

Arbres Binaires de Recherche

Définition, Insertions, Suppressions

Ce document a pour objectif de vous familiariser avec une nouvelle structure de données algorithmique abstraite : les *arbres* (en anglais : *trees*). En particulier, nous y aborderons les *arbres binaires de recherche* ou *ABR* (en anglais : *binary search tree* ou *BST*) qui sont des cas spécifiques d'arbres binaires.

Pour rappel, un arbre binaire est un arbre d'arité/de degré 2, c'est-à-dire que chaque nœud dispose au maximum de 2 fils : un fils gauche et un fils droit. Il n'existe aucune boucle dans un arbre : chaque nœud n'a qu'un seul père (excepté la racine). La profondeur d'un nœud est la différence de niveau entre le nœud étudié et la racine de l'arbre. La hauteur d'un arbre est la profondeur du nœud le plus éloigné de la racine. La taille d'un arbre est le nombre de nœuds contenus dans l'arbre.

1 Définitions

Les *arbres binaires de recherche* ou *ABR* (en anglais : *binary search tree* ou *BST*) sont des arbres binaires dont les clés sont ordonnées, formant un ensemble totalement ordonné. Les clés représentent les éléments que l'on souhaite insérer dans l'arbre. Mathématiquement parlant, il est donc nécessaire que les éléments soient comparables deux à deux avec une relation d'ordre (lequel est le plus petit ? lequel est le plus grand ? sont-ils égaux ?).

Les éléments d'un arbre binaire de recherche respectent une contrainte très précise : tous les éléments dans le sous-arbre gauche sont plus petits (ou égaux) que l'élément dans la racine, et tous les éléments dans le sous-arbre droit sont plus grands que l'élément dans la racine. En appliquant ce principe récursivement, on est capable de retrouver rapidement des clés parmi l'ensemble des clés stockées. Selon les cas, on peut refuser (ou accepter) que des éléments égaux soient simultanément stockés.

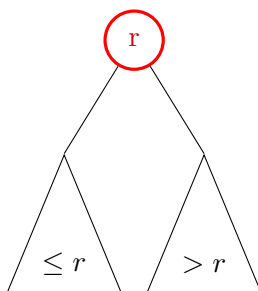


Fig.1 : Règle générale des arbre binaire de recherche

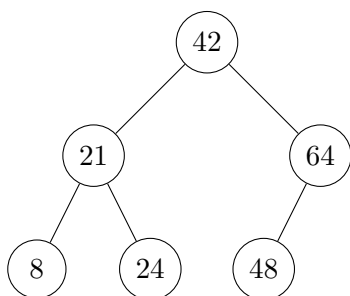


Fig.2 : Un arbre binaire de recherche

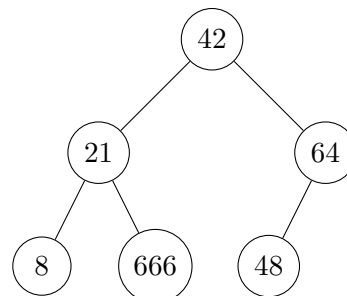


Fig.3 : Un arbre binaire qui n'est **pas** un ABR

2 Recherche dans un ABR

La recherche d'un élément dans l'arbre va donc simplement se faire en comparant la clé recherchée avec celle de la racine de l'arbre : si la clé recherchée est plus petite, alors on descend dans le sous-arbre gauche et on recommence récursivement le test, sinon, on descend dans le sous-arbre droit et on recommence récursivement le test.

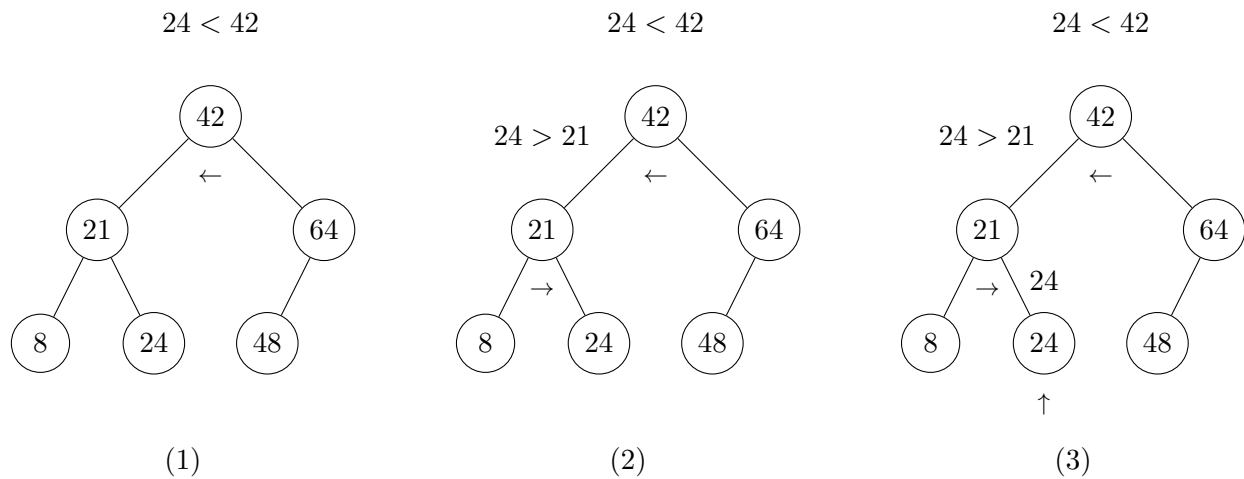


Fig.4 : Recherche de la clé 24 dans un ABR

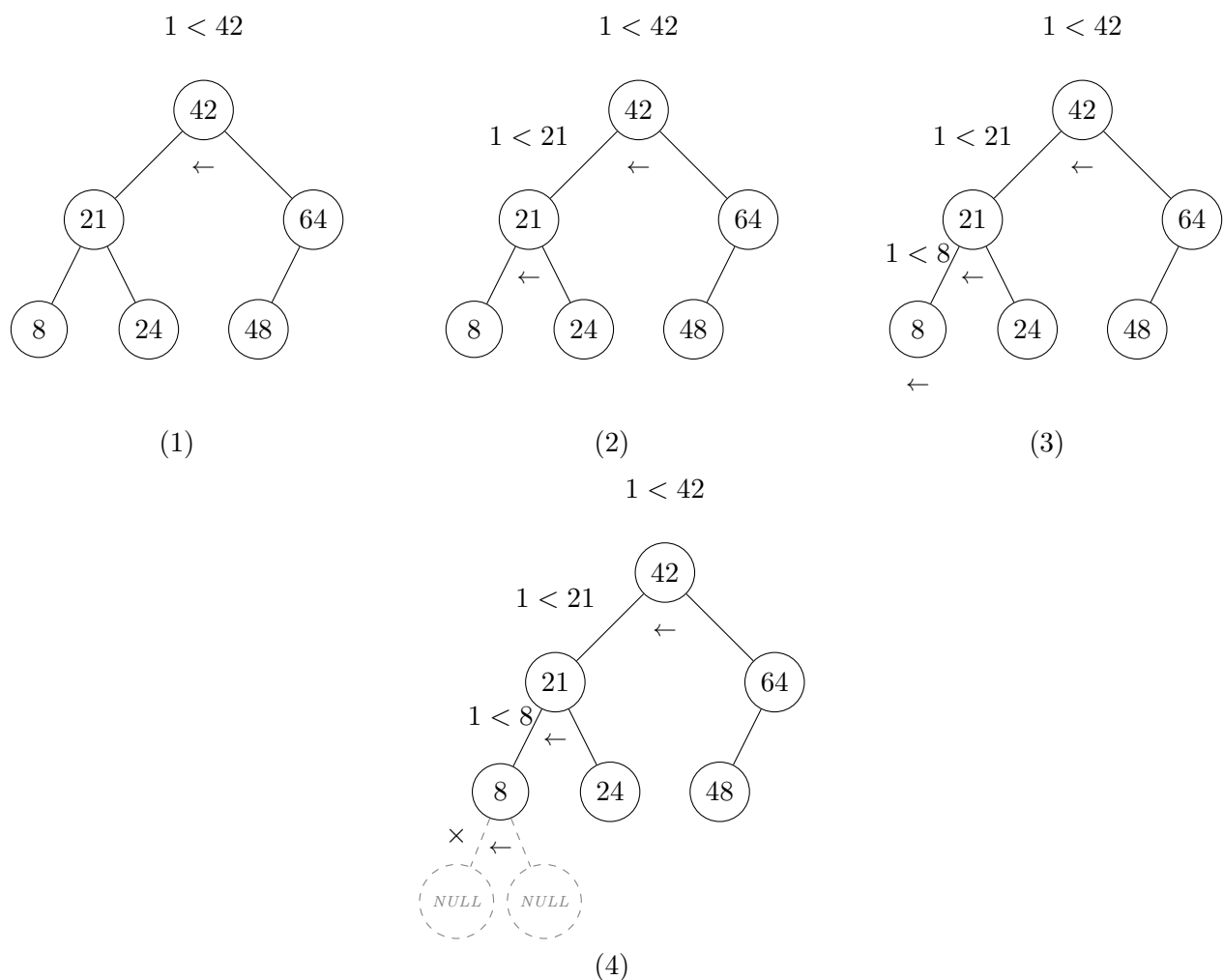


Fig.5 : Recherche de la clé 1 dans un ABR

Quand la clé est identique à celle du nœud courant, alors on a trouvé le nœud correspondant, et on peut arrêter la recherche.

