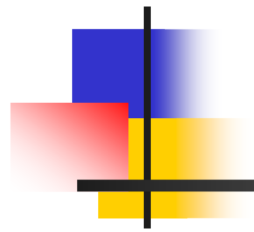
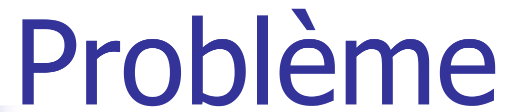


Algorithmique fonctionnelle

Arbres binaires de recherche



Problème

L’opération de recherche dans un arbre binaire est de complexité linaire en fonction du nombre de nœuds (car il faut les parcourir tous).

Nous ne gagnons rien par rapport à une liste.

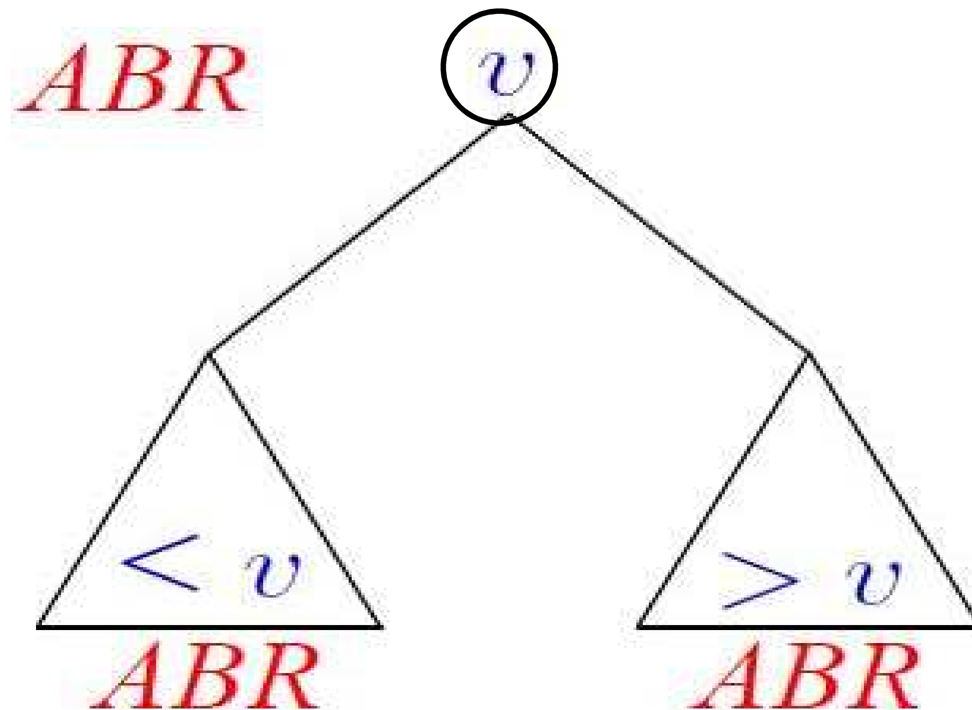
Est-ce qu’on peut accélérer
la recherche dans un arbre?

Définition

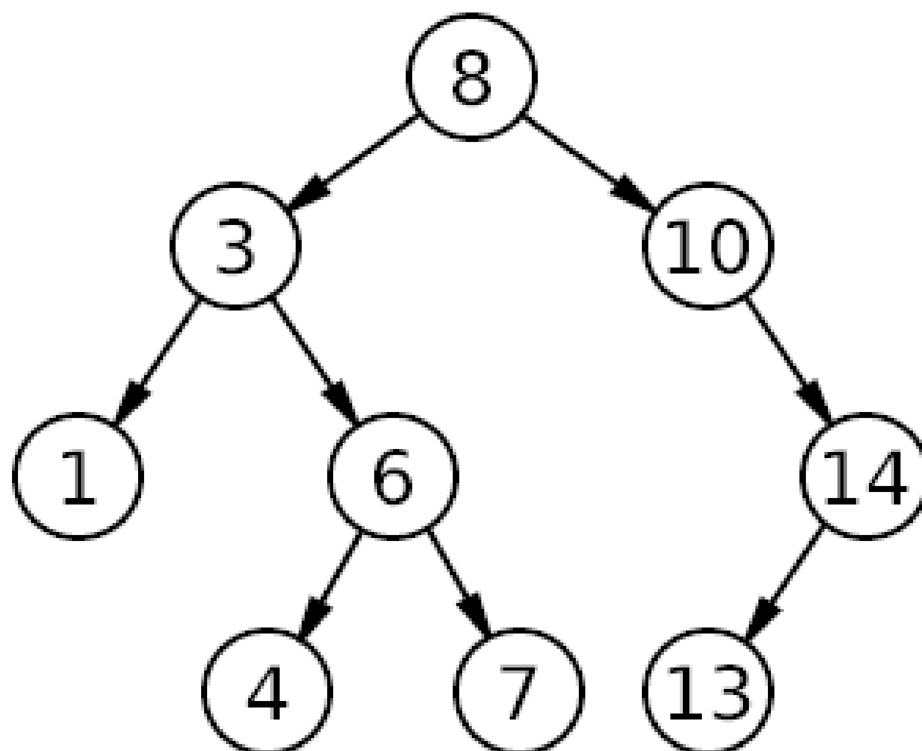
Un arbre binaire A est appelé un arbre binaire de recherche (ABR) si et seulement si

