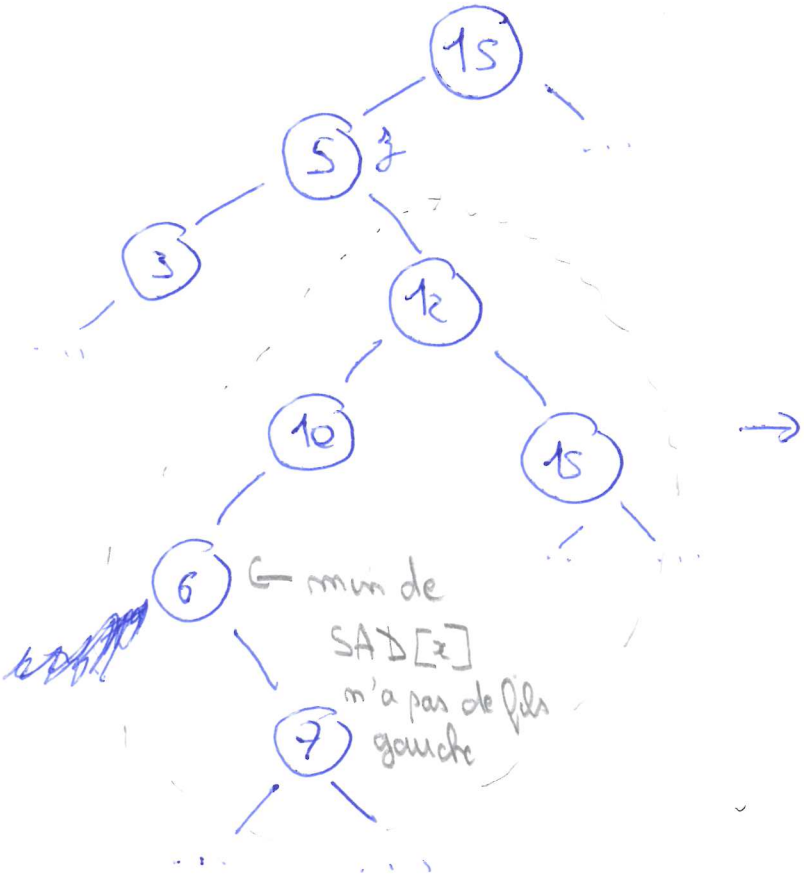
Plus petite valeur du SAD Plus grande du SAG

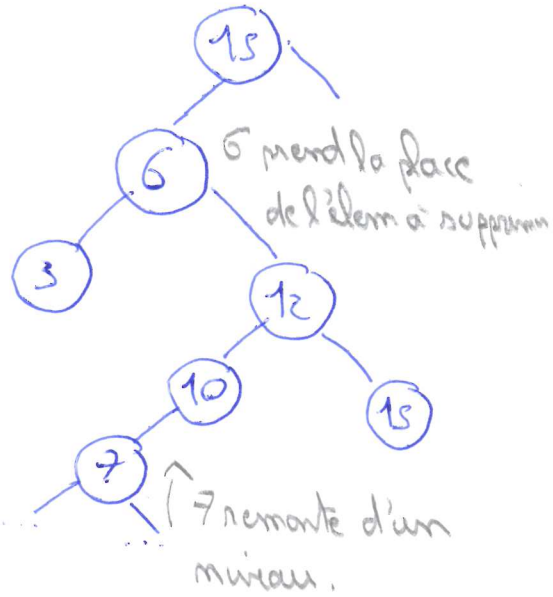
Arbre binaire de recherche : Rappels - Suppression