Si une clé est absente, on le saura dans le pire des cas en atteignant les feuilles de l'arbre.

La recherche dans un ABR est donc extrêmement simple à implémenter en récursif (et elle est fortement recommandée). Les cas d'arrêts sont : le nœud courant est-il vide ? le nœud courant contient-il la clé ? Si aucun de ces deux cas n'est vérifié, alors on compare la clé recherchée avec celle contenue dans le nœud pour en déduire vers quel fils descendre.

3 Insertion en feuille dans un ABR

Insérer des éléments dans un ABR se fait généralement en ajoutant des feuilles : on insère un premier élément dans un arbre vide, on obtient donc un arbre composé d'une racine toute seule. Ensuite, on ajoute un deuxième élément qui se placera à la gauche ou à la droite de la racine selon la règle précédente (les éléments plus grands que la racine sont placés à sa droite, et les éléments plus petits ou éventuellement égaux à la racine sont placés à sa gauche).

Vous vous rendez compte que ceci peut engendrer des problèmes car un côté pourrait se remplir plus vite que l'autre. Plusieurs solutions existent, mais impliquent d'ajouter des règles présentées plus tard.

L'ajout en feuille implique tout d'abord de chercher à quel endroit placer le nouvel élément, et, dès que l'on tombe sur un nœud vide/inexistant, alors on le crée et on ajoute l'élément à cet endroit. On descend donc récursivement dans l'arbre avec la règle « plus petit (ou égal), ou plus grand », puis on place l'élément dans le tableau à la position adaptée *ou* on crée un nouveau *node* que l'on place en fils d'un parent.

Si l'arbre est vide, on ne fait bien évidemment que créer une racine qui sera également une feuille.

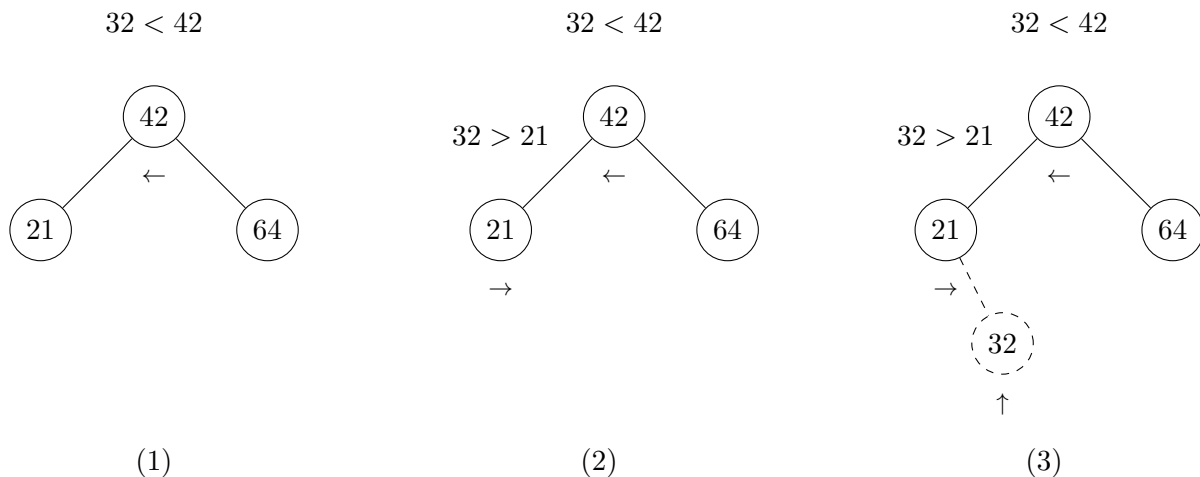


Fig.6 : Insertion de la clé 32 dans un ABR

La version récursive implique de mettre à jour chaque nœud traversé : lorsque l'on descend dans un fils, on va mettre à jour sa valeur avec celle renvoyée par l'algorithme récursif (qui va renvoyer l'adresse existante, excepté pour le nœud parent de la nouvelle feuille).

Attention, lors de l'implémentation de cet algorithme en version itérative dans le cas d'une structure à pointeurs, vous devrez cette fois observer avec un cran d'avance l'état des fils (principalement pour pouvoir mettre à jour le père).

Néanmoins, n'oubliez pas le cas où l'arbre est vide, ou qu'il ne contient qu'une racine.

4 Suppression dans un ABR

Supprimer un élément d'un ABR est une opération complexe dans certaines situations précises.

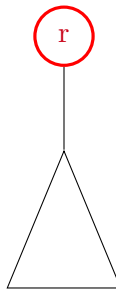
On cherche d'abord le nœud que l'on veut supprimer, puis, on avise selon la situation rencontrée. Tout comme pour l'insertion, le traitement récursif implique d'avoir un cran d'avance pour observer l'état des fils.

Trois cas peuvent se produire :

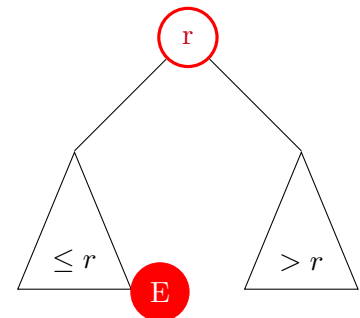
- Cas n°1 : le nœud que l'on veut supprimer est une feuille : on met à jour les informations du père pour supprimer le lien, puis on supprime le nœud en lui-même.
- Cas n°2 : le nœud que l'on veut supprimer est un point simple (il n'a qu'un fils) : on met à jour les informations du père pour se lier au fils du nœud à supprimer, puis on supprime le nœud en lui-même.
- Cas n°3 : le nœud que l'on veut supprimer a ses deux fils remplis : on va chercher l'élément le plus proche de la clé que l'on veut supprimer pour pouvoir échanger leur place, puis supprimer le nœud. On peut donc remonter le nœud dont la clé est juste inférieure (ou supérieure) à celle que l'on cherche à supprimer.



(Cas n°1) Le nœud à supprimer est une feuille



(Cas n°2) Le nœud à supprimer est un point simple



(Cas n°3) Le nœud à supprimer est un point double (remontée par la gauche)

En pratique, les trois cas peuvent se matérialiser ainsi :