- soit A est vide.
- soit A contient au moins un nœud et
  - toute valeur associée à un nœud de son fils gauche est < racine(A)
  - toute valeur associée à un nœud de son fils droit est > racine(A)
  - tout sous-arbre de A est lui-même un ABR

Vision schématique d'un ABR



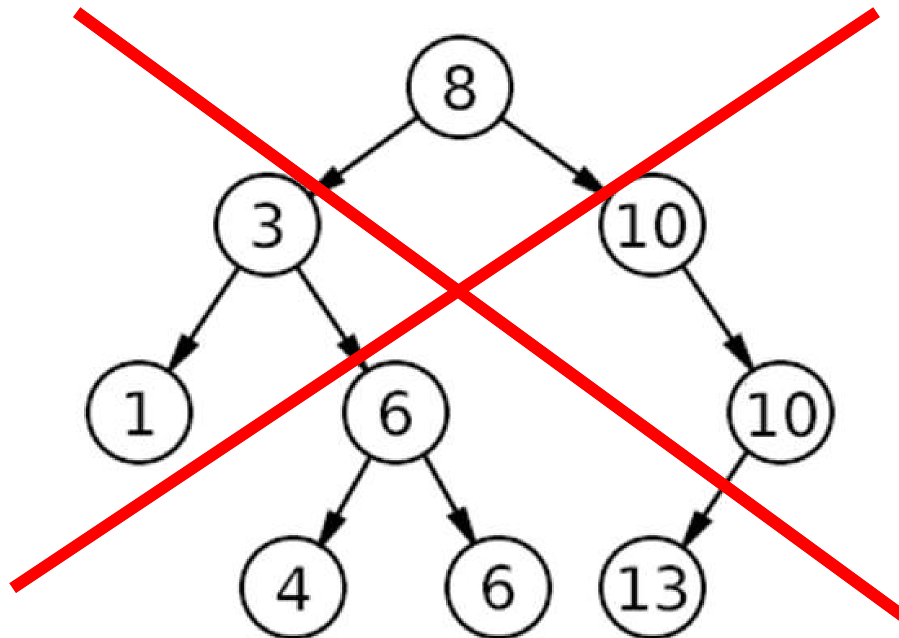
Vision concrète d’un ABR



Notez

Selon la mise en œuvre de l'ABR, on interdit ou non des nœuds contenant des éléments de valeur égale.

On travaillera dans notre cours avec des arbres à éléments uniques.



ABR en pseudo-code

- Nous définissons le type « ABR » pour lequel nous récupérons les fonctions d'un arbre binaire classique:
  - **estVide**: ABR -> Booléen
  - **cons**: T x ABR x ABR -> ABR
  - **racine, r**: ABR -> T
  - **filsGauche, fG** : ABR -> ABR
  - **filsDroit, fD** : ABR -> ABR

Opérations dans un ABR

- Recherche
- Insertion
- Suppression

Recherche

- Comparer l'élément cherché à la racine de l'arbre. Nous aurons alors deux cas :
  - s'il est inferieur, on fait une recherche dans le fils gauche.
  - s'il est supérieur, on fait une recherche dans le fils droit.
- La recherche stoppe
  - lorsque l'on trouve l'élément ou
  - lorsque l'arbre est vide, ce qui résulte dans un échec.

