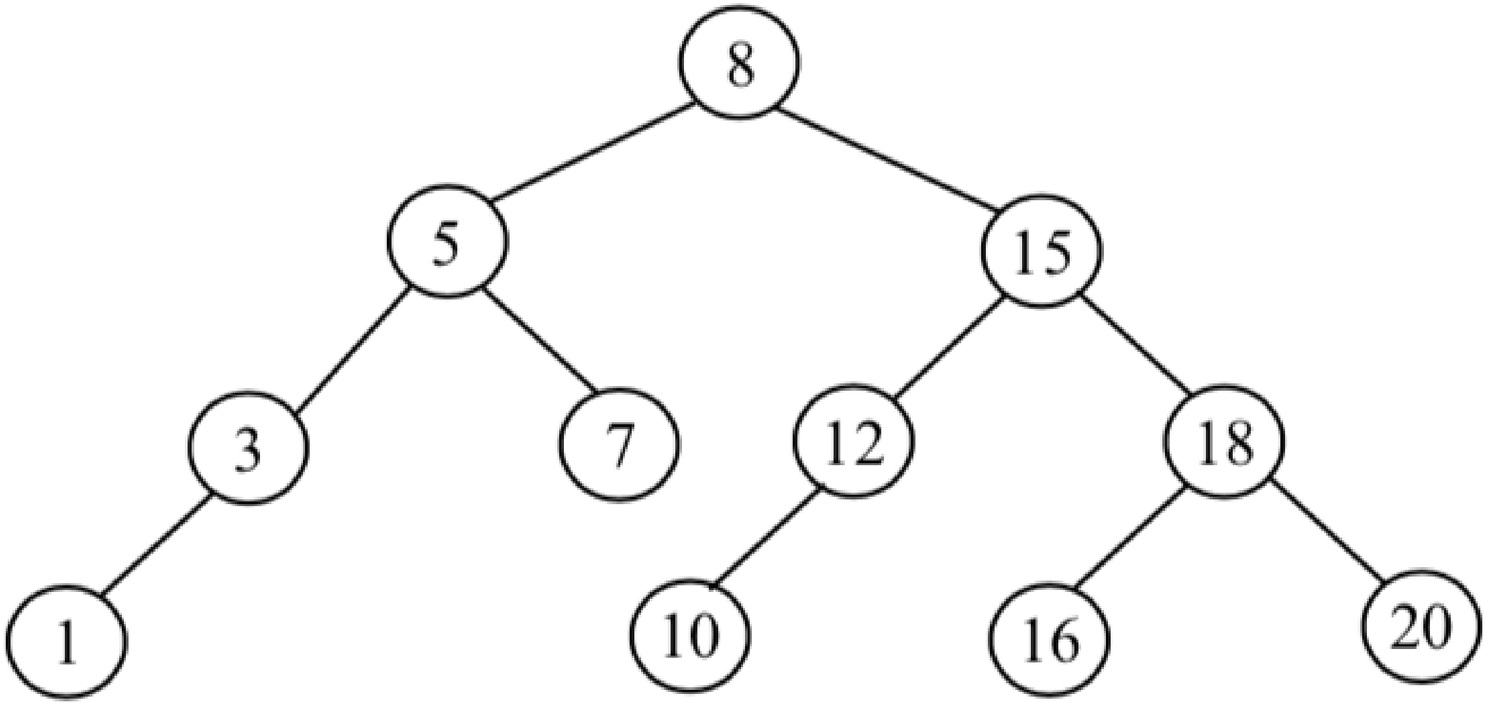
Arbre binaire de recherche

- Un arbre de recherche binaire (BST) est un arbre binaire ordonné.
- P. F. Windley, A. D. Booth, A. J. T. Colin et T. N. Hibbard inventent un arbre de recherche binaire, en 1960.

Définition: un arbre de binaire recherche est soit **vide** ou un arbre avec chaque nœud de l'arborescence contient une clé et

- i) toutes les clés du sous-arbre gauche sont inférieures aux clés du nœud racine,
- ii) toutes les clés du sous-arbre droit sont supérieures aux clés du nœud racine,
- iii)les sous-arbres gauche et droite sont aussi des arbres binaires de recherche.

Arbre binaire de recherche



Arbre binaire de recherche

Les opérations prises en charge par un arbre de recherche binaire sont les suivantes:

- **parcourt**: parcourt exactement une fois tous les nœuds de l'arbre.
- **Insertion**: cette opération insère un nœud dans l'arbre.
- **Effacement**: cette opération supprime un nœud de l'arbre.
- **Recherche**: cette opération recherche une valeur de clé dans l'arbre.
- **Successeur**: cette opération trouve le successeur d'un nœud donné dans l'arbre.
- **Prédécesseur**: cette opération trouve le prédécesseur du nœud donné dans l'arbre.

