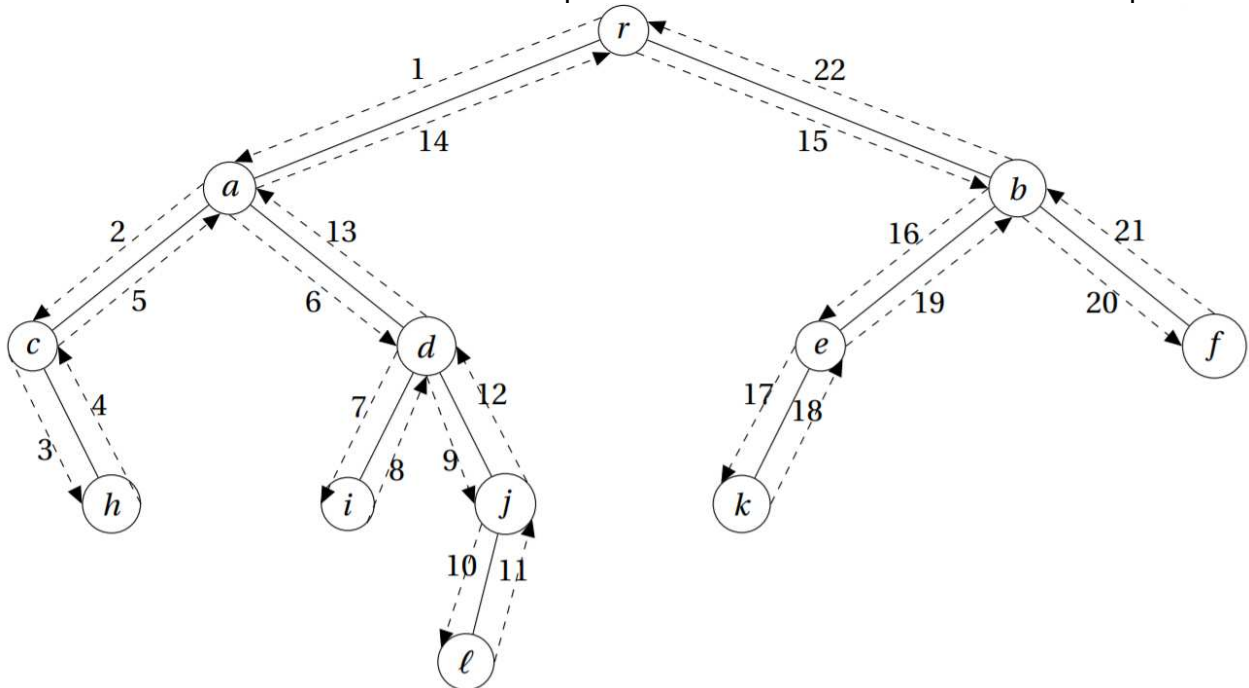


1 Parcours en profondeur

On se balade autour de l'arbre en suivant les pointillés dans l'ordre des numéros indiqués :