Exercice

Ecrire une fonction booléenne **contient** qui vérifie si un élément donné appartient à un ABR.

Exercice

Ecrire une fonction booléenne **contient** qui vérifie si un élément donné appartient à un ABR.

```
Fonction contient(x: T, A: ABR): Booléen
Début
        Si estVide(A) Alors
                Retourner faux
        SinonSi x=r(A)
                Retourner vrai
        SinonSi x>r(A)
                Retourner contient(x,fD(A))
        Sinon {x<r(A)}
                Retourner contient(x,fG(A))
        FinSi
Fin
```

Recherche: Complexité

Le nombre d'opérations de la recherche dépend de la **hauteur** de l'arbre.
Pour un ABR de n nœuds, on effectuera

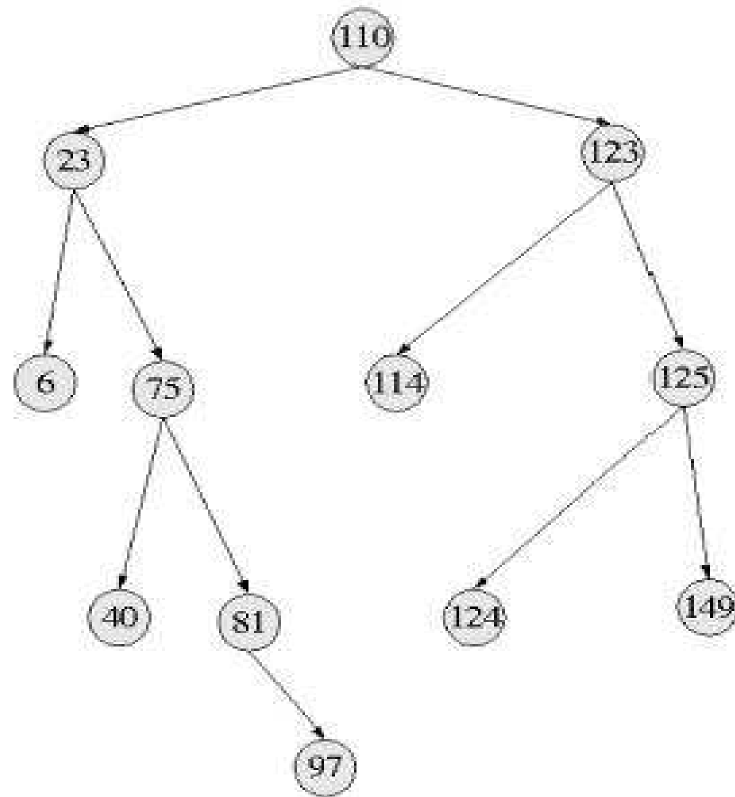
- Meilleur cas: O(log n) opérations si l'arbre est bien équilibré => mieux que la liste
- Pire cas: O(n) opérations si l'arbre est complètement dégénéré

Insertion

- Si l’arbre initial est vide, le résultat est formé d’un ABR réduit à sa racine, celle-ci contenant le nouvel élément.
- Si l’élément à ajouter est déjà dans l’arbre, l’hypothèse d’unicité des éléments fait qu’on ne réalise pas l’ajout.
- Sinon, il existe un seul endroit pour insérer l’élément sans détruire la propriété ABR. Pour le trouver, effectuer une recherche (récursive) jusqu’à tomber sur l’arbre vide puis ajouter l’élément.

Exercice

Dans l’arbre donné, insérer successivement les éléments 9, 121, 79, 114, 100, 80.



Insertion: Complexité

Tout comme pour la recherche, le nombre d'opérations dépend de la **hauteur** de l'arbre. Pour un ABR de n nœuds, on effectuera

- Meilleur cas: O(log n) opérations si l'arbre est bien équilibré => mieux que la liste
- Pire cas: O(n) opérations si l'arbre est complètement dégénéré

Exercice

Ecrire la fonction **insérer** qui insère un élément dans un ABR (la fonction ne fait rien si l'élément existe déjà).