Parcours d'un ABR

L'algorithme de parcours d'arbre (pré-ordre, post-ordre et en ordre) est le moyen standard de parcourir un arbre de recherche binaire, qui est similaire à la traversée dans un arbre binaire. Dans un arbre de recherche binaire, la traversée dans l'ordre extrait toujours les éléments de données dans un ordre de tri croissant.

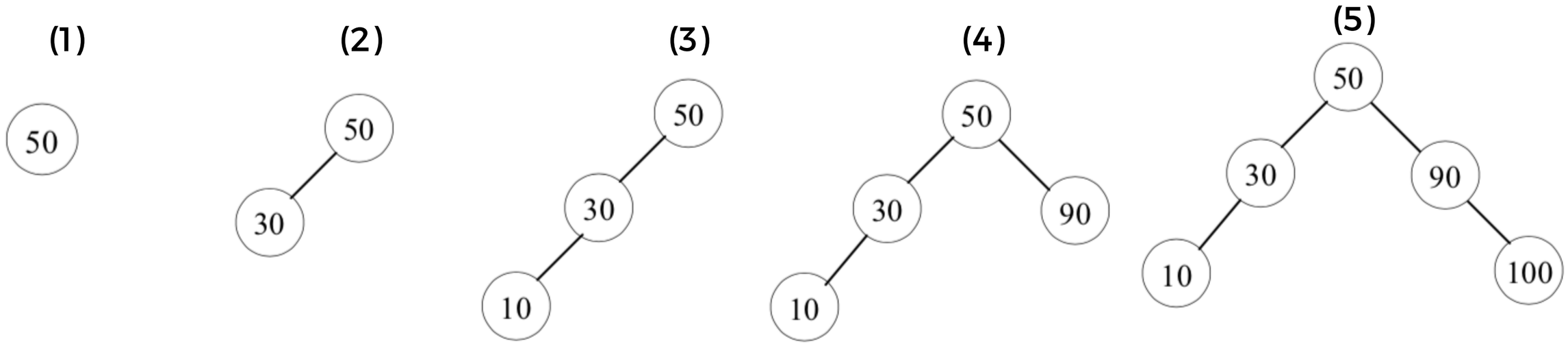
Insertion dans un ABR

- Supposons qu'un nouvel élément de données ayant une clé et l'arbre dans lequel la clé doit être insérée soient donnés en entrée.
- L'opération d'insertion démarre à partir du nœud racine. Si l'arbre est vide, le nouvel élément est inséré en tant que nœud racine.
- Sinon, si l'arbre n'est pas vide, comparer la valeur de la clé avec le nœud racine.
- Si la clé est inférieure au nœud racine, elle est insérée dans le sous-arbre gauche, sinon elle est insérée dans le sous-arbre droit.

Insertion dans un ABR

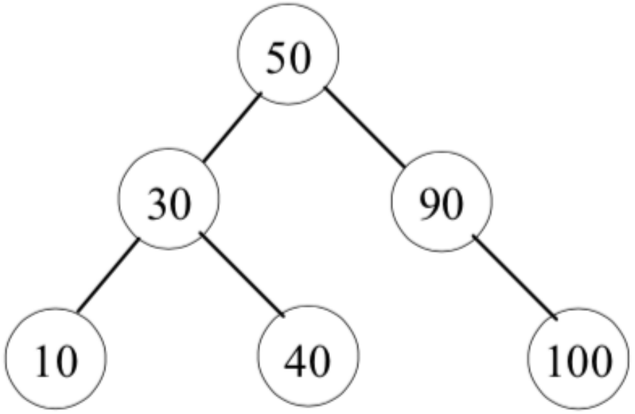
Example:

- Inserer les éléments suivants dans d'arbre binaire de recherche :
  50, 30, 10, 90, 100, 40, 60, 20, 110, 5

