Algorithmique

ABR : Ajouts et suppression

1 Insertions

Deux méthodes d'insertion d'un élément dans un arbre binaire de recherche (ABR) :

- L'insertion en feuille qui consiste à ajouter le nouvel élément en tant que nouvelle feuille de l'arbre ;
- l'insertion en racine qui consiste à faire du nouvel élément la nouvelle racine de l'arbre, qui sera "coupé" pour produire les deux sous-arbres du résultat.

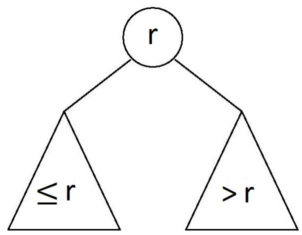
1.1 Insertion en feuille

Insérer l'élément x en feuille dans l'arbre binaire de recherche B :

- Si l'arbre est vide, on crée à la place une nouvelle feuille contenant x.
- Si l'arbre n'est pas vide, on compare x à l'élément contenu dans la racine :
  - s'il est inférieur ou égal, on ajoute x en feuille au sous-arbre gauche de B (l'arbre résultat a donc la même racine et le même sous-arbre droit),
  - s'il est supérieur, on ajoute x en feuille au sous-arbre droit de B.

Version fonction : elle retourne l'arbre résultat de l'insertion

```
fonction ajoutFeuille(élément x, ABR B): ABR
variables
   noeud r
debut
  si B = arbrevide alors
     contenu(r) ← x
     retourne <r, arbrevide, arbrevide>
  sinon
     si x <= contenu(racine(B)) alors
          retourne <racine(B), ajoutFeuille(x, g(B)), d(B)>
     sinon
          retourne <racine(B), g(B), ajoutFeuille(x, d(B))>
     fin si
  fin si
fin
```



Version procédure : l'arbre est modifié

```
procedure ajoutFeuille(élément x, ref ABR B)
variables
   noeud r
debut
  si B = arbrevide alors
     contenu(r) ← x
     B ← <r, arbrevide, arbrevide>
  sinon
     si x <= contenu(racine(B)) alors
          g(B) ← ajoutFeuille(x, g(B))
     sinon
          d(B) ← ajoutFeuille(x, d(B))
     fin si
  fin si
fin
```

1.2 Insertion en racine

La coupure de l'arbre B selon x consiste à construire les deux sous-arbres G et D contenant respectivement les éléments inférieurs ou identiques à x et les éléments supérieurs à x :

- Si B est vide, G et D sont deux arbres vides ;
- Sinon, on compare r le contenu de la racine de B à x :
  - si $r \leq x$, G est l'arbre de racine r et de sous-arbre gauche $g(B)$.
    Le résultat de la coupure de $d(B)$ donnera le sous-arbre droit de G et D ;
  - si $r > x$, D est l'arbre de racine r et de sous-arbre droit $d(B)$.
    Le résultat de la coupure de $g(B)$ donnera G et le sous arbre gauche de D ;

```
procedure couper(élément x, ABR B, ref ABR G, D)
debut
  si B = arbrevide alors
     G ← arbrevide
     D ← arbrevide
  sinon
     si contenu(racine(B)) <= x alors
          G ← B
          couper(x, d(B), d(B), D)
     sinon
          D ← B
          couper(x, g(B), G, g(D))
     fin si
  fin si
fin
```

Insérer l'élément x en racine dans l'arbre binaire de recherche B : une nouvelle racine contenant x. Ces deux sous-arbres sont le résultat de la coupure de B selon x.

Version fonction : elle retourne le nouvel arbre

```
fonction ajoutRacine(élément x, ABR B): ABR
variables
   noeud r
   ABR G, D
debut
     contenu(r) ← x
     couper(x, B, G, D)
     retourne <r, G, D>
fin
```

Version procédure : On peut également directement passer à la coupure les deux sous-arbres à compléter.

```
procedure ajoutRacine(élément x, ref ABR B)
variables
   ABR R
debut
     contenu(racine(R)) ← x
     couper(x, B, g(R), d(R))
     B ← R
fin
```