Exercice

Ecrire la fonction **insérer** qui insère un élément dans un ABR (la fonction ne fait rien si l'élément existe déjà).

```
Fonction insérer(x: T, A: ABR): ABR
Début
  Si estVide(A) Alors
       Retourner cons(x,arbreVide, arbreVide)
  SinonSi x=r(A)
       Retourner A       // ne rien faire
  SinonSi x>r(A)
       Retourner cons(r(A),fG(A), insérer(x,fD(A)))
  Sinon {x<r(A)}
       Retourner cons(r(A),insérer(x,fG(A)), fD(A))
  FinSi
Fin
```

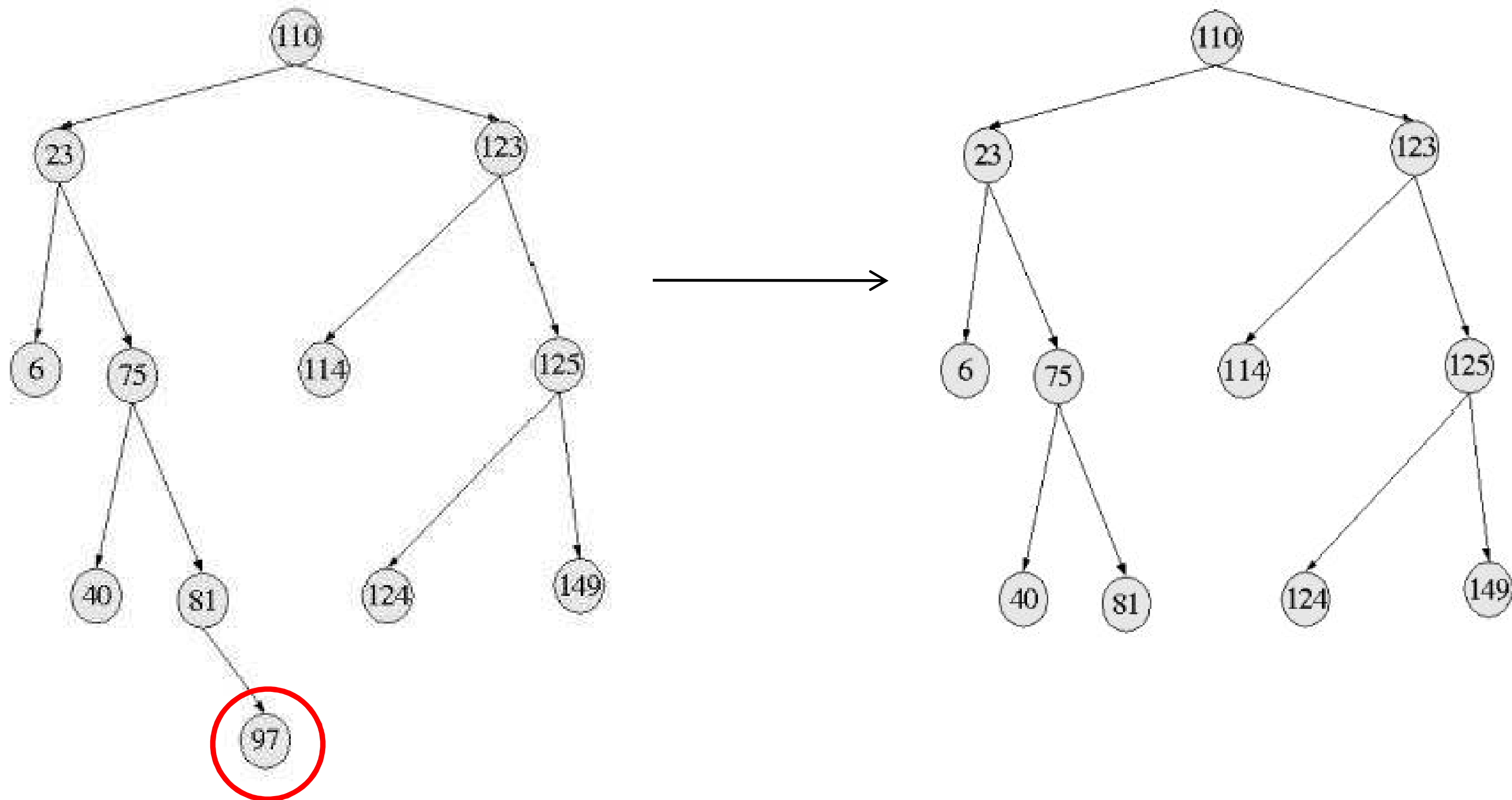
Suppression

- Plus complexe que la recherche et l'insertion
- Ne doit pas compromettre la relation d'ordre sur les clés des éléments restant dans l'arbre.