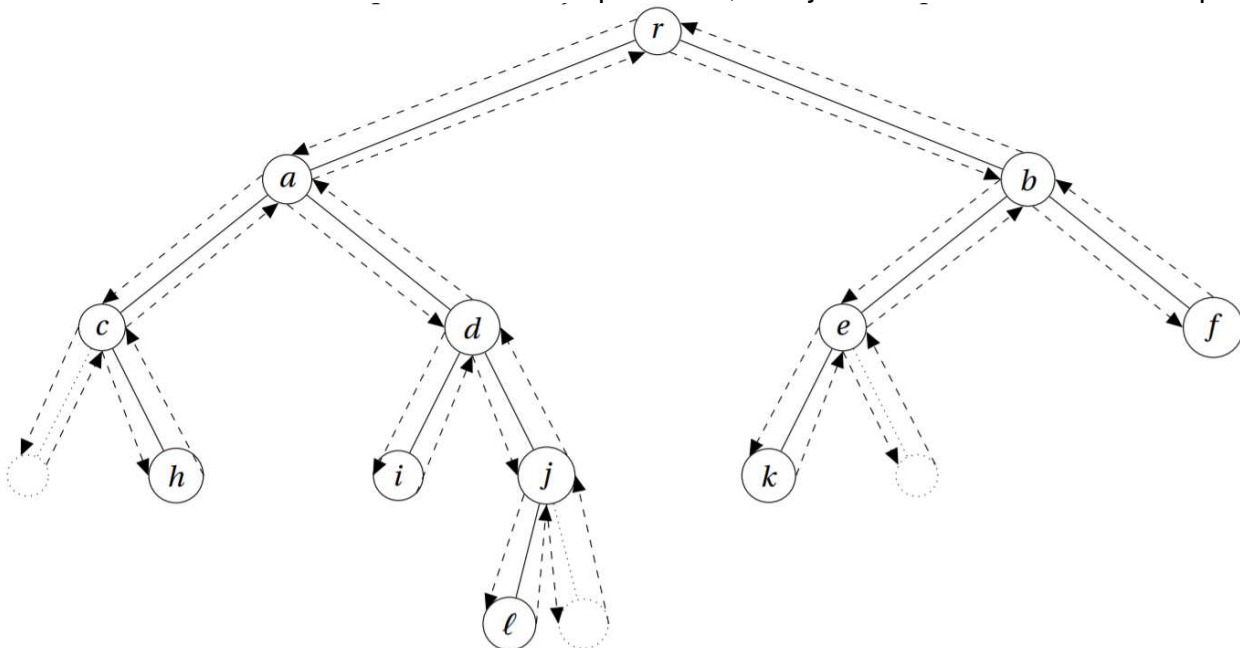


A partir de ce contour, on définit trois parcours des sommets de l'arbre :

1. l'ordre **préfixe** : on liste chaque sommet la première fois qu'on le rencontre dans la balade.
Ce qui donne ici : **r, a, c, h, d, i, j, l, b, e, k, f**
2. l'ordre **postfixe** : on liste chaque sommet la dernière fois qu'on le rencontre.
Ce qui donne ici : **h, c, i, l, j, d, a, k, e, f, b, r**
3. l'ordre **infixe** : on liste chaque sommet ayant un fils gauche la seconde fois qu'on le voit et chaque sommet sans fils gauche la première fois qu'on le voit.
Ce qui donne ici : **c, h, a, i, d, l, j, r, k, e, b, f**

Visualisation graphique des parcours :

Dans la balade schématisée plus haut, on ajoute les fils fantômes manquants :



On peut ainsi considérer qu'on passe une fois à **gauche** de chaque nœud (en descendant), une fois **en dessous** de chaque nœud, une fois à **droite** de chaque nœud (en remontant).

Vérifier, sur l'exemple, que chacun des ordres préfixe, infixe, postfixe est obtenu en listant tous les mots :

- soit lorsqu'on passe à leur gauche, cela nous donne l'ordre _____
- soit lorsqu'on passe à leur droite, cela nous donne l'ordre _____
- soit lorsqu'on passe en-dessous, cela nous donne l'ordre _____

Préfixe	Infixe	Suffixe
Fonction visitePréfixe(arbre) : Si arbre n'est pas vide : visiteTerrace visitePréfixe (fils gauche) visitePréfixe (fils droit)	Fonction visiteInfixe(arbre) : Si arbre n'est pas vide : visiteInfixe (fils gauche) visiteTerrace visiteInfixe (fils droit)	Fonction visiteSuffixe(arbre) : Si arbre n'est pas vide : visiteSuffixe (fils gauche) visiteSuffixe (fils droit) visiteTerrace

2 Parcours en largeur

Principe : on parcourt tous les nœuds de hauteur 1 (la racine), puis tous les nœuds de hauteur 2, ceux de hauteur 3 etc. Le parcours se fait en général de gauche à droite.

On utilise pour cela une file

Étapes de l'algorithme :

1. Mettre le nœud source dans la file.
2. Retirer le nœud du début de la file pour le traiter.
3. Mettre le fils gauche et le fils droits lorsqu'ils sont non vides, non explorés, à la fin de la file.
4. Si la file n'est pas vide reprendre à l'étape 2.

3 Arbres binaires de recherche (ABR)

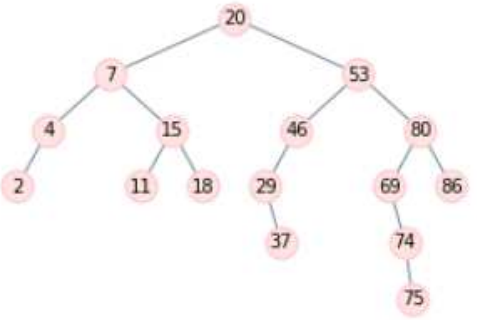
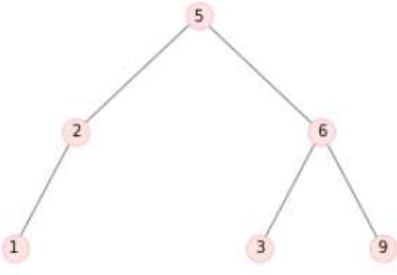
3.1 Définition

Un arbre binaire de recherche, ou ABR, est un arbre binaire étiqueté possédant la propriété suivante :

Pour tout nœud x , tous les nœuds situés dans le sous-arbre gauche de x ont une valeur inférieure ou égale à celle de x , et tous les nœuds situés dans le sous-arbre droit ont une valeur supérieure ou égale à celle de x .