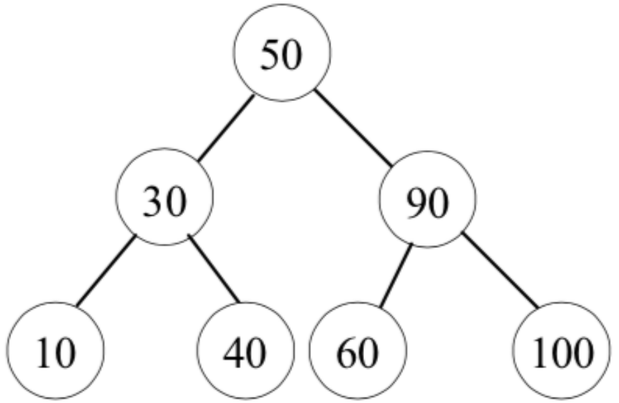


Insertion dans un ABR

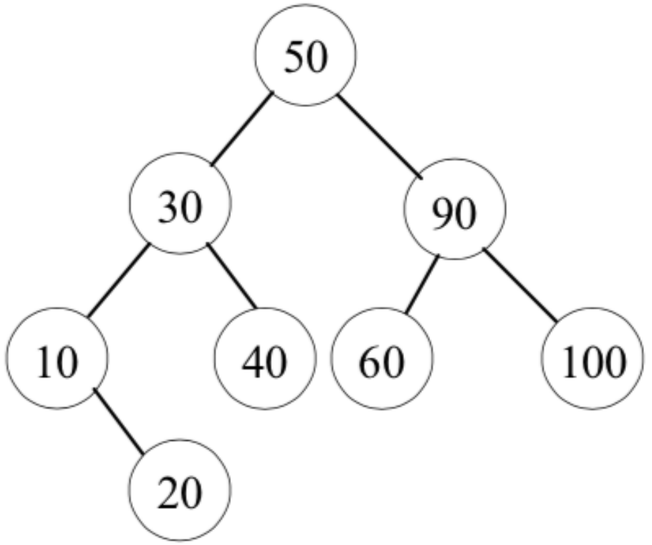
(6)



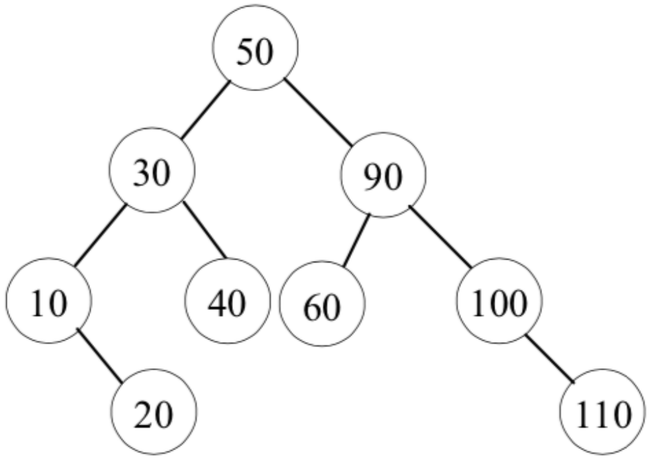
(7)



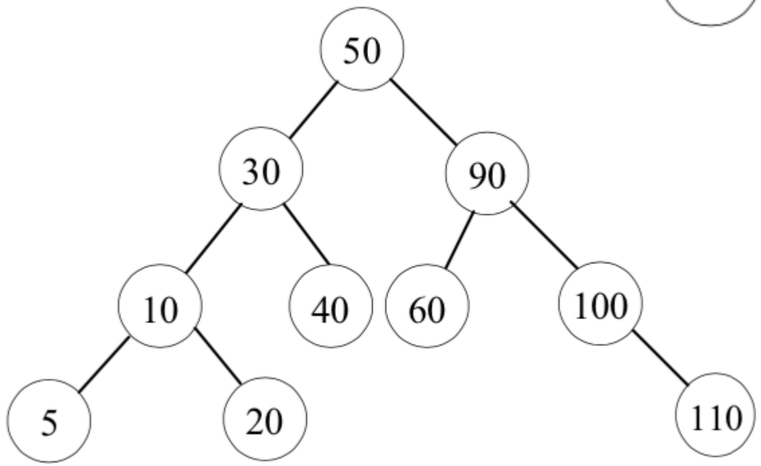
(8)



(9)



(10)



Insertion dans un ABR

```
Algorithm: INSERER (ROOT, DATA)
 1. IF ROOT = NULL THEN
 2.    i)   Allocate Memory for ROOT node.
 3.    ii)  ROOT->INFO=DATA
 4.    iii) ROOT->LCHILD=NULL
 5.    iv)  ROOT->RCHILD=NULL
 6. ELSE
 7.    IF ROOT->INFO>DATA THEN
 8.       CALL INSERT(ROOT->LCHILD, DATA)
 9.    ELSE IF ROOT->INFO<DATA THEN
10.       CALL INSERT (ROOT->RCHILD,DATA)
11. RETURN
```