La suppression commence par une recherche de l'élément. Ensuite, plusieurs cas sont à considérer…

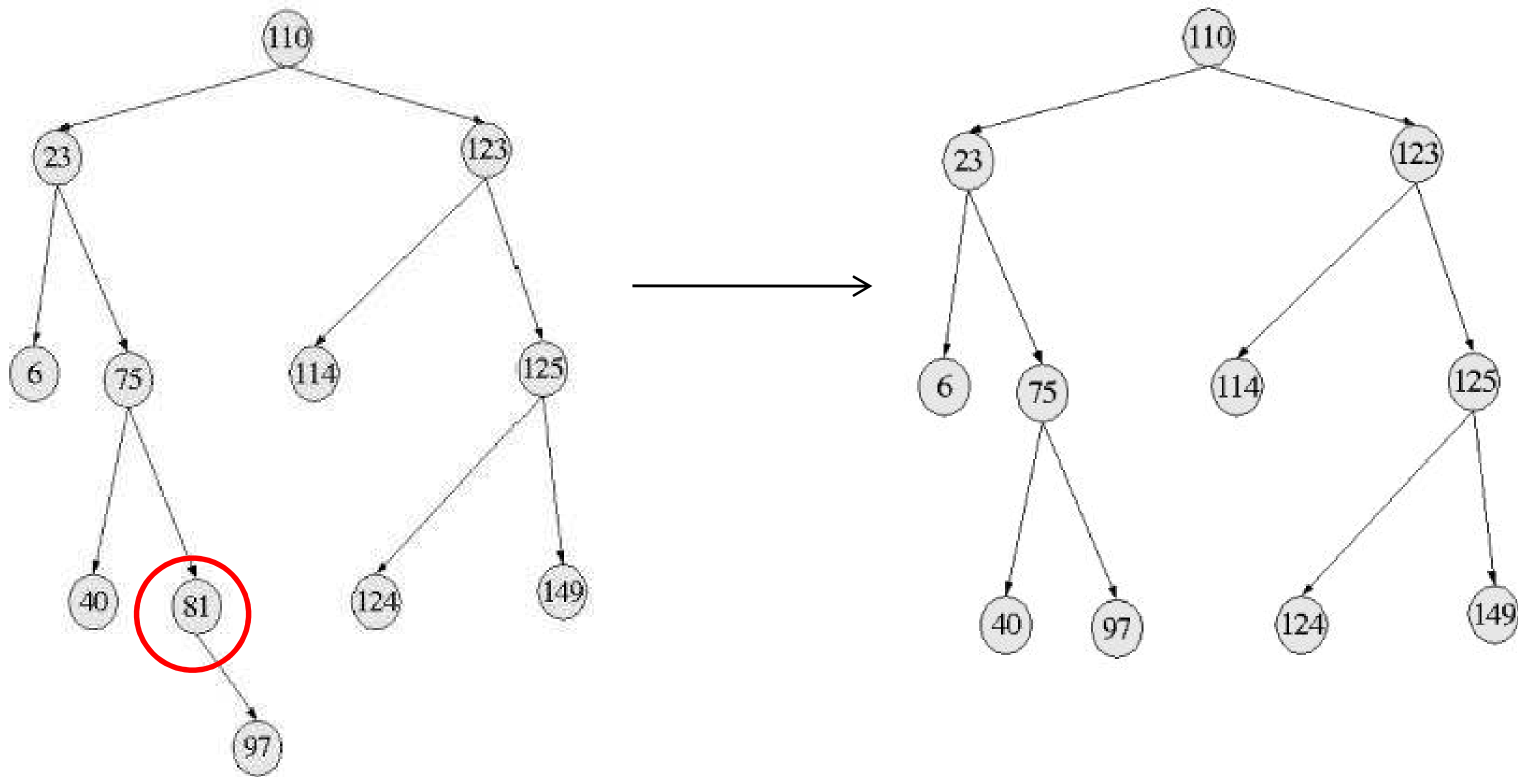
Suppression: pas de fils

Si l’élément à supprimer est situé sur une feuille de l’arbre: la suppression n’entraîne aucune réorganisation.



