Les arbres binaires de recherche

Rajae El Ouazzani

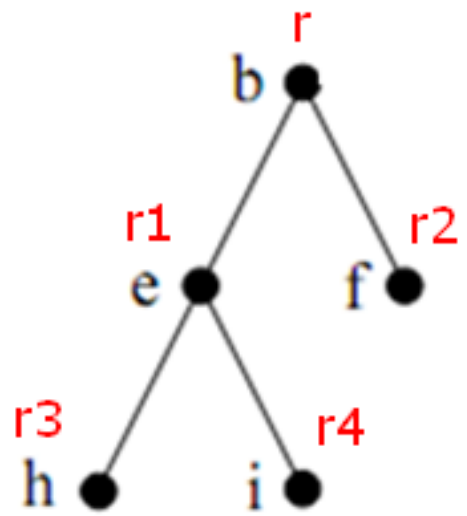
Chapitre 9: Les arbres binaires de recherche

Plan

- Définition
- Vocabulaire et propriétés
- Les arbres binaires de recherche: ABR
- L'usage des arbres
- Traitement sur les arbres binaires de recherche

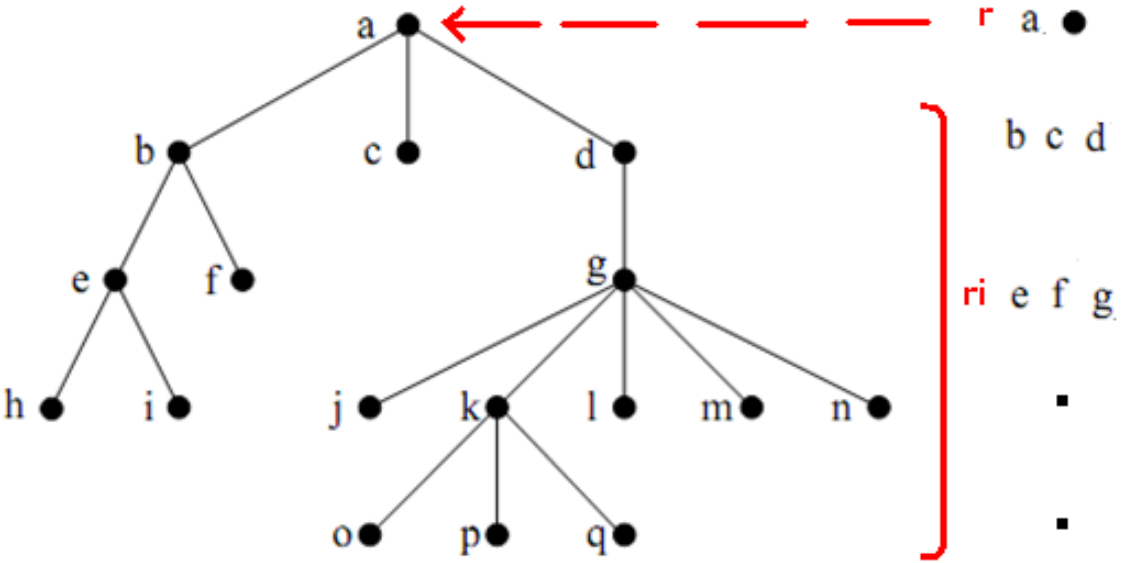
1- Définition de l'arborescence

- Une arborescence est une collection de nœuds reliés entre eux par des arcs.
- Une arborescence contient un nœud particulier **r** appelé racine et une séquence de (sous-) arborescences de racines respectives $r_1, r_2, \ldots, r_k$.
- La racine r est reliée aux r_i par des arcs orientés du haut en bas (de r vers r_i).