Recherche dans un ABR

- Supposons qu'une clé et l'arbre dans laquelle la clé est recherchée soient données en entrée. À partir du nœud racine, vérifier si la valeur du nœud actuel est égale ou non à la clé. Dans le cas, lorsqu'un nœud actuel est nul, la valeur de clé recherchée n'existe pas dans l'arbre de recherche binaire. Si le nœud possède la clé recherchée, la recherche réussit.
- Sinon, la clé du nœud actuel est soit inférieure, soit supérieure à la valeur de clé recherchée. Dans le premier cas, toutes les clés du sous-arbre gauche sont inférieures à la valeur de la clé de recherche. Cela signifie que vous n'avez pas besoin de rechercher dans le sous-arbre gauche. Ainsi, il doit rechercher uniquement le bon sous-arbre. De même, dans le second cas, il doit rechercher uniquement le bon sous-arbre.

Recherche dans un ABR

```
Algorithm: BSTSearch (ROOT, DATA, P)
 1. IF ROOT = NULL THEN
 2.    i) PRINT: NOT FOUND
 3.    ii)  P = NULL
 4.    iii) RETURN
 5. IF ROOT->INFO=DATA THEN
 6.    SET P=ROOT
 7.    ELSE IF ROOT->INFO>DATA THEN
 8.       CALL BSTSearch(ROOT->LCHILD,DATA,P)
 9.    ELSE CALL BSTSearch(ROOT->RCHILD,DATA,P)
10. RETURN
```

```
Algorithm: BSTSearch (ROOT, DATA, P)
1. P = ROOT
2. Repeat while P ≠ Null
3.      If DATA = P->INFO then
4.         Return
5.      Else If DATA < P->INFO then
6.         P = P->LCHILD
7.      Else P = P->RCHILD
8. [Fin de Repeat]
9. Return
```

Recherche dans un ABR

Supposons qu'un arbre de recherche binaire contienne n éléments de données.

- Il y a donc n! permutations des n éléments.
- La profondeur moyenne des n! de l'arbre est approximativement c, où c = 1,4.
- Le temps d'exécution moyen f (n) pour rechercher un élément dans un arbre de recherche binaire avec n éléments est proportionnel à, c'est-à-dire f (n) = O ().

Successeur dans un ABR

Dans un arbre binaire, le sécréteur en ordre d'un nœud est le nœud suivant dans la traversée en ordre de l'arbre binaire. Le successeur dans l'ordre est NULL pour le dernier nœud dans la traversée dans l'ordre. Dans l'arbre de recherche binaire, le successeur d'un nœud avec la clé k est une plus petite valeur de clé appartenant à l'arbre et strictement supérieure à k.

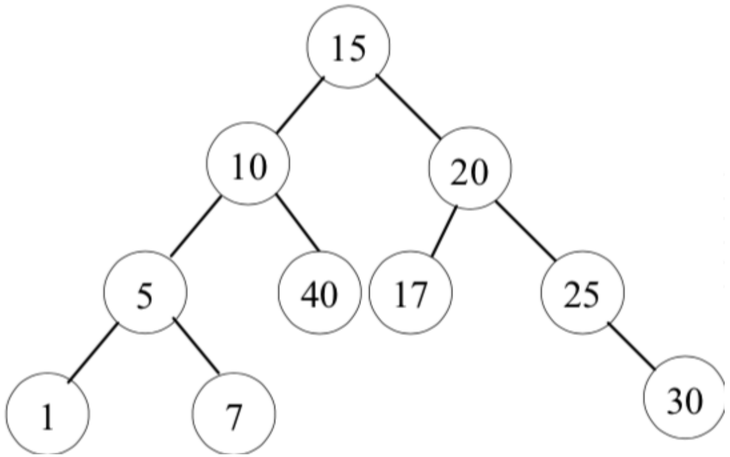
Successeur dans un ABR

L'idée de trouver le successeur d'un nœud donné x:

- i) Si le x a un enfant droit, son successeur en ordre sera l'élément le plus à gauche dans le sous-arbre droit de x (c'est-à-dire le minimum dans le sous-arbre droit de x).
- ii) Sinon, si le **x** n'a pas d'enfant à droite, son successeur en ordre sera celui de ses ancêtres, le successeur en ordre est le nœud le plus éloigné qui peut être atteint à partir de **x** en suivant uniquement les branches droites vers l'arrière.
- iii) Sinon, si **x** est le nœud le plus à droite alors son successeur sera nul.