Cas 1 : Le noeud x à supprimer n'a pas de fils → RAS

Cas général : Le noeud x est racine d'un sous-arbre non vide. On a 2 choix

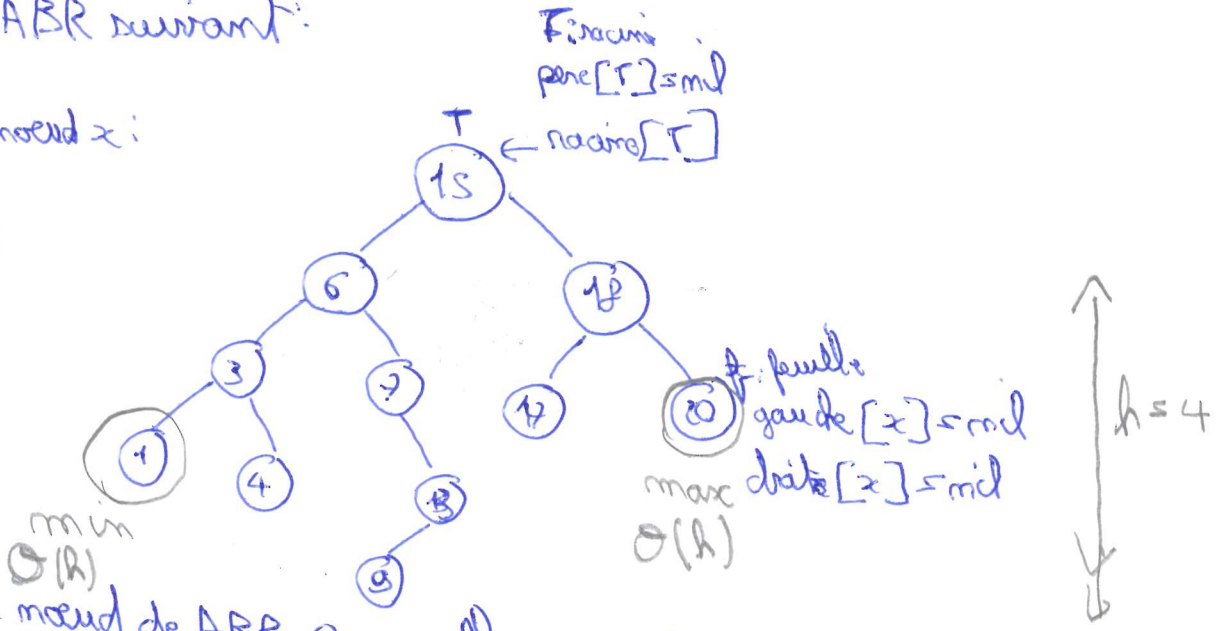
- Le remplacer par le min de SAD[x]
- le max de SAG[x]





Arbre binaire de recherche : Rappels - Recherche - Min - Max - Insertion

Soit T l'ABR suivant :



Attribut d'un noeud x :
- père[x]
- gauche[x]
- droit[x]
- clé[x]

Soit x un noeud de ABR. On appelle sous-arbre gauche (resp sous-arbre droit) de x le sous-arbre dont la racine est le fils gauche (resp droit) de x.

Si $y \in SAG[x]$ alors $clé[y] < clé[x]$

Si $y \in SAD[x]$ alors $clé[y] > clé[x]$

Recherche / Insertion (T, z).

- Recherche de 13
- Recherche de 19 $O(h)$
- Insertion de 8 $O(h)$

Mq l'algo renvoie "vrai" ssi I intersecte un noeud de A

$\Rightarrow$ Mq si intersecte(I, A) = vrai alors I intersecte l'intervalle associé à un noeud de A.

A chaque appel de "intersecte", x correspond au sommet racine d'un sous-arbre de A. I_x correspond à l'intervalle associé à un noeud de A.

L'algorithme renvoie vrai si $x \neq nil$ et $max(I_x) \geq min(I)$ et $min(I_x) \leq max(I)$ → L'intervalle I_x associé à x, noeud de l'arbre A intersecte I.

$\Leftarrow$ Mq si I intersecte l'intervalle associé à un noeud de A alors intersecte(I, A) = vrai

Démonstration par récurrence : Pour un arbre A de hauteur 0 :

Si I intersecte l'intervalle associé à un noeud de A, alors il s'agit du noeud racine et l'algorithme renvoie directement vrai.

Hérédité : Supposons la propriété vraie pour tout arbre de hauteur k ~~contenant~~ intersectant I. Mq la propriété est vraie pour tout arbre de hauteur k+1 ... un noeud.

Soit x_{k+1} le sommet racine.

* Si $I_{x_{k+1}}$ intersecte I, la propriété est vraie

* Si $min(I) > max(I_{x_{k+1}})$: Si I intersecte l'intervalle associé à un sommet de l'arbre de racine x_{k+1} alors ce sommet appartient à $SAD(x_{k+1})$ de hauteur k. L'algo effectue un appel recursif à intersecte en parcourant le paramètre $\neq SAD$, qui correspond donc à un arbre de hauteur k intersectant I. ⇒ Renvoie vrai par hypothèse de récurrence

* Si $max(I) < min(I_x)$: //

NF16-TD8-Exercice 3-2

B) intersecte (I : intervalle ; A : ABR) : booleen

x : noeud := racine[A]

Si x = nil

retourner faux

Sinon si min(I) > max(I_x)

retourner intersecte (I, A.SAD)

Sinon si max(I) < min(I_x)

retourner intersecte (I, A.SAG)

Sinon

retourner vrai

FinSi

Fin intersecte

Terminaison + Complexité

L'algo est initialement appelé avec l'arbre fourni depuis sa racine. A chaque appel :
- Soit intersection trouvée
- Soit le noeud racine est nulle et l'algo s'arrête
- Soit appelle recursif en descendant de 1 niveau de profondeur dans l'arbre.