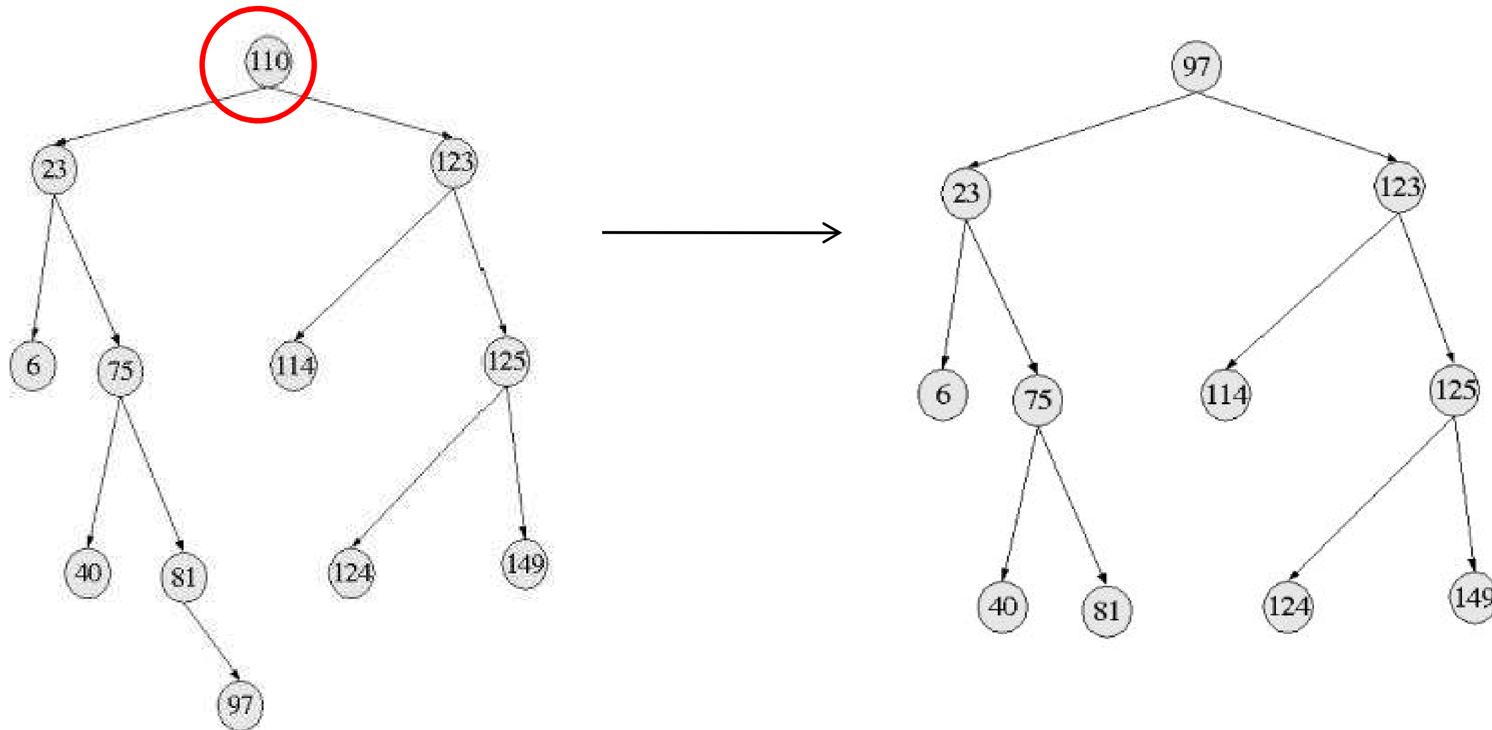
Suppression: un seul fils

Si l'élément à supprimer est sur un noeud possédant un seul fils: il suffit de remplacer le noeud en question par le fils unique.



Suppression: deux fils

On remplace le nœud à supprimer par son plus proche successeur ou son plus proche prédécesseur N. Puis on supprime N qui est forcément une feuille ou un nœud à un seul fils.



Suppression: Complexité

Une suppression comporte donc

- un parcours d’une unique branche
- un éventuel échange de valeur, puis
- la suppression d’un nœud possédant au maximum 1 successeur.