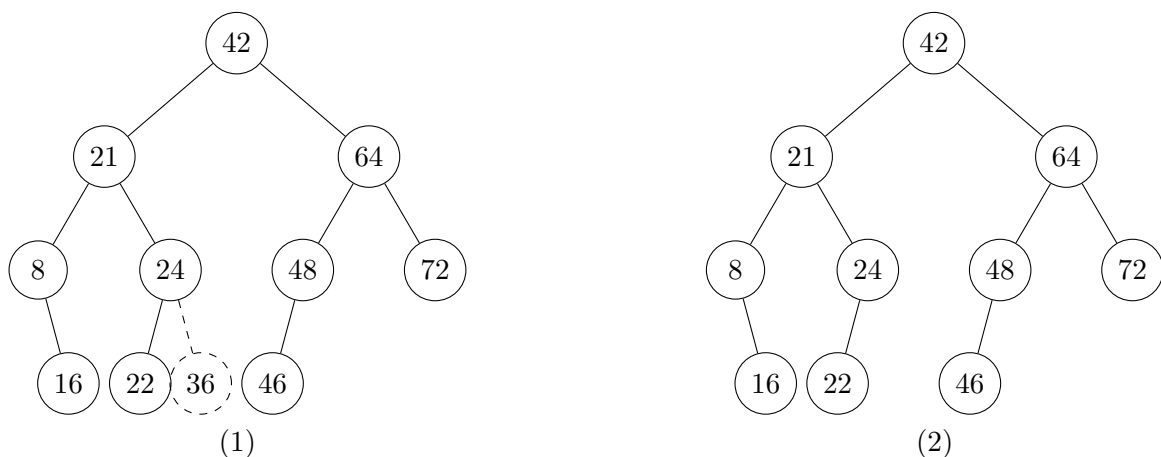


Fig.7 : (Cas n°1) Suppression d'une feuille dans un ABR

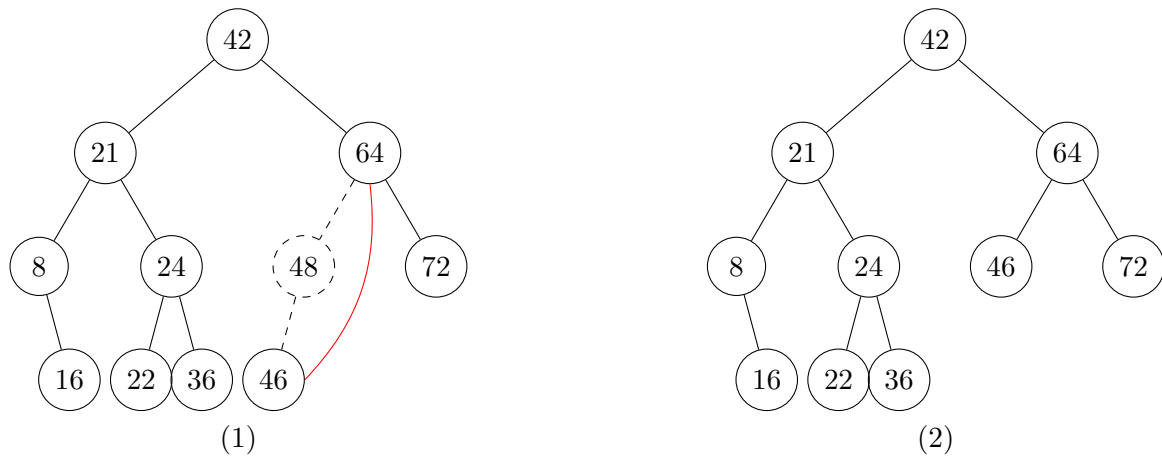


Fig.8 : (Cas n°2) Suppression d'un point simple dans un ABR

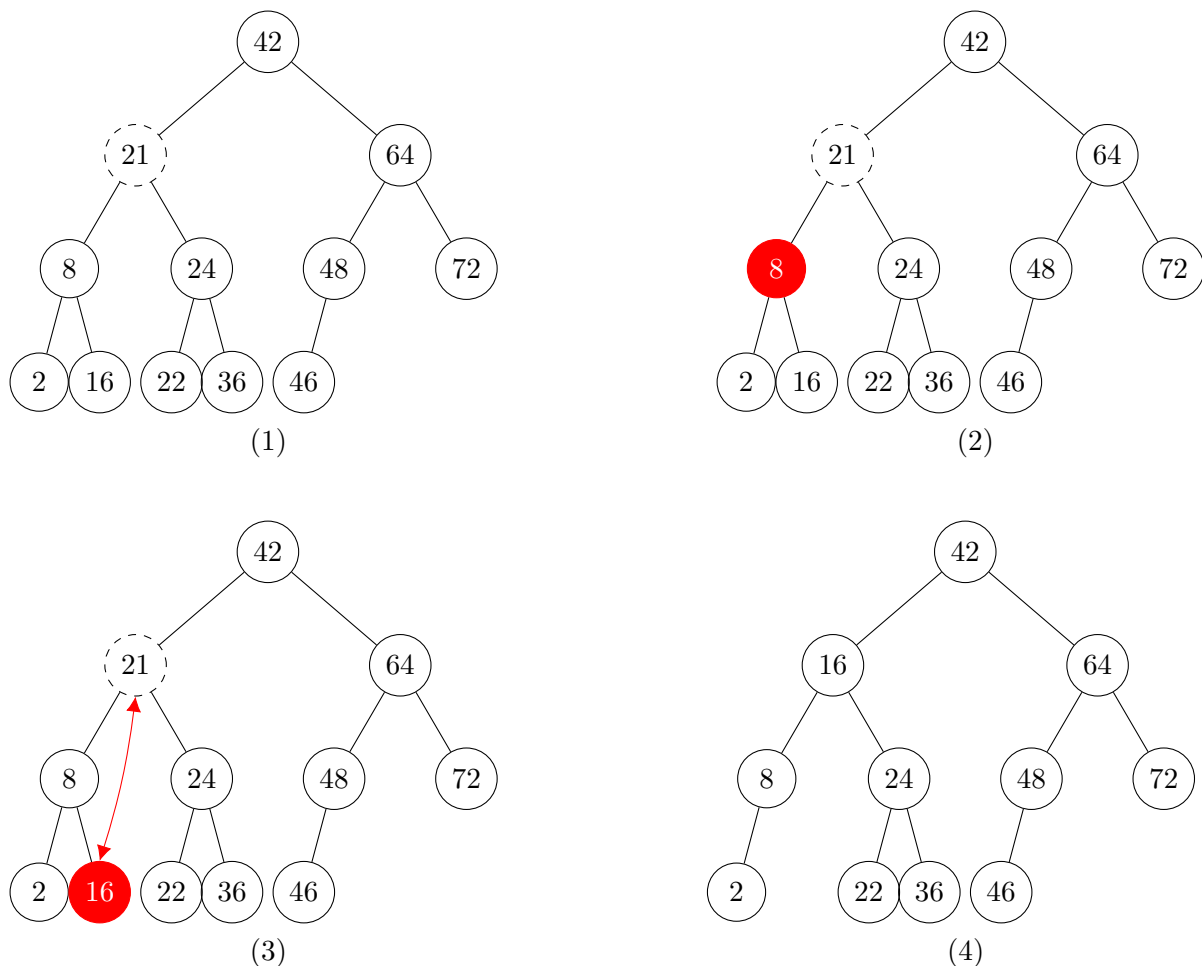


Fig.9 : (Cas n°3) Suppression d'un nœud à deux fils dans un ABR

La suppression d'un nœud contenant deux fils fonctionne donc sur le principe d'échange avec le nœud contenant la clé la plus proche : dans l'exemple précédent, 21 peut être remplacé par 16 ou 22 (selon la stratégie choisie). Il faut donc chercher le plus grand des éléments plus petits (respectivement le plus petit des éléments plus grands) afin d'échanger le nœud le contenant avec celui que l'on veut supprimer. Attention, ce nœud peut lui même être un parent, mais dans ce cas il sera un point simple.

5 Insertion en racine dans un ABR

L'insertion la plus commune dans des ABR se fait par les feuilles. Il existe une autre méthode où l'on insère les éléments par la racine.

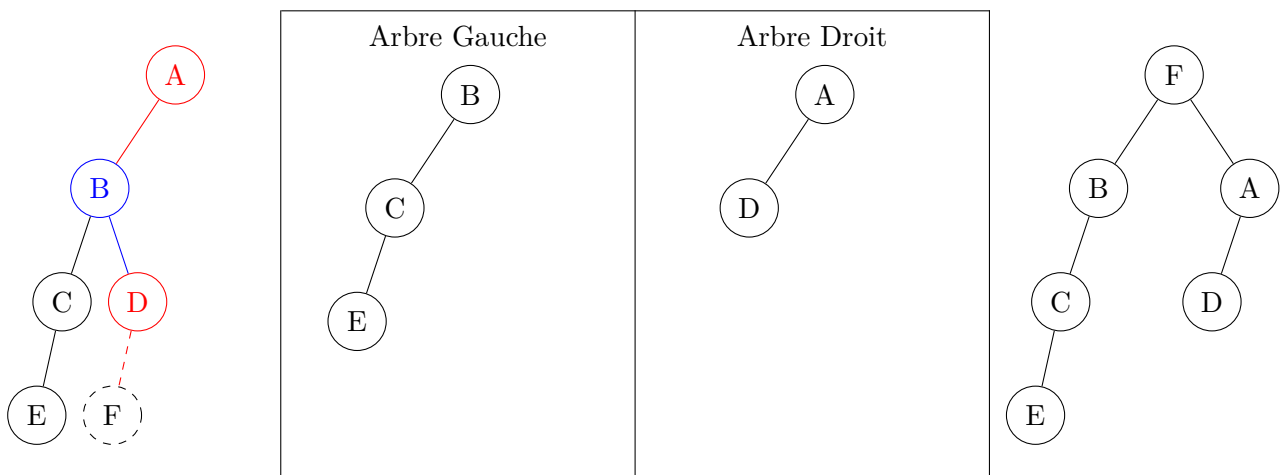
L'algorithme naïf à ce type d'insertion pourrait être d'ajouter chaque élément en tant que racine des précédents. Néanmoins, cette méthode ne permet de former que des arbres filiformes, et elle n'assure absolument pas de conserver l'ordre des éléments dans un ABR.

Essayez d'insérer exclusivement en tant que racine ces éléments dans cet ordre : $15 - 10 - 5 - 8$. Vous vous rendez compte que la racine 8 aura comme fils gauche le nœud contenant 5, qui aura à son tour un fils droit contenant 10. Or, d'après la contrainte fondamentale des ABR « tous les nœuds à gauche de la racine doivent être plus petits que celle-ci », aucun nœud du sous-arbre gauche devrait être plus grand que 8, ce qui n'est pas le cas puisque 10 et 15 s'y trouvent. Ainsi, l'algorithme naïf ne peut donc pas fonctionner.

L'insertion en racine implique de devoir respecter l'ordre des éléments contenus dans l'arbre actuel, tout en mettant à jour la racine. Deux approches fonctionnent : soit on coupe l'arbre au fur et à mesure de l'insertion récursive du nœud, soit on insère le nœud en feuille puis on effectue des rotations successives pour le remonter en racine.

5.1 Coupes

L'insertion en racine avec la technique de coupe vise à couper l'arbre en deux pour y insérer le nouvel élément comme racine : l'un des deux arbres contiendra tous les éléments plus petits (ou égaux) à celui inséré, et l'autre arbre contiendra tous les éléments plus grands que celui inséré. Ainsi, une fois ces arbres temporaires construits, il suffit d'utiliser le nouvel élément comme racine et lui adjoindre les deux arbres comme fils gauche et droit.



Pour couper l'arbre en deux sous-arbres, on s'appuie sur le parcours profondur comme lors d'une recherche : on descend dans le fils gauche si l'élément est plus petit (ou égal) à celui recherché, sinon on descend dans le fils droit. Chaque élément rencontré (et l'un des sous-arbres associé) est ajouté à l'arbre temporaire gauche ou droit.

- Si le nœud courant est plus petit (ou égal) que l'élément inséré, alors on déplace ce nœud et son fils gauche dans l'arbre temporaire gauche, puis, on supprime la référence en fils droit qu'il possède. Ceci permet d'ajouter éventuellement d'autres éléments en partie droite de l'arbre temporaire gauche.
- Si le nœud courant est plus grand que l'élément inséré, alors on déplace ce nœud et son fils droit dans l'arbre temporaire droit, puis, on supprime la référence en fils gauche qu'il possède. Ceci permet d'ajouter éventuellement d'autres éléments en partie gauche de l'arbre temporaire droit.
- Si le nœud courant est vide, alors on a fini la coupe de l'arbre, et il suffit maintenant de construire un nœud contenant le nouvel élément, et lui adjoindre les deux arbres temporaires.