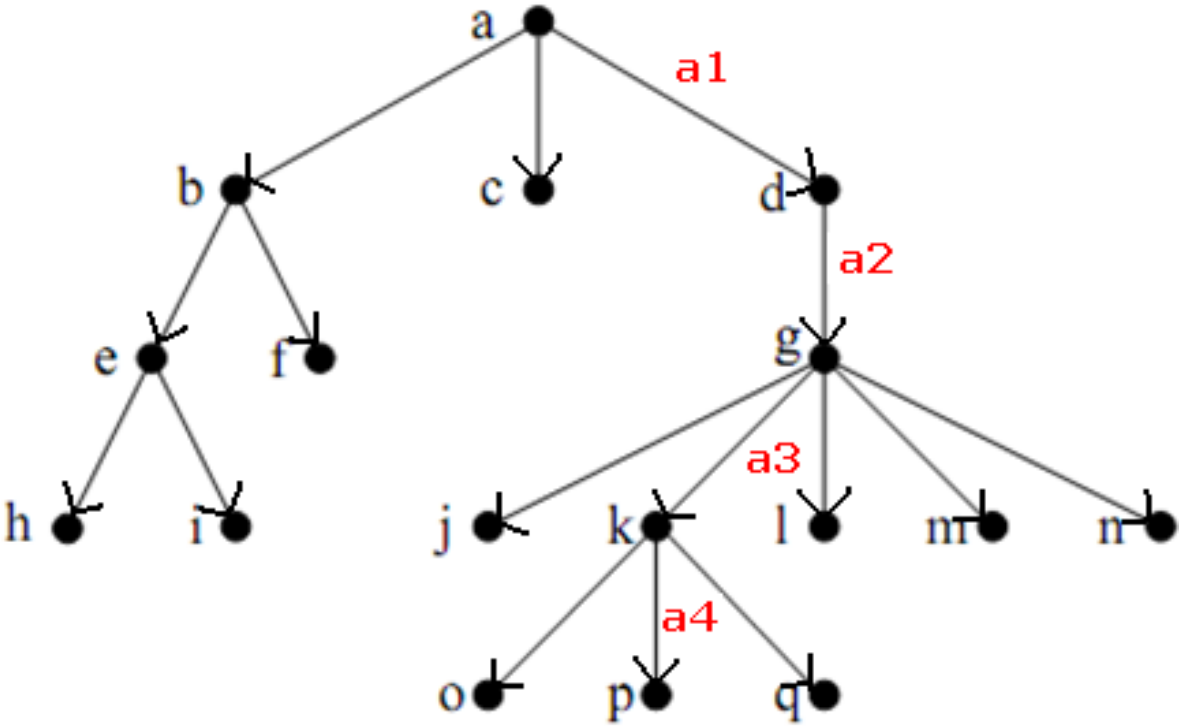


2- Vocabulaire et propriétés

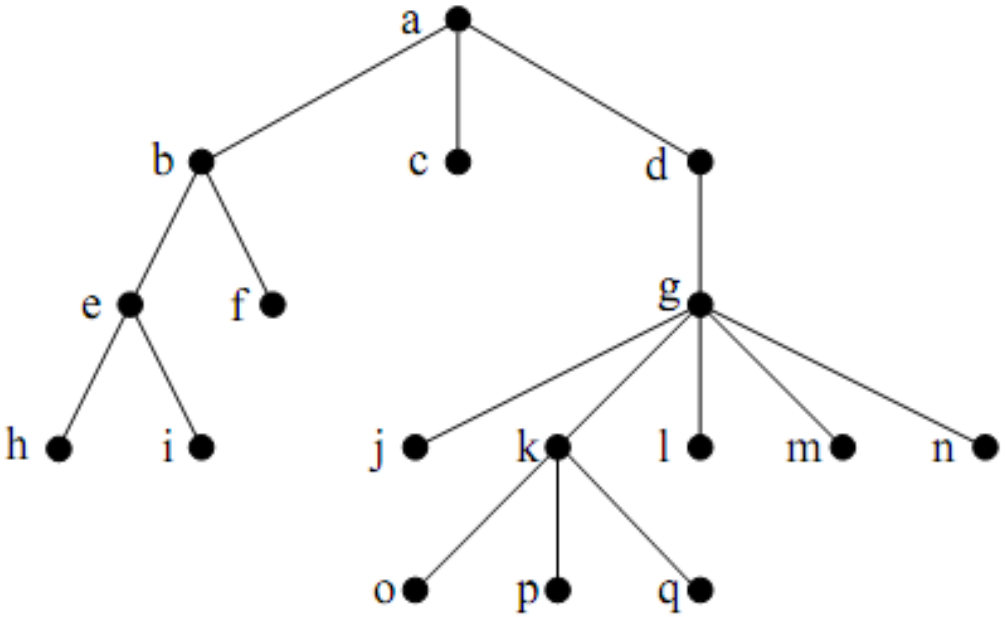
- Cette arborescence a une racine ‘a’.
- Les r_i (b, c, d) sont les fils de r (a).
- Les fils du nœud g sont: j, k l, m et n. Le père de j est g.
- Les nœuds qui n’ont pas de fils sont appelés feuilles.
- Les feuilles de cette arborescence sont: h, i, f, c, j, o, p, q, l, m, n.



- Une séquence de nœuds partant du nœud 'a' en suivant l'orientation des arcs jusqu'au nœud 'p' s'appelle un chemin de a à p.
- La longueur d'un chemin est le nombre d'arcs sur le chemin.
- Il existe exactement un seul chemin de la racine à chaque nœud de l'arborescence.



- La hauteur d'un nœud est la longueur du plus long chemin partant de ce nœud et aboutissant à une feuille.
- Les hauteurs respectives des nœuds b, c, d sont 2, 0, 3.
- La hauteur de l'arborescence est la hauteur de sa racine.
- L'arité d'un nœud est le nombre de ses fils.



3- Les arbres binaires de recherche: ABR

- Un **ABR** est une arborescence binaire représentée sous une forme chaînée et chaque nœud de l'arbre contient :
  - un ou plusieurs champ(s) donnée(s),
  - deux champs **droit** et **gauche**, ce sont des pointeurs vers les deux fils du nœud,
  - un champ **père** optionnel, c'est un pointeur vers le père du nœud.