2 Suppression

La fonction qui supprime le maximum d'un ABR

```
fonction suppmax(ref ABR B) : élément   /* B non vide */
variables
   élément  m
debut
   si d(B) = arbrevide alors
      m ← contenu(racine(B))
      B ← g(B)
      retourne m
   sinon
      retourne suppmax(d(B))
   fin si
fin
```

L'algorithme de suppression

La suppression de l'élément x dans un ABR commence par la recherche de celui-ci. Si cette recherche est positive, x se trouve dans le nœud r, racine de B :

- r est une feuille, elle peut être détruite : B devient un arbre vide ;
- r est un point simple : le noeud peut être détruit, le fils unique remonte, B devient son sous-arbre ;
- r est un point double : le contenu de r est remplacé par la valeur maximale[1] du sous arbre gauche de B duquel elle est supprimée.

```
procedure supprimerABR(élément x, ref ABR B)
debut
   si B <> arbrevide alors
      si x < contenu(racine(B)) alors
         supprimerABR(x, g(B))
      sinon
         si x > contenu(racine(B)) alors
            supprimerABR(x, d(B))
         sinon
            si g(B) = arbrevide alors          /* racine(B) = point simple à droite ou feuille */
               B ← d(B)
            sinon
               si d(B) = arbrevide alors       /* racine(B) = point simple à gauche */
                  B ← g(B)
               sinon                           /* racine(B) = point double */
                  contenu(racine(B)) ← suppmax(g(B))
               fin si
            fin si
         fin si
      fin si
   fin si
fin
```

1. On peut également remplacer par le minimum du sous-arbre droit dans le cas d'arbres à valeurs toutes distinctes

3 Complexité

Pour mémoire voici le tableau récapitulant la complexité de la **recherche** d'une valeur dans un arbre binaire de recherche B :

	moyenne	pire
positive	$PM(B)$	$hauteur(B)$
négative	$PME(B)$	$hauteur(B)$

avec $PM(B)$: profondeur moyenne de B et $PME(B)$: profondeur moyenne externe de B.

La complexité de l'**insertion**, que ce soit en feuille ou en racine, est la même que celle de la recherche négative : la branche empruntée est la même que celle du chemin de recherche.

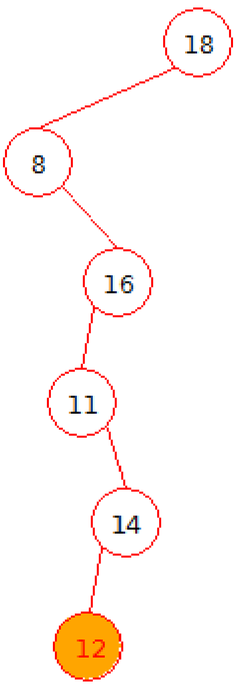
La complexité de la **suppression** est la même que celle de la recherche : le chemin de recherche de l'élément à supprimer (auquel on ajoute éventuellement le chemin pour supprimer le maximum du sous-arbre gauche en cas de suppression dans un point double).

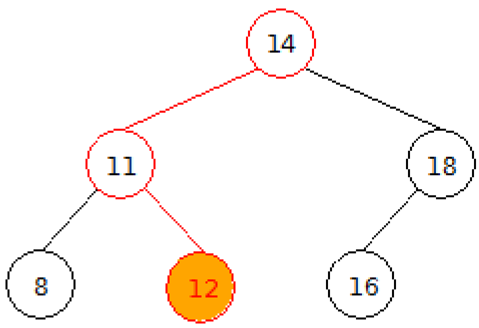
Soit, n la taille de l'arbre binaire de recherche B et C_n le nombre de comparaisons, au pire, de la recherche, l'ajout et la suppression :

- $C_n = \text{hauteur}(B)$
- $\lceil log_2(n) \rceil \leq C_n \leq n$

La complexité des algorithmes (recherche, ajouts, suppression) sur les arbres binaires de recherche dépend donc de la hauteur de celui-ci :

- Le meilleurs des cas : l'arbre est complet (ou presque, tous ses niveaux sont remplis sauf le dernier), la complexité est logarithmique.
- Le pire des cas : l'arbre est filiforme, la complexité est linéaire.





L'idéal serait donc de toujours avoir des ABR de hauteur réduite...