L'implémentation sous forme de pointeurs permet donc d'effectuer assez facilement cette séparation par nœuds : il suffit de modifier un pointeur pour couper.

Afin d'illustrer le fonctionnement de l'insertion à la racine par coupes, un arbre est construit en suivant cette méthode pour insérer $8 - 12 - 21 - 16 - 96 - 64 - 72 - 42$ dans cet ordre précis.

(1) insertion en racine de 8

↓
∅

Coupe Gauche	Coupe Droite
∅	∅

Sous-arbre de Gauche	Sous-arbre de Droite
∅	∅

8

(2) insertion en racine de 12

12 > 8
↓
8

Coupe Gauche	Coupe Droite
∅	∅

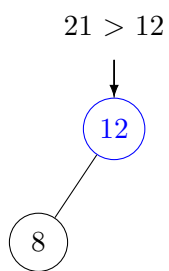
8
↓
∅

Coupe Gauche	Coupe Droite
8	∅

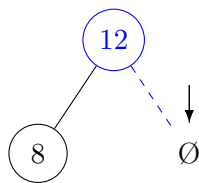
Sous-arbre de Gauche	Sous-arbre de Droite
8	∅

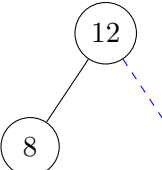
12
8

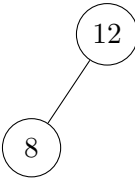
(3) insertion en racine de 21

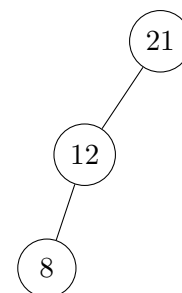


Coupe Gauche \emptyset	Coupe Droite \emptyset
-----------------------------	-----------------------------

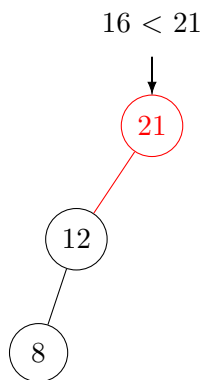


Coupe Gauche 	Coupe Droite \emptyset
---	-----------------------------

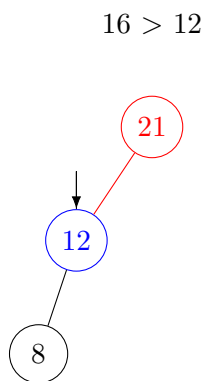
Sous-arbre de Gauche 	Sous-arbre de Droite \emptyset
---	-------------------------------------



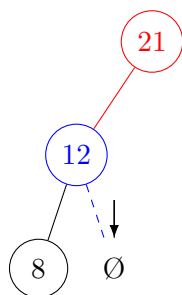
(4) insertion en racine de 16



Coupe Gauche	Coupe Droite
\emptyset	\emptyset

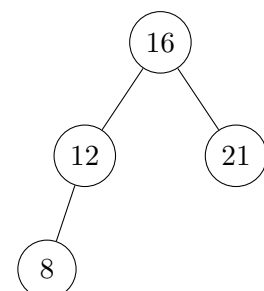


Coupe Gauche	Coupe Droite
\emptyset	

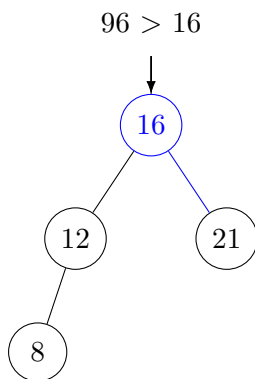


Coupe Gauche	Coupe Droite

Sous-arbre de Gauche	Sous-arbre de Droite



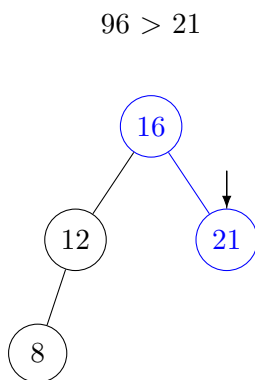
(5) insertion en racine de 96



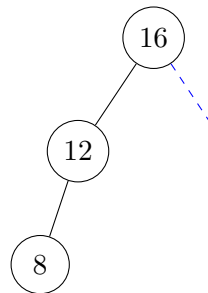
Coupe Gauche

 \emptyset

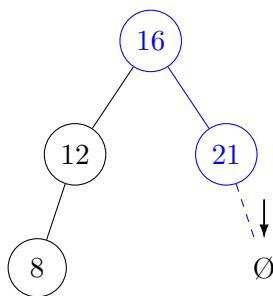
Coupe Droite

 \emptyset 

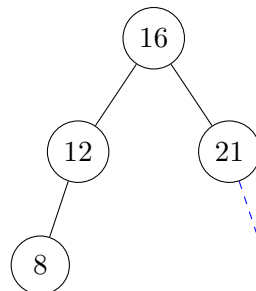
Coupe Gauche



Coupe Droite

 \emptyset 

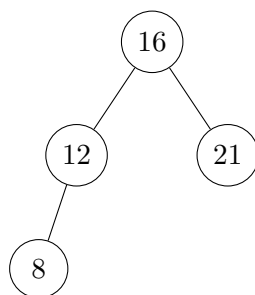
Coupe Gauche



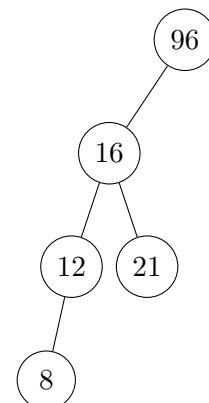
Coupe Droite

 \emptyset

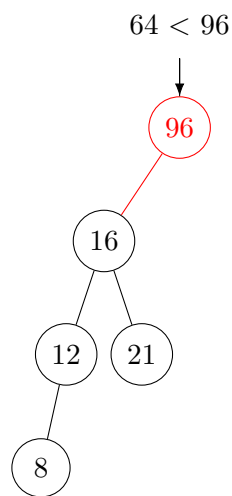
Sous-arbre de Gauche



Sous-arbre de Droite

 \emptyset 

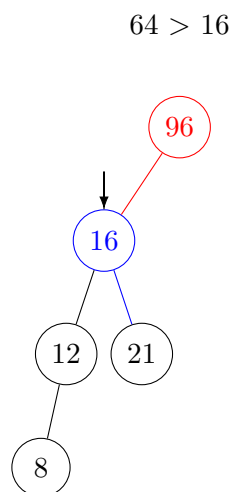
(6) insertion en racine de 64



Coupe Gauche

 \emptyset

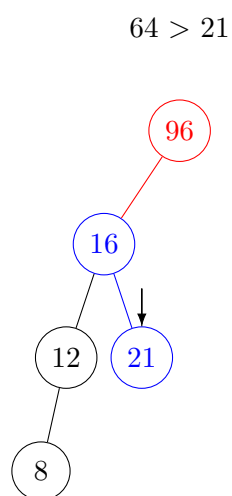
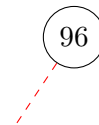
Coupe Droite

 \emptyset 

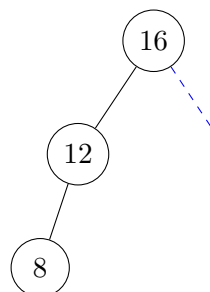
Coupe Gauche

 \emptyset

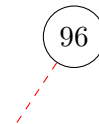
Coupe Droite

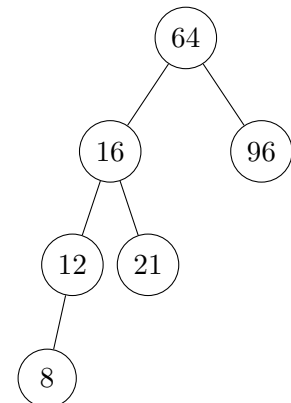
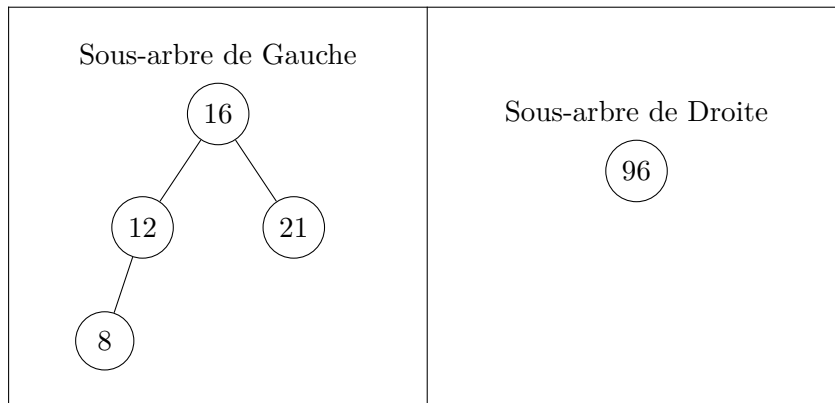
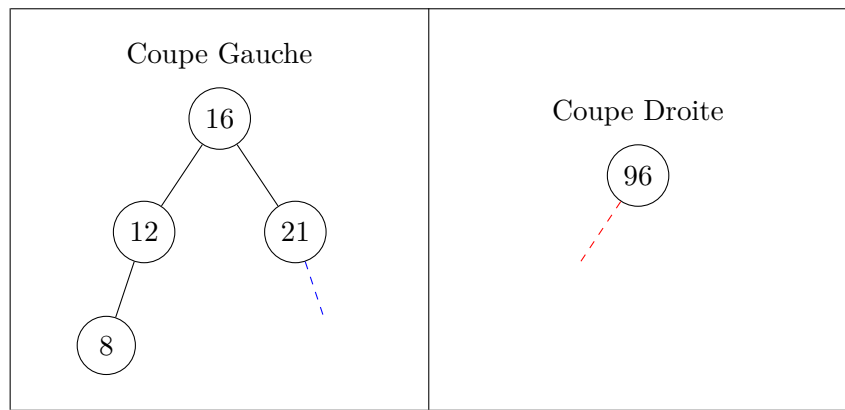
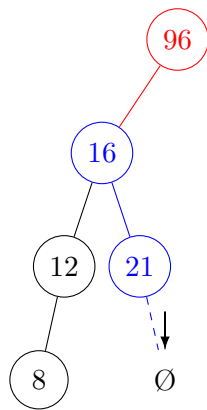