Pour un arbre de hauteur h, après h appels (dans le pire des cas), le parametre A passé à la fonction correspond au fils d'une feuille de l'arbre → NIL.

L'algo se termine donc et est de complexité $O(h)$

NF16 - TD8 - Exercice 2

```
succ (A: ABR, a: entier) : entier
    x : Noeud := racine[A]
    Si (x = NIL)
        retourner + Infini
    Sinon si (cle[x] = a)
        retourner a
    Sinon si (cle[x] < a)
        retourner succ (A.SAD, a)
    Sinon
        retourner inf (cle[x], succ (A.SAG, a))
```

Soit :

- inf(a: entier, b: entier) fonction qui renvoie le min entre a et b
- A.SAD (resp SAG) le sous arbre de racine le fils droit (resp gauche de A)

valeurs plus petites que cle[x] mais peuvent toutes être plus petites que a // dans ce cas là est le noeud de cle succ[a]

Mq l'algo renvoie vrai ssi I intersecte un noeud de A

$\Rightarrow$ Mq si intersecte $(I, A) =$ vrai alors I intersecte l'intervalle associé à un noeud de A. Soit x le noeud racine.

L'algo renvoie vrai lorsque

- $a \neq$ nil et $\max(I_x) \geq \min(I)$ et $\min(I_x) \leq \max(I)$
  ↪ L'intervalle associé à la tête intersecte I.
- ~~(Un appel récursif, avec un sous-arbre de A en paramètre renvoie vrai~~ (x, noeud de l'arbre A)
  ~~↪ L'intervalle associé à un noeud × tête intersecte I.)~~

~~Dans les 2 cas où l'algo renvoie vrai~~, I intersecte un noeud de l'arbre A.

$\Leftarrow$ Mq si I intersecte l'intervalle d'un noeud de A alors intersecte $(I, A) =$ vrai.

Soit $[x_0, x_1, \dots, x_p]$ le chemin depuis x_0, racine de A jusqu'au noeud x_p qui intersecte I le moins profond.

A_i arbre de racine x_i, $i \in [\![0; p]\!]$

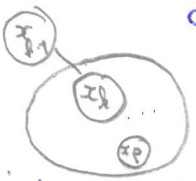
Mq $\forall i \in [\![0; p]\!]$, intersecte $(I, A_i) =$ vrai

Pour A_p : intersecte $(I, A_p) =$ vrai (I_{x_p} intersecte I)

Hérédité : Supposons la propriété vraie au rang $k \in [\![1; p]\!]$. Mq la propriété est vraie au rang $k-1$.

Le noeud x_{k-1} racine de A_{k-1} est moins profond que x_p. Donc par définition de x_p, x_{k-1} n'intersecte pas I. Donc (1) $\max(I) < \min(I_{x_{k-1}})$ ou (2) $\min(I) > \max(I_{x_{k-1}})$

Si x_k fils droit de x_{k-1}



Le sous-arbre de racine x_k contient x_p, qui intersecte avec I

~~$\max(I) > \min(I_{x_p})$~~

$\min(I_{x_p}) > \max(I_{x_{k-1}})$ // $x_p \in$ Sous arbre droit de A_{k-1}

+ (3) $\Rightarrow \max(I) > \max(I_{x_{k-1}})$ ~~(1)~~ fausse donc (2) vrai

↪ intersecte (I, A_{k-1}) = intersecte (I, A_k) = vrai

(A_{k-1} SAD) (hyp. de rec)

I_{x_p} intersecte I donc
(3) $\max(I) > \min(I_{x_p})$
et (4) $\min(I) < \max(I_{x_p})$

NF16 - TD8 - Exercice 3 - C)

profondeur de l'arbre
h : ~~hauteur~~

	Liste quelconque	Liste triée	ABR
Insérer	$O(1)$	$O(n)$	$O(h)$
Supprimer	$O(n)$	$O(n)$	$O(h)$
Rechercher	$O(n)$	$O(n)$	$O(h)$
Intersection	$O(n)$	$O(n)$	$O(h)$

Arbre binaire complet ~~d'ordre~~ 2 : Toutes les feuilles ont la même profondeur et tous les nœuds internes ont pour degré 2.

$2^{h+1} - 1$ nœuds

$n = 2^{h+1} - 1$

$n+1 = 2^{h+1}$

$h+1 = \log_2(n+1)$

$h = \log_2(n+1) - 1$