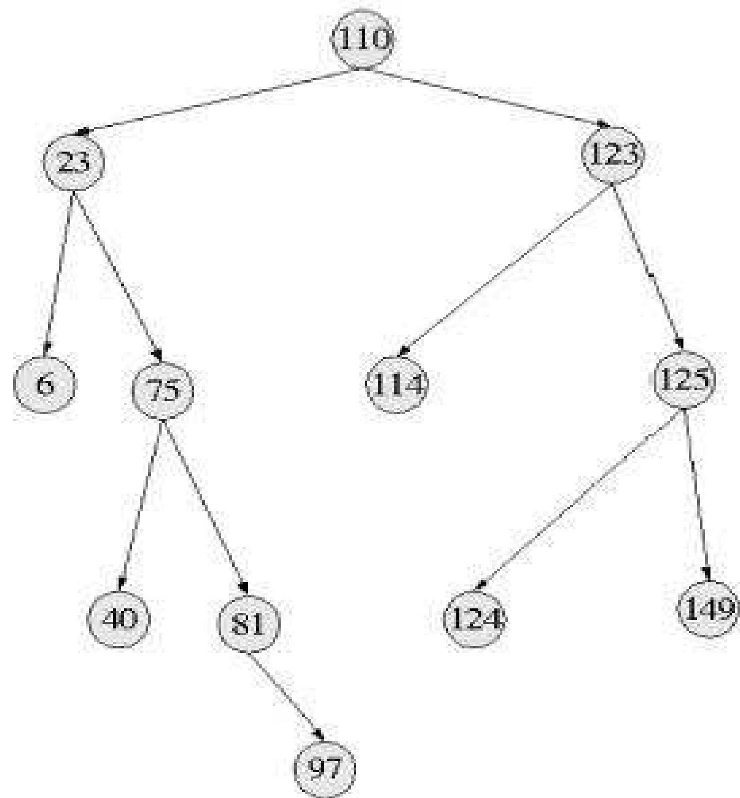
Les deux dernières opérations peuvent être faites en O(1).

Tout comme pour la recherche et l’insertion, on effectuera:

- Meilleur cas: O(log n) opérations si l’arbre est bien équilibré => mieux que la liste
- Pire cas: O(n) opérations si l’arbre est complètement dégénéré

Suppression

Exercice: Supprimer de l'arbre donnée successivement les éléments 6, 123, 114, 75, 110.



Suppression

Pour définir la fonction de suppression, on a besoin de deux fonctions auxiliaires:

1. une fonction **max** qui retourne l’élément de clé maximale dans un ABR;

2. une fonction **supprMax** qui retourne l’arbre privé de son plus grand élément.

Exercice

Ecrire une opération **max** qui retourne l'élément de clé maximale dans un ABR (on suppose que l'arbre n'est pas vide).

Exercice

Ecrire une opération **max** qui retourne l'élément de clé maximale dans un ABR (on suppose que l'arbre n'est pas vide).

```
Fonction max(A : ABR): ABR
Début
      Si estVide(fD(A)) Alors
            Retourner r(A)
      Sinon
            Retourner max(fD(A))
      FinSi
Fin
```

Exercice

Ecrire une opération **supprMax** qui retourne l’arbre privé de son plus grand élément (on suppose que l’arbre n’est pas vide).

Exercice

Ecrire une opération **supprMax** qui retourne l'arbre privé de son plus grand élément (on suppose que l'arbre n'est pas vide).

```
Fonction supprMax(A : ABR): ABR
Début
      Si estVide(fD(A)) Alors
            Retourner fG(A)
      Sinon
            Retourner cons(racine(A),fg(A),supprMax(fD(A))
      FinSi
Fin
```

Exercice

Ecrire une fonction **supprimer** qui supprime un élément donné de l’ABR (la fonction ne fait rien si l’élément n’existe pas).

Exercice

```
Fonction supprimer(x: T, A : ABR): ABR
Début
  Si estVide(A) Alors
    Retourner A
  SinonSi x>r(A)
    Retourner cons(r(A),fG(A),supprimer(x,fD(A)))
  SinonSi x<r(A)
    Retourner cons(r(A),supprimer(x,fG(A)),fD(A))
  SinonSi estVide(fG(A))
    Retourner fD(A)
  SinonSi estVide(fD(A))
    Retourner fG(A)
  Sinon
    Retourner cons(max(fG(A)),supprMax(fG(A)), fD(A))
  FinSi
Fin
```