Successeur dans un ABR

```
Algorithm: BST_SUCCESSOR(X)
 1. If X → RCHILD ≠ NULL then
 2.    Y=X→RCHILD
 3.    Repeat while Y→LCHILD ≠ NULL
 4.       Y = Y → LCHILD
 5.    [End of Repeat]
 6.    Return Y
 7. Else
 8.    Y=X
 9.    Repeat while Parent(X) → RCHILD = X
10.       X = Parent(X)
11.    [End of Repeat]
12.    If Parent(X) ≠ NULL then Return Parent(X)
13.    Else Print: No successor
14.    [End of If]
15. [End of If]
16. Return
```



- Successeur de 1 est 5
- Successeur de 7 est 10
- Successeur de 15 est 17

Suppression dans un ABR

Il y a trois cas possibles à considérer.

- **Cas 1:** lorsque le nœud à supprimer est un nœud feuille, alors supprimer simplement le nœud de l'arbre et mettre **null** sur le lien correspondant du parent.
- **Cas 2**: Lorsque le nœud à supprimer a un enfant, gauche ou droit, alors remplacer simplement le nœud par son enfant unique.

Suppression dans un ABR

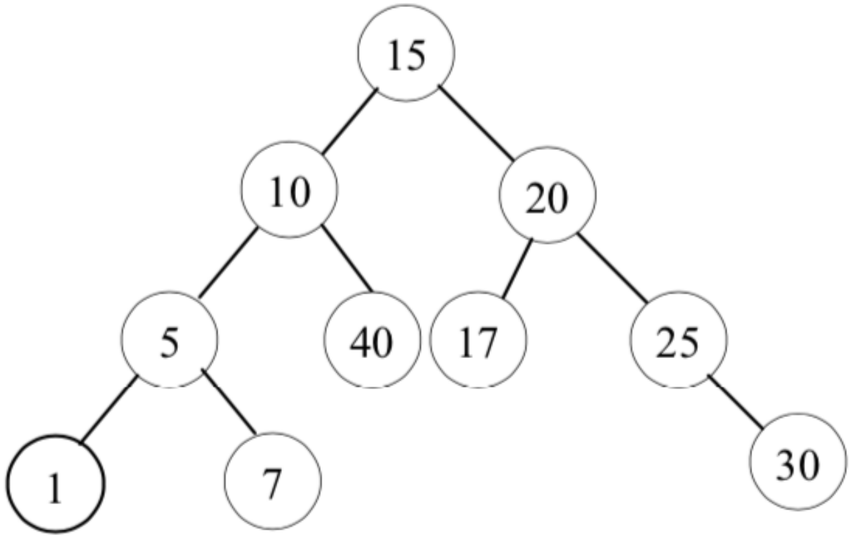
- **Cas 3**: lorsque le nœud a deux enfants
  - sélectionner ensuite son nœud successeur en ordre ou son nœud prédécesseur en ordre (R)
  - copier la valeur de clé du nœud R vers le nœud P,
  - puis supprimer récursivement le nœud R jusqu'à ce qu'il satisfasse l'un des deux premiers cas.

  Dans un arbre binaire, le successeur en ordre (R) d'un nœud n'est que l'enfant le plus à gauche de son sous-arbre droit, car le sous-arbre droit n'est pas nul. Maintenant, le successeur en ordre peut avoir zéro ou un seul enfant de droite, il peut donc être supprimé en utilisant l'un des deux premiers cas.