Coupe Gauche

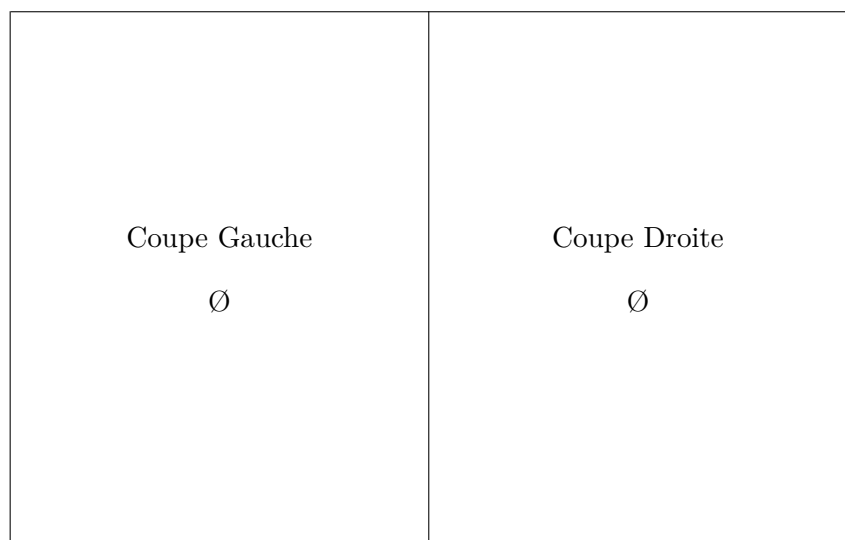
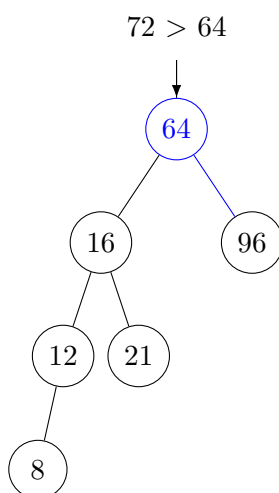