Les arbres binaires de recherche servent, comme leur nom l'indique, à rechercher rapidement des éléments ordonnés.

Exemples :

	
Est un arbre binaire de recherche	N'est pas un arbre binaire de recherche

3.2 Algorithmes sur les ABR

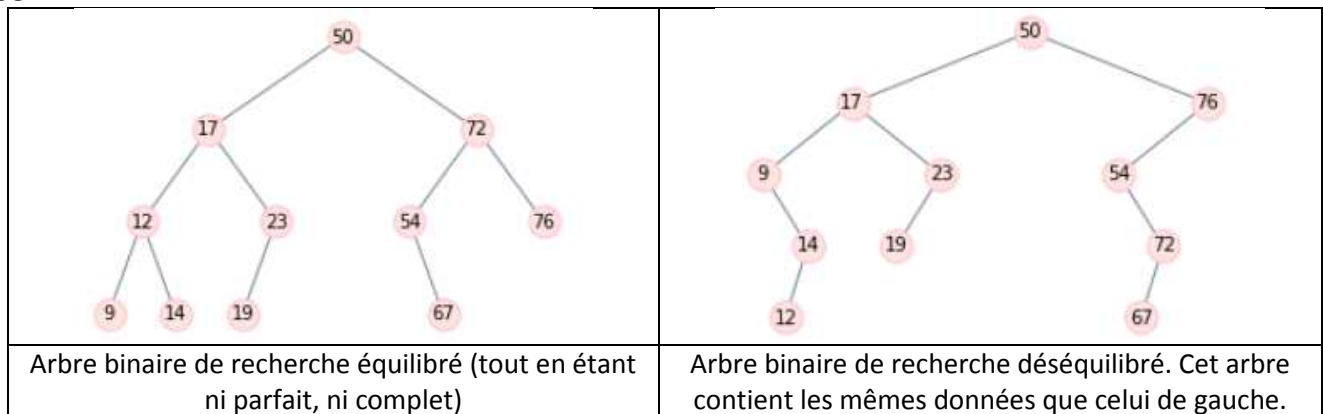
3.2.1 Recherche dans un ABR

Principe : suivant la valeur à rechercher, et la valeur du nœud sur lequel on est, on cherche soit dans le sous-arbre gauche, soit dans le sous-arbre droit

Fonction récursive recherche(*ABR*, *valeur*) :

- Si** *ABR* est vide
 - Retourner** None # variante : retourner Faux
- Sinon**
 - $valeur(x) = \text{étiquette}(ABR)$ # valeur de la racine
 - Si** $valeur < valeur(x)$:
 - Retourner** recherche(*fils_gauche*, *valeur*)
 - Sinon si** $valeur > valeur(x)$:
 - Retourner** recherche(*fils_droit*, *valeur*)
 - Sinon**
 - Retourner** ABR # variante : retourner Vrai

Pseudo-définition (peu rigoureuse et incomplète) : un arbre est équilibré si les hauteurs entre les sous-arbres gauche et droit sont différentes de 1 au maximum. Les arbres complets ou parfaits sont équilibrés.



Complexité :

Si l'arbre binaire de recherche est équilibré, alors le coût en temps de la recherche est en $O(\log n)$, où n est le nombre de nœuds de l'arbre.

Dans le cas où l'arbre binaire de recherche n'est pas équilibré, la recherche est en $O(n)$. Le cas le pire étant le cas où l'ABR est filiforme.

Remarque : on peut écrire une version itérative de cet algorithme (exercice intéressant)

3.2.2 Insertion dans un ABR

Le problème de l'insertion dans un ABR correspond à celui de la construction de l'ABR. L'insertion dans un ABR commence par la recherche de l'endroit où l'on doit insérer la valeur. Si la valeur à insérer est plus petite –ou égale– que la valeur du nœud, on va à gauche, si elle est plus grande on va à droite. On arrive à un moment à un arbre vide : c'est là où on ajoute la nouvelle valeur. La gestion des arbres vides n'est cependant pas si simple dans cette approche.