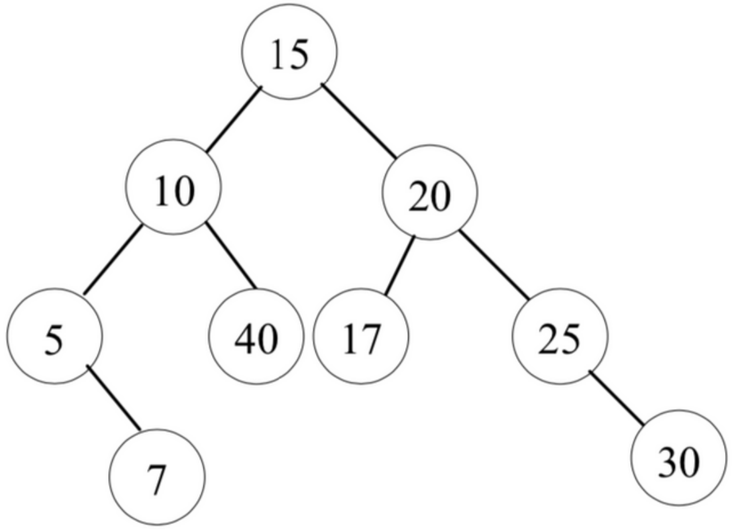
Suppression dans un ABR

```
1. Algorithm: DELETE (ROOT, P, PARENT, DATA)
2. IF P→LCHILD=NULL AND P→RCHILD=NULL THEN
3.    i)IF PARENT→LCHILD=P THEN
4.          SET PARENT→LCHILD=NULL
5.    ii) ELSE SET PARENT→RCHILD=NULL
6. ELSE IF P→LCHILD=NULL THEN
7.    i) IF PARENT→LCHILD=P THEN
8.          SET PARENT→LCHILD=P→RCHILD
9.    ii) ELSE SET PARENT→RCHILD=P→RCHILD
10. ELSE IF P→RCHILD=NULL THEN
11.   i) IF PARENT→LCHILD=P THEN
12.         SET PARENT→LCHILD=P→LCHILD
13.   ii) ELSE SET PARENT→RCHILD=P→LCHILD
13.
14. ELSE P→LCHILD≠NULL AND P→RCHILD≠NULL THEN
15.   i) SET PARENT = P
16.   ii) SET IN=P→RCHILD
17.   iii) REPEAT WHILE IN→LCHILD≠ NULL
18.         a) SET PARENT = IN
19.         b) SET IN=IN→LCHILD
20.   [END OF LOOP]
21.   iv) SET P→INFO=IN→INFO
22.   v) SET P = INvi) Call DELETE(ROOT, P,
   PARENT, DATA) [END OF IF]
23. Deallocate memory for P node.
24. RETURN.
```

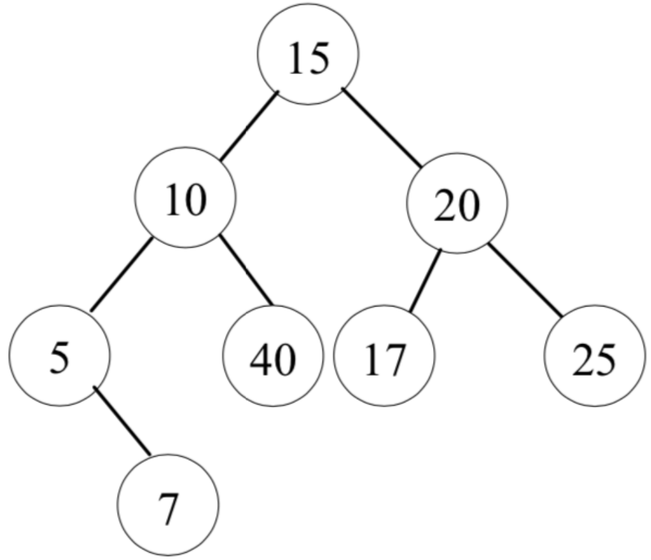
Suppression dans un ABR



Supprimer les éléments suivants de l'ABR:
1, 30, 5, 15

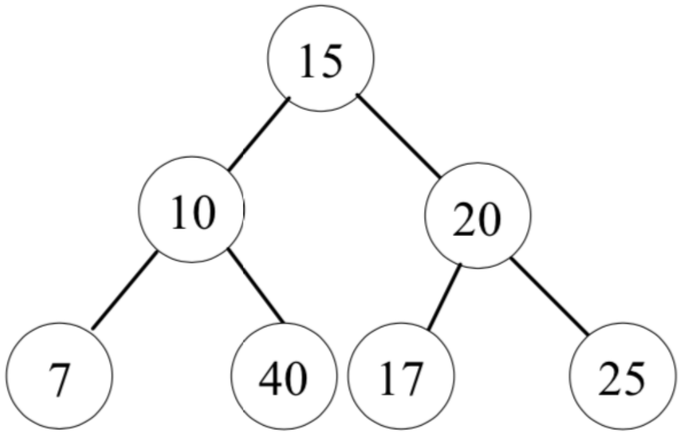


Supprimer 1: Ici, le nœud supprimé est l'enfant gauche de son nœud parent. Par conséquent, après le suppression, nous obtenons PARENT → LCHILD = NULL ,