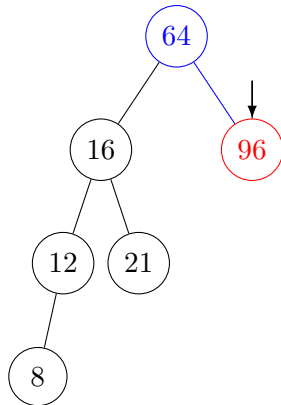
Coupe Droite



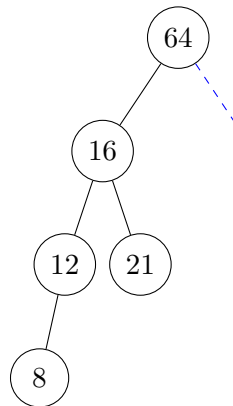


(7) insertion en racine de 72

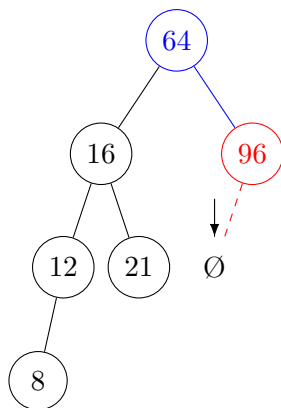


$72 < 96$ 

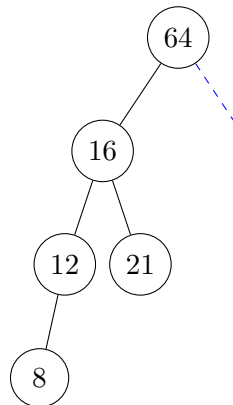
Coupe Gauche



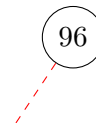
Coupe Droite

 \emptyset 

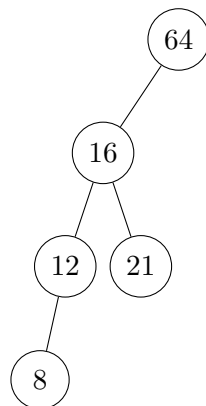
Coupe Gauche



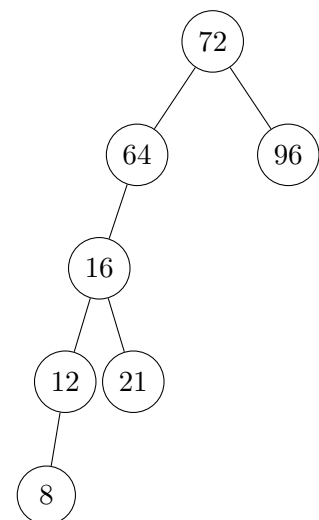
Coupe Droite



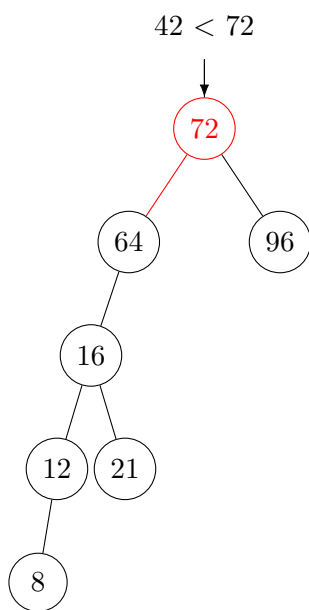
Sous-arbre de Gauche



Sous-arbre de Droite



(8) insertion en racine de 42

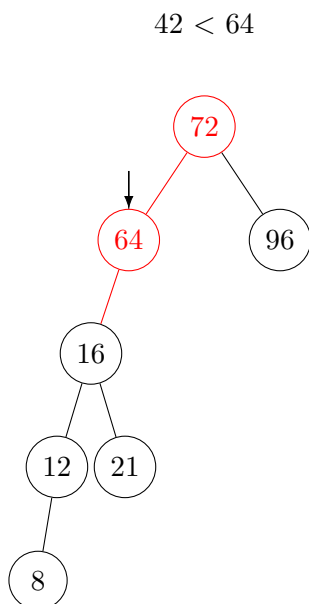


Coupe Gauche

\emptyset

Coupe Droite

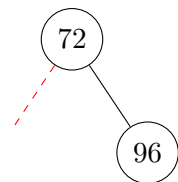
\emptyset

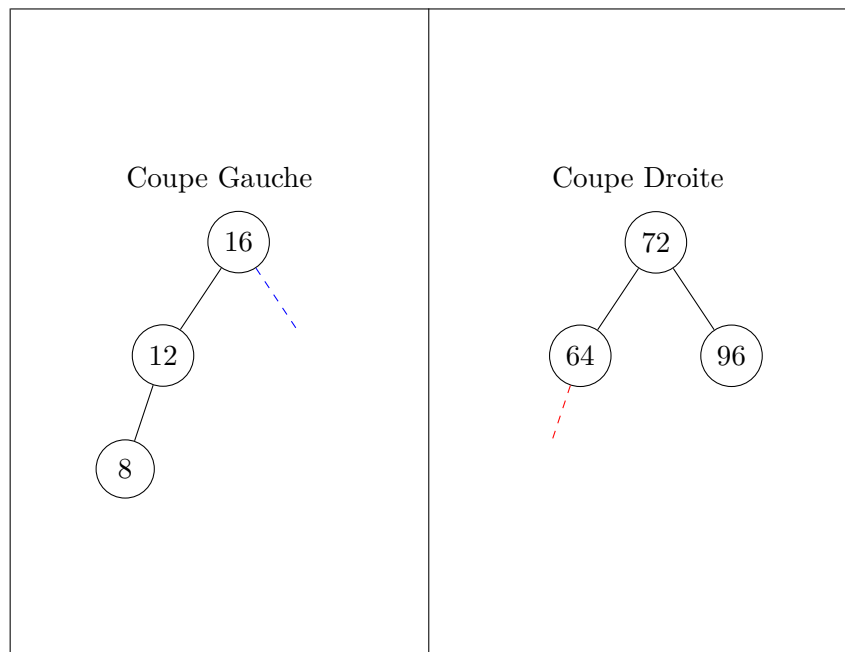
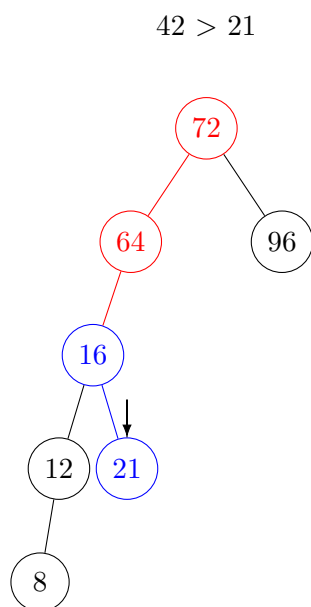
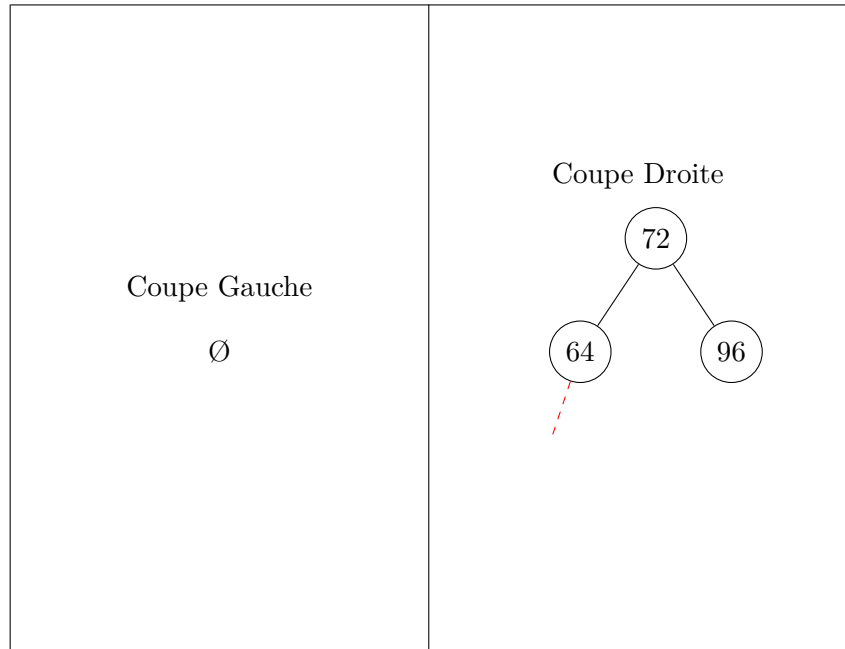
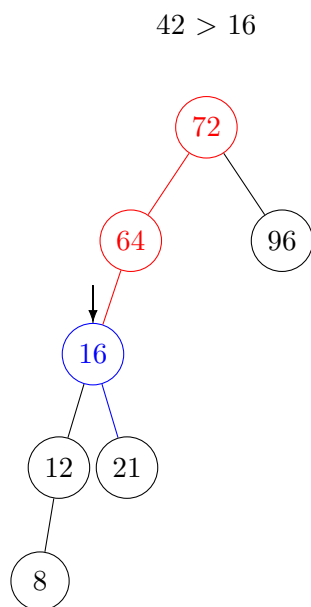


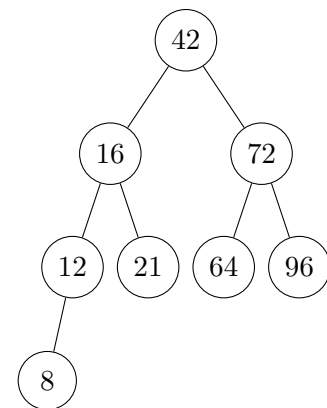
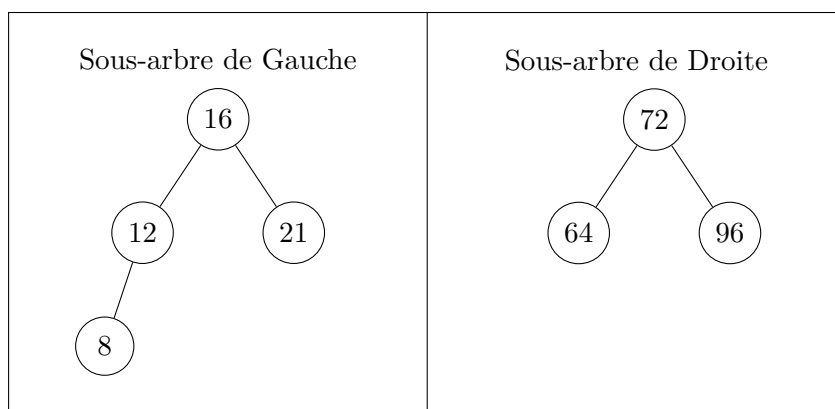
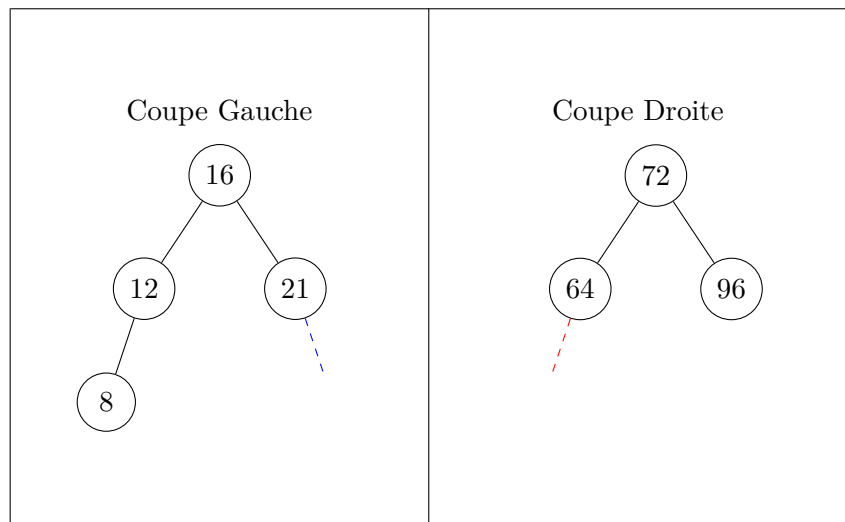
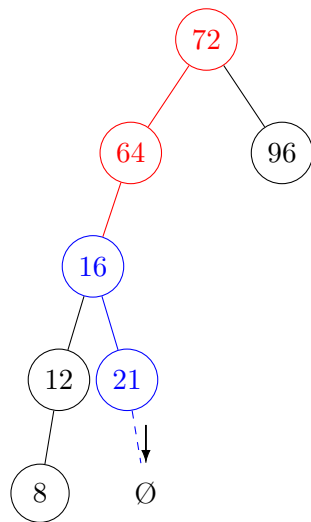
Coupe Gauche

\emptyset

Coupe Droite







On remarque qu'à la fin de l'étape 7 (insertion de 72) l'arbre a beaucoup plus de nœuds dans le sous-arbre gauche de la racine (5 nœuds : 64, 21, 16, 12, 8) que dans le sous-arbre droit de la racine (1 nœud : 96). À l'inverse, à la fin de l'étape 8 (insertion de 42), l'arbre est devenu presque complet : seul le dernier niveau n'est pas plein.

Cet effet est impossible à obtenir avec l'ajout en feuille : une fois que la racine est fixée, elle l'est définitivement. Pour éviter d'obtenir un arbre filiforme ou en peigne, il faudrait donc que les éléments soient ordonnés d'une manière convenable (par exemple en étant distribués de telle manière que le premier élément soit la médiane de la distribution, que le deuxième soit l'élément juste à gauche de la médiane, que le troisième soit l'élément juste à droite de la médiane, et ainsi de suite).

Néanmoins, pouvoir ordonner les éléments est déjà un objectif des arbres : on ne doit donc pas les trier avant de les insérer, cela n'aurait que peu de sens.

Une autre méthode d'insertion en racine existe et s'appuie sur les *rotations* qui sont des techniques couramment utilisées avec les arbres dans d'autres cas.

5.2 Rotations

La rotation implique d'échanger la place de 2 nœuds et d'un de leurs fils. Plusieurs rotations existent, mais nous nous intéresserons pour le moment à deux d'entre elles : la *rotation gauche* et la *rotation droite*. Ces deux rotations s'inversent mutuellement (faire une rotation gauche puis une rotation droite remet l'arbre dans son état d'origine).

Les rotations sont très utilisées pour équilibrer les arbres, mais cette notion sera abordée plus tard. Dans le contexte de l'insertion en racine, les rotations sont utilisées pour déplacer un nœud et le remonter progressivement jusqu'à la racine.

5.2.1 Rotation Gauche

La rotation gauche vise à remonter le fils droit d'un nœud (il passe donc à gauche vers la place de son père).