Fonction récursive ajoute(*ABR*, *valeur*) :

- Si** *ABR* est vide
 - $ABR = \text{Arbre}(valeur, \text{None}, \text{None})$
- Sinon si** $valeur \leq \text{étiquette_noeud}(ABR)$
 - ajoute(*fils_gauche*, *valeur*)
- Sinon**
 - ajoute(*fils_droit*, *valeur*)

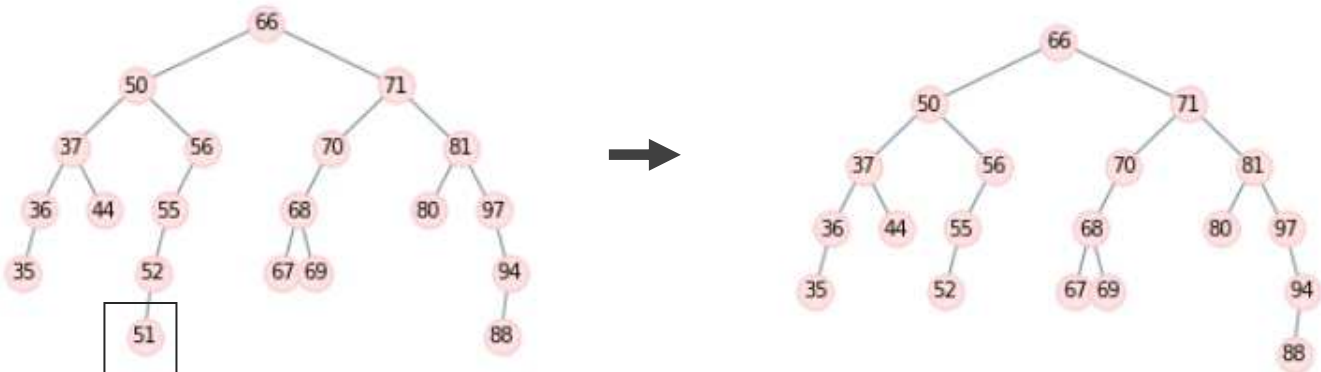
Complexité : la complexité est identique à celle de la fonction de recherche. Dans le cas d'un arbre équilibré, elle est en $O(\log n)$.

Remarque : cette méthode ne donne pas forcément des arbres équilibrés. Pour avoir un arbre équilibré, une méthode efficace est d'insérer les éléments dans un ordre aléatoire.

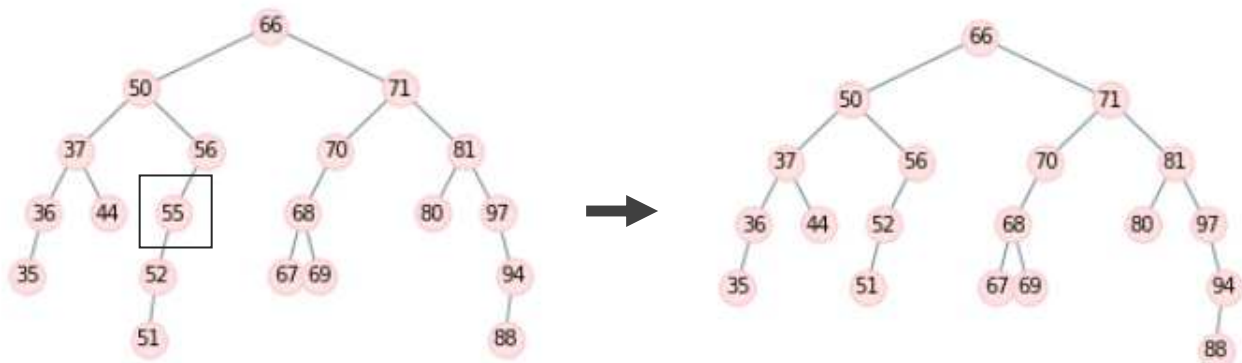
3.2.3 Suppression dans un ABR

Le problème se décompose en trois cas, suivant le nombre de fils du nœud à supprimer.

Si le nœud est une feuille (pas de fils), alors on supprime simplement le lien du père vers ce nœud (on décroche le nœud). Si ce lien n'existe pas, l'arbre devient l'arbre vide.



Si le nœud a un seul fils, alors on décroche le nœud comme précédemment, et on remplace son fils dans le nœud père. Si le père n'existe pas, le nœud fils devient la racine de l'arbre.



Si le nœud a deux fils, on cherche dans le sous-arbre gauche le nœud *MaxLocal* de valeur la plus grande (ou le nœud de valeur la plus petite dans le sous-arbre droit). La valeur de ce nœud va remplacer la valeur supprimée. Comme ce nœud *MaxLocal* a la valeur maximale dans le sous-arbre gauche, il n'a pas de fils droit. On peut donc le décrocher, comme on l'a fait dans le cas précédent.