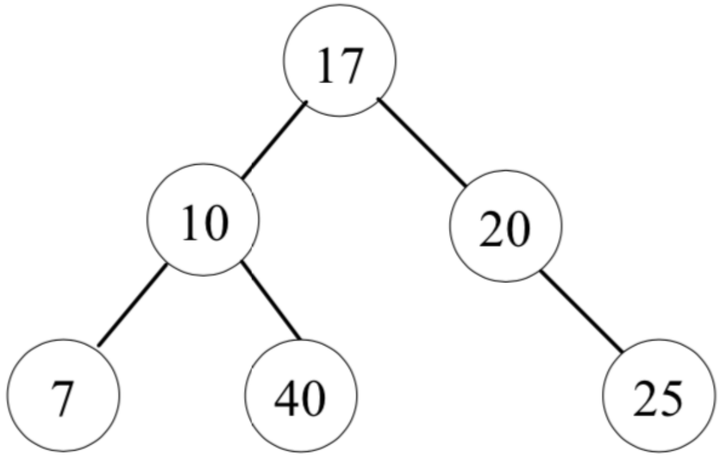


Supprimer 30: Ici, le nœud supprimé est l'enfant droit de son nœud parent. Par conséquent, après avoir supprimé, nous obtenons
PARENT → RCHILD = NULL.

Suppression dans un ABR



Supprimer 5: Ici, le nœud supprimé est l'enfant gauche de son nœud parent et il a un sous-arbre droit. Par conséquent, après avoir supprimé, nous obtenons PARENT → LCHILD = P → RCHILD,



Supprimer 15: Ici, le nœud supprimé a deux sous-arbres. Dans un premier temps, rechercher le successeur en ordre du nœud à supprimer, puis remplacer la clé du nœud à supprimer par la clé de son successeur en ordre. Enfin, supprimez le successeur inorder.

Complexité

- Sur un ABR de hauteur **h**, différentes opérations telles que recherche, minimum, maximum, successeur, prédécesseur, insertion et suppression peuvent être effectuées en un temps **O(h)**.

- En moyenne, les arbres de recherche binaire avec n nœuds ont une hauteur **log(n)** et dans le pire des cas, les arbres de recherche binaire peuvent avoir **n** comme hauteur. Par conséquent, différentes opérations comme la recherche, le minimum, le maximum, le successeur, le prédécesseur, l'insertion et la suppression prennent **O(log(n))** temps dans le cas moyen et **O(n)** dans le pire des cas.

Complexité

- Les ABR sont une structure de données de base utilisée pour construire des structures de données abstraites telles que des ensembles, des multi-ensembles, des tableaux associatifs. Pour trier une séquence de nombres, dans un premier temps, tous les nombres doivent être insérés dans un nouvel ABR, puis traverser l'arbre dans l'ordre.

Résumé ABR

Avantages de l'ABR

- i) L'algorithme de tri et de recherche peut être très efficace.
- ii) Facile à coder la plupart des opérations effectuées sur l'arbre de recherche binaire.

Inconvénients de l'ABR

- i) La forme de l'ABR dépend entièrement de la séquence des opérations d'insertion et de suppression qui peuvent entraîner une asymétrie.
- ii) La hauteur de l'ABR est beaucoup plus élevée que dans la plupart des cas, par conséquent, le temps d'exécution peut augmenter.
- iii) Comme l'ABR n'est pas un arbre équilibré, le temps d'exécution de la plupart des opérations est O (n) dans le pire des cas.

Les tas

Un tas est un arbre binaire qui doit satisfaire les propriétés suivantes:

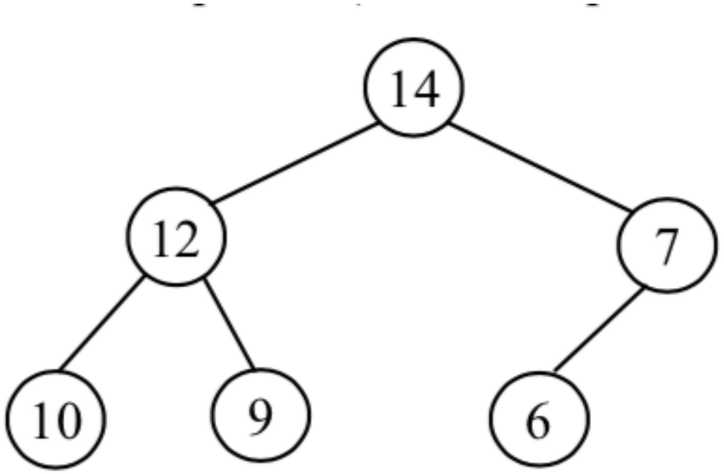
- i) L'arbre binaire essentiellement complet, ce qui signifie que l'arbre a complètement rempli tous les niveaux, le dernier niveau peut être partiellement rempli de gauche à droite et certaines feuilles à l'extrême droite peuvent manquer.
- ii) Toutes les clés de l'arborescence, à l'exception du nœud racine, sont supérieures/inférieures ou égales à la clé du nœud parent.