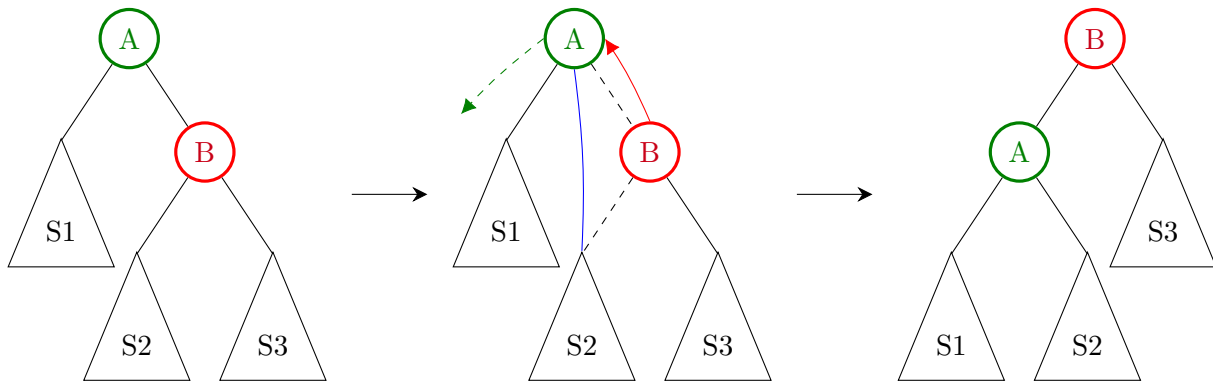


Fig.10 : Rotation gauche (sous-arbres)

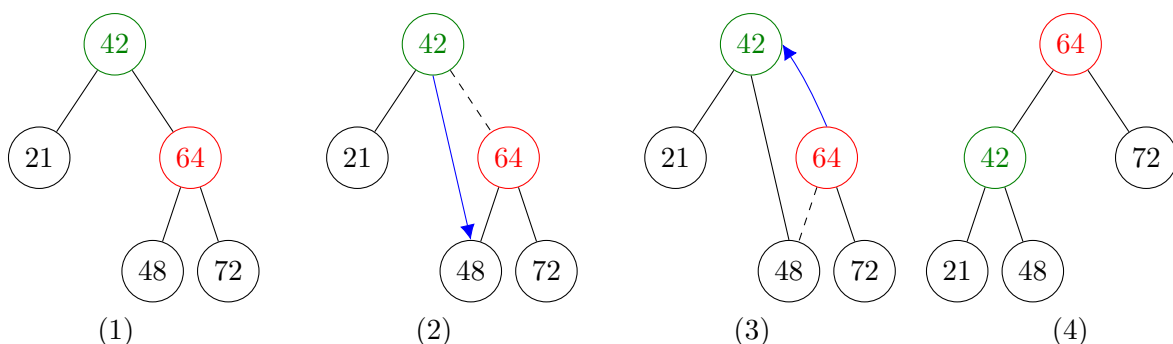


Fig.11 : Rotation gauche (nœuds)

5.2.2 Rotation Droite

La rotation droite vise à remonter le fils gauche d'un nœud (il passe donc à droite vers la place de son père).

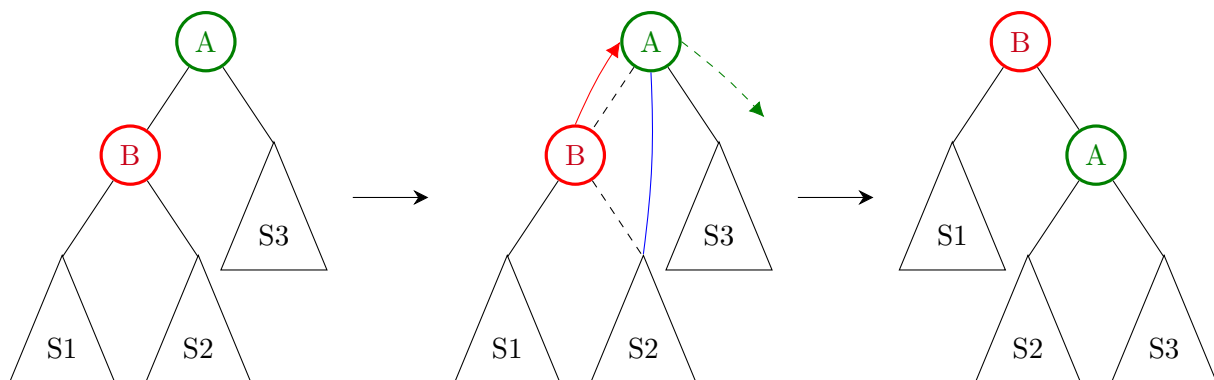


Fig.12 : Rotation droite (sous-arbres)

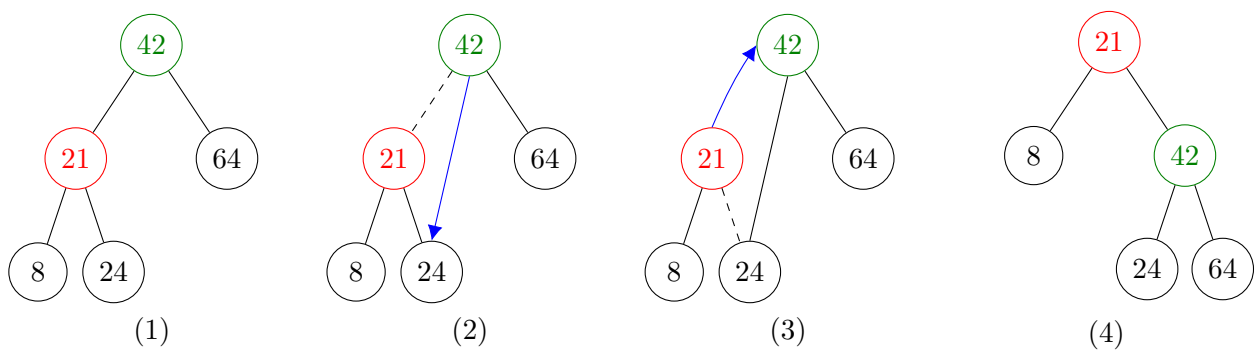
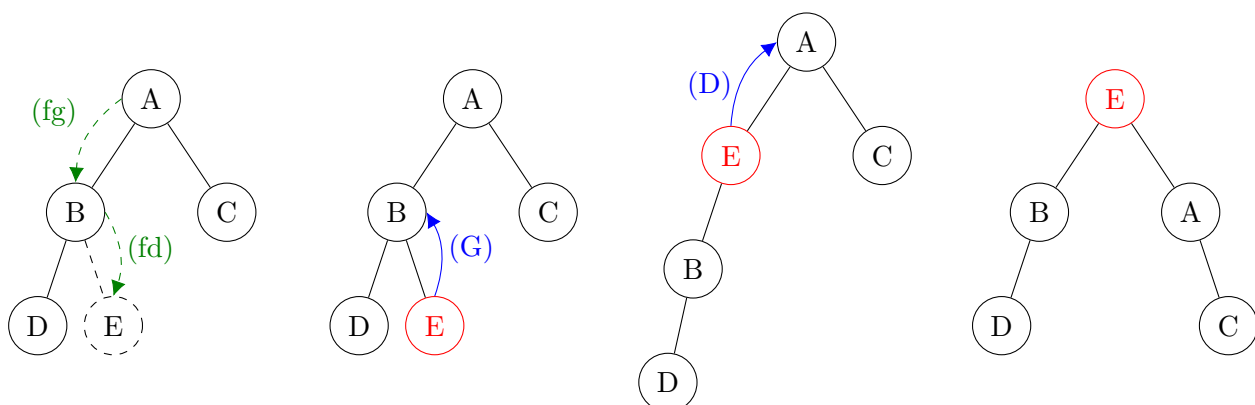


Fig.13 : Rotation droite (nœuds)

5.2.3 Insertion en racine par rotations

L'usage des rotations gauche et droite dans le cadre de l'insertion en racine d'un ABR est très simple : on insère en feuille le nœud, puis, on applique autant de rotations que nécessaires pour remonter le nœud. La rotation à appliquer est l'inverse de la position du fils dans lequel on est descendu pour placer le nœud : lorsque l'on insère le nœud dans le fils droit, alors on applique une rotation gauche pour le remonter (et inversement).



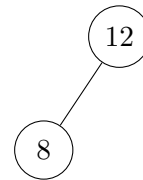
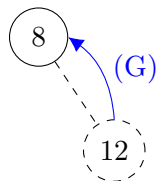
La construction d'un ABR par insertion en racine avec rotations des éléments 8 – 12 – 21 – 16 – 96 – 64 – 72 – 42 (dans cet ordre précis) donnera les étapes suivantes.

Attention : les premières insertions sont évidentes car les rotations ne modifient qu'un seul sous-arbre, mais, à partir de l'insertion de l'élément 64, les rotations échangent plusieurs sous-arbres.

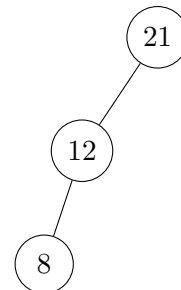
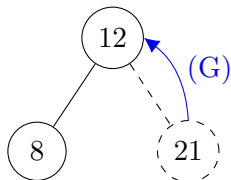
On notera que pour créer des feuilles, il faut faire en sorte que les insertions successives autour de la racine s'inversent (une insertion fait une rotation à gauche sur la racine, puis l'insertion suivante fait une rotation à droite sur la racine, et ainsi de suite).