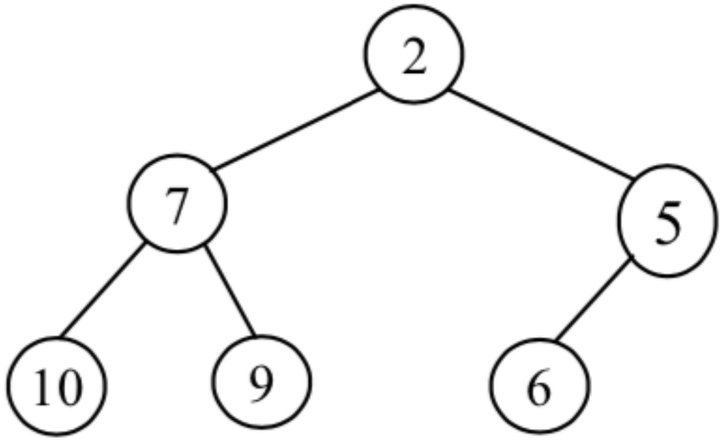
Les tas

Il existe deux types de tas:

i) tas max

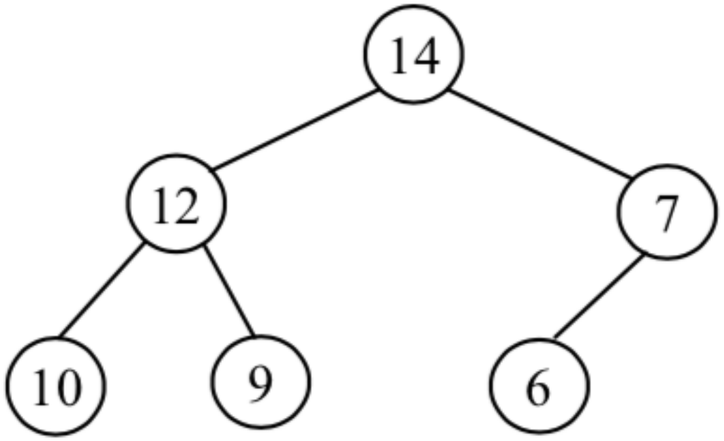
ii)tas min



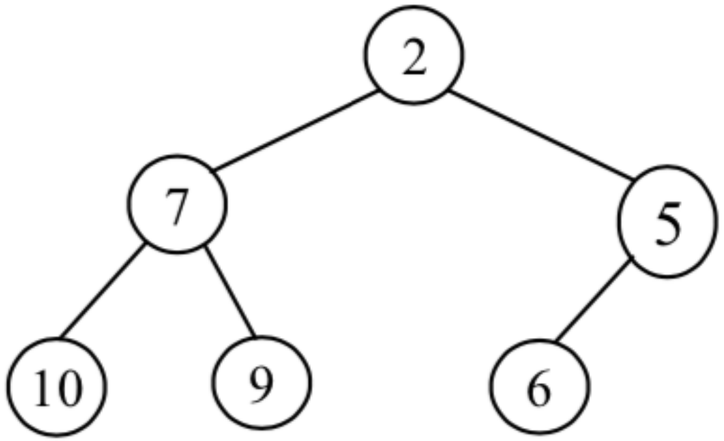


Les tas

- Un **tas max** est défini comme un arbre binaire complet avec la propriété que la clé de chaque nœud est supérieur ou égal aux clés de ses nœuds enfants.
- Un **tas min** est également un arbre binaire complet avec la propriété que la clé de chaque nœud est inférieure ou égale aux clés de ses nœuds enfants.



tas max



tas min

Les tas

- **Heapify**: Cette opération restaure la condition de segment de mémoire. Par exemple, si un nœud a changé dans l'arbre, la condition de segment de mémoire n'est plus valide. Il doit ensuite restaurer la condition en déplaçant les nœuds vers le haut ou vers le bas de l'arbre.
- **Insertion**: Cette opération insère un nœud dans le tas.
- **Effacement**: Cette opération supprime un nœud du tas.
- **Décale vers le haut**: Cette opération déplace un noeud vers le haut dans l'arbre, aussi longtemps que nécessaire (en fonction de la condition de segment de mémoire: segment de mémoire minimal ou segment de stockage maximal).
- **Rétrograder**: Cette opération déplace un nœud vers le bas dans l'arbre.

Les tas

Application des tas

- □ Systèmes d'exploitation - Planification des tâches/processus
- □ Tri par tas
- □ Application graphique
- □ File d'attente prioritaire