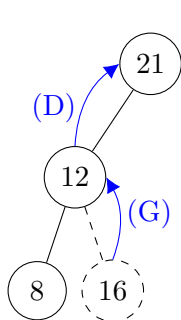
(1) - insertion de 8



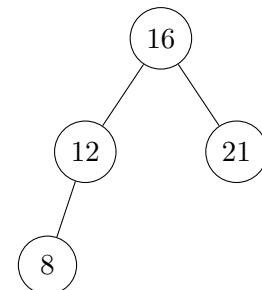
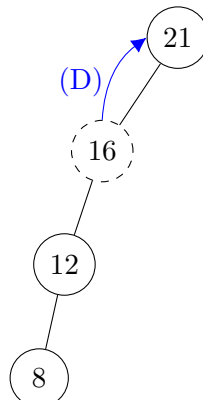
(2) - insertion de 12

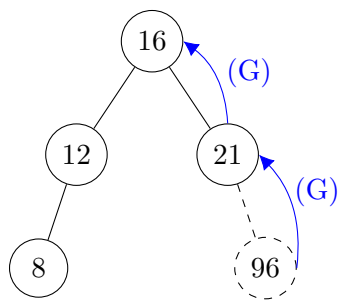


(3) - insertion de 21

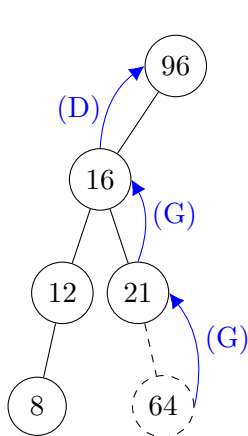
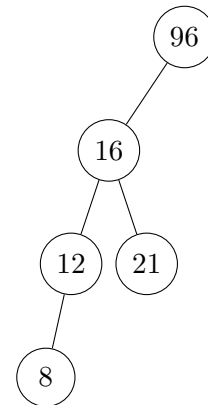
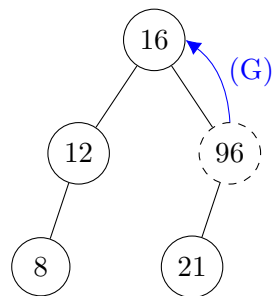


(4) - insertion de 16

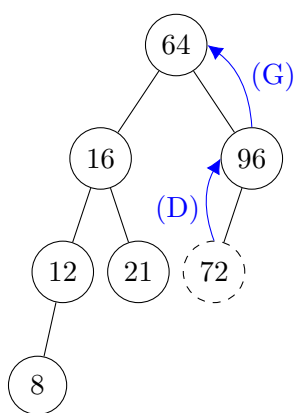
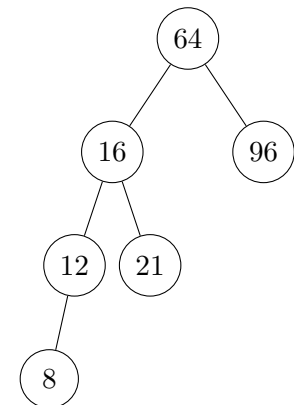
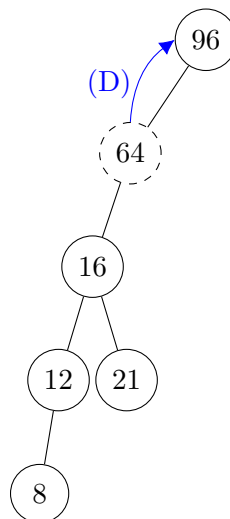
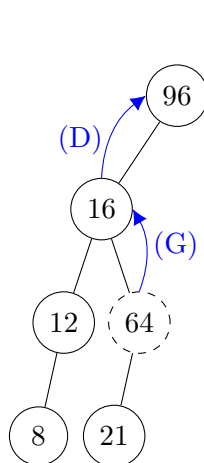




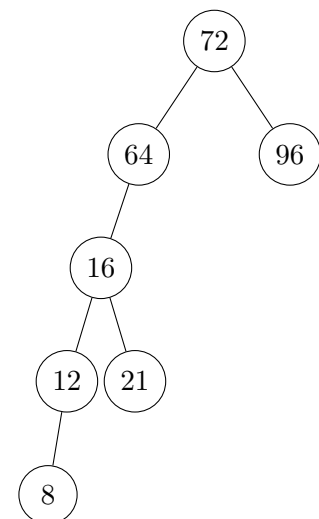
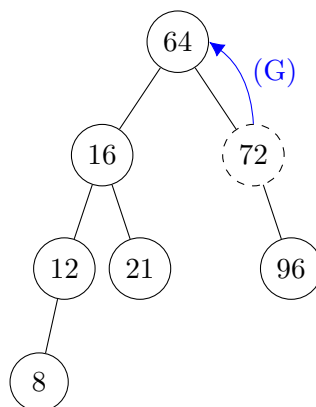
(5) - insertion de 96

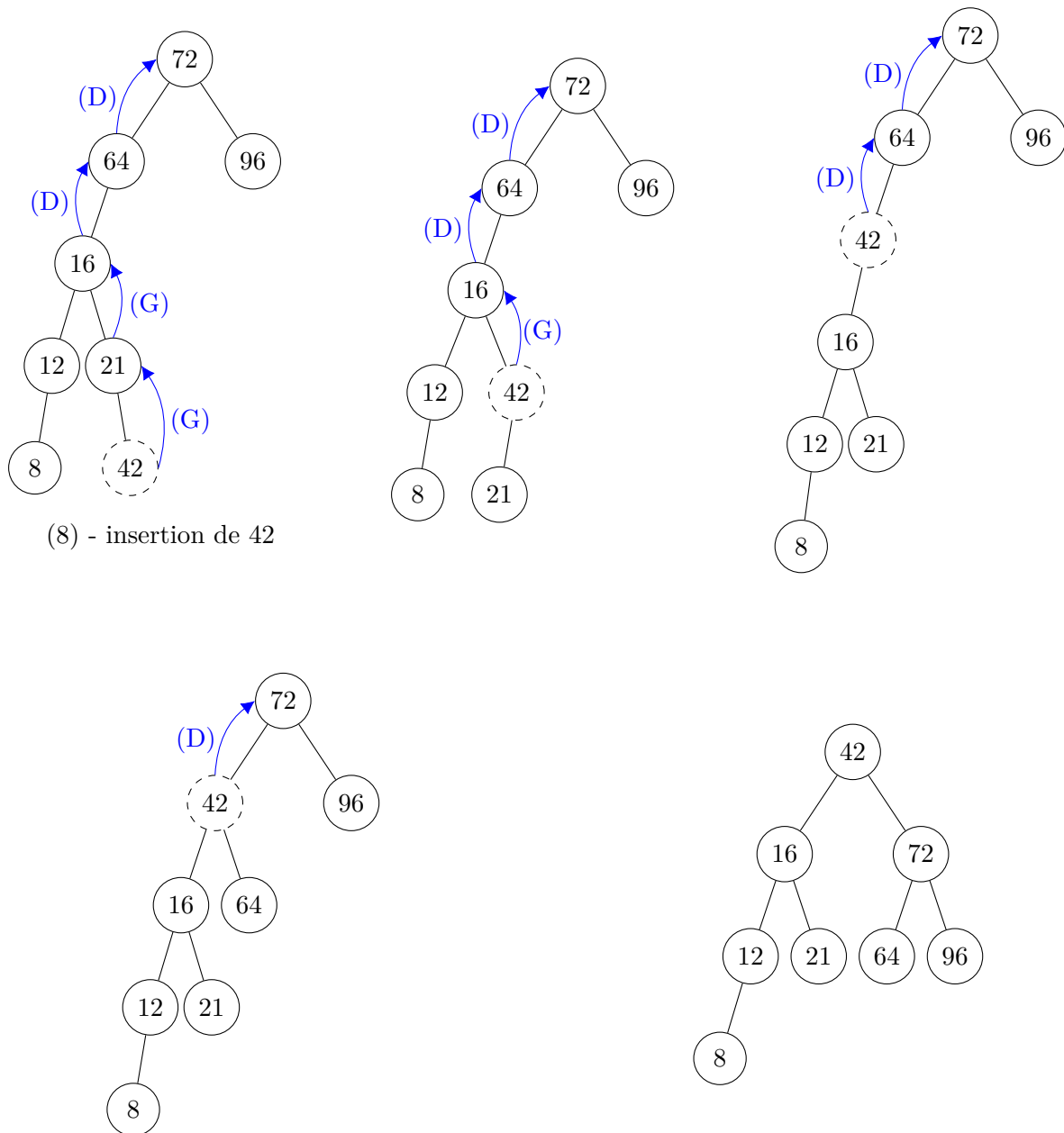


(6) - insertion de 64



(7) - insertion de 72





On remarque qu'à la fin de l'étape 7 (insertion de 72) l'arbre a beaucoup plus de nœuds dans le sous-arbre gauche de la racine (5 nœuds : 64, 21, 16, 12, 8) que dans le sous-arbre droit de la racine (1 nœud : 96). À l'inverse, à la fin de l'étape 8 (insertion de 42), l'arbre est devenu presque complet : seul le dernier niveau n'est pas plein.

Cet effet des rotations est extrêmement important : l'arbre a pu être quasiment rééquilibré après une insertion en feuille. Néanmoins, si on n'avait pas ajouté 42, l'arbre serait resté dans un état très déséquilibré.

Cette notion d'équilibre est essentielle pour optimiser la recherche dans un arbre : plus l'arbre est équilibré, moins il y a de cas extrêmes dans lesquels on pourrait tomber. D'autres contraintes peuvent être ajoutées dans les règles d'ajout/suppression pour permettre de maintenir cet équilibre, et elles seront abordées dans le cours suivant.

Ce document et ses illustrations ont été réalisés par Fabrice BOISSIER en mars 2023. Certaines illustrations sont inspirées des supports de cours de Nathalie "Junior" BOUQUET, et Christophe "Krisboul" BOULLAY.

(dernière mise à jour en mars 2024)