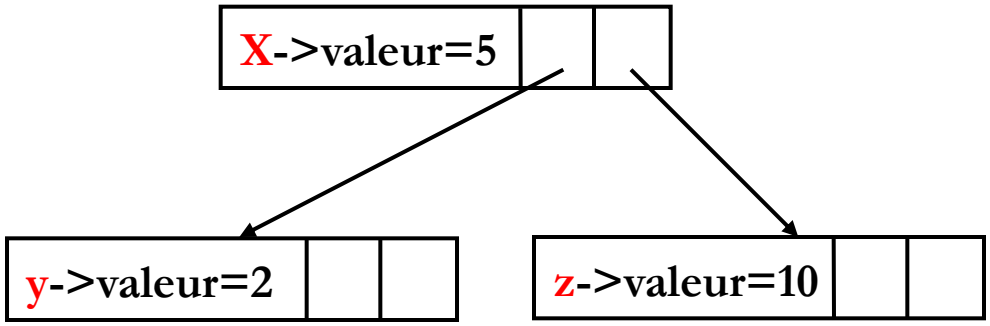
NB:

- Lorsque les nœuds droit ou gauche sont vides, les champs correspondants valent NULL.
- Le nœud racine est le seul pour lequel le champ père est NULL.

- **L'ABR possède de plus une propriété fondamentale sur les valeurs des nœuds fils.**

Exemple:

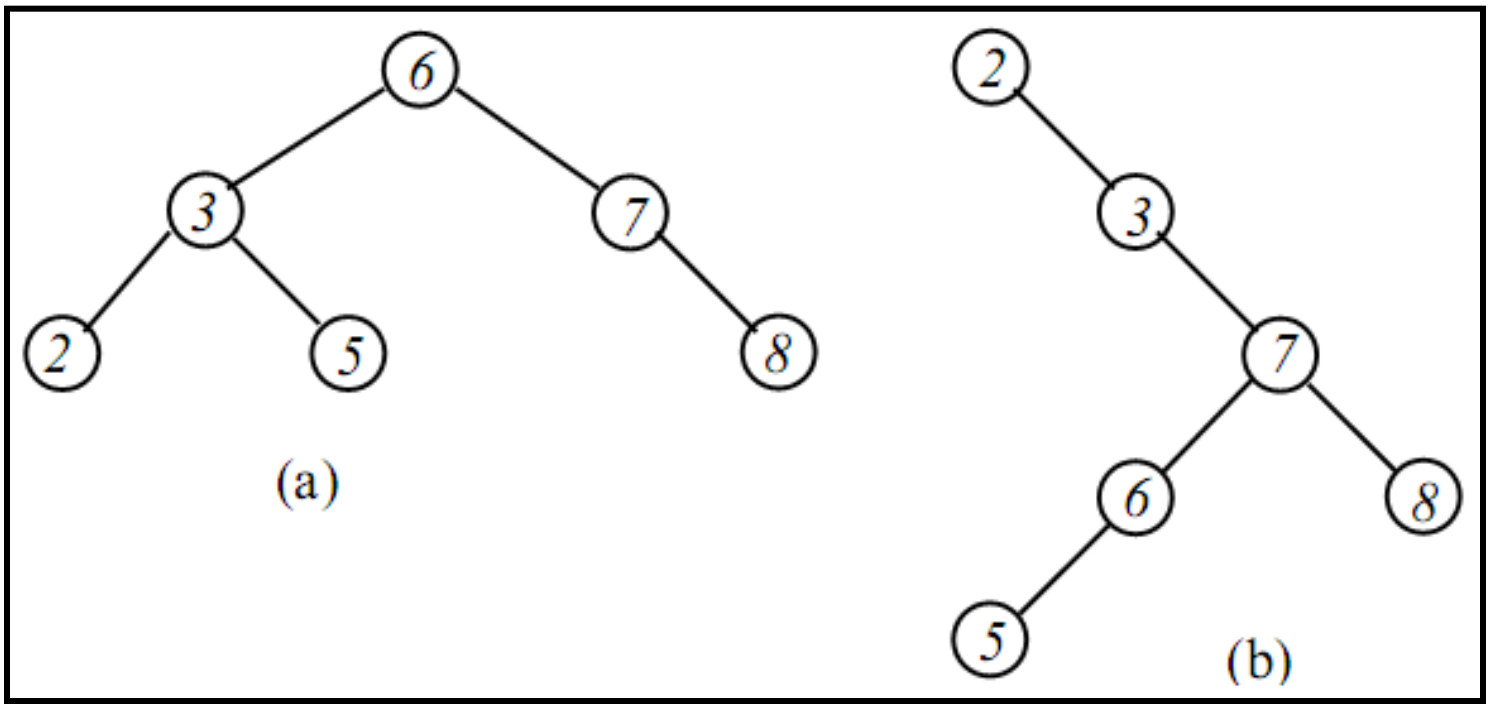
- **Soit x un nœud quelconque d'un ABR. Pour tout nœud y du sous-arbre gauche de x, la valeur de y est inférieure ou égale à la valeur de x, et pour tout nœud z du sous-arbre droit de x, la valeur de z est supérieure ou égale à la valeur de x.**



Exemple d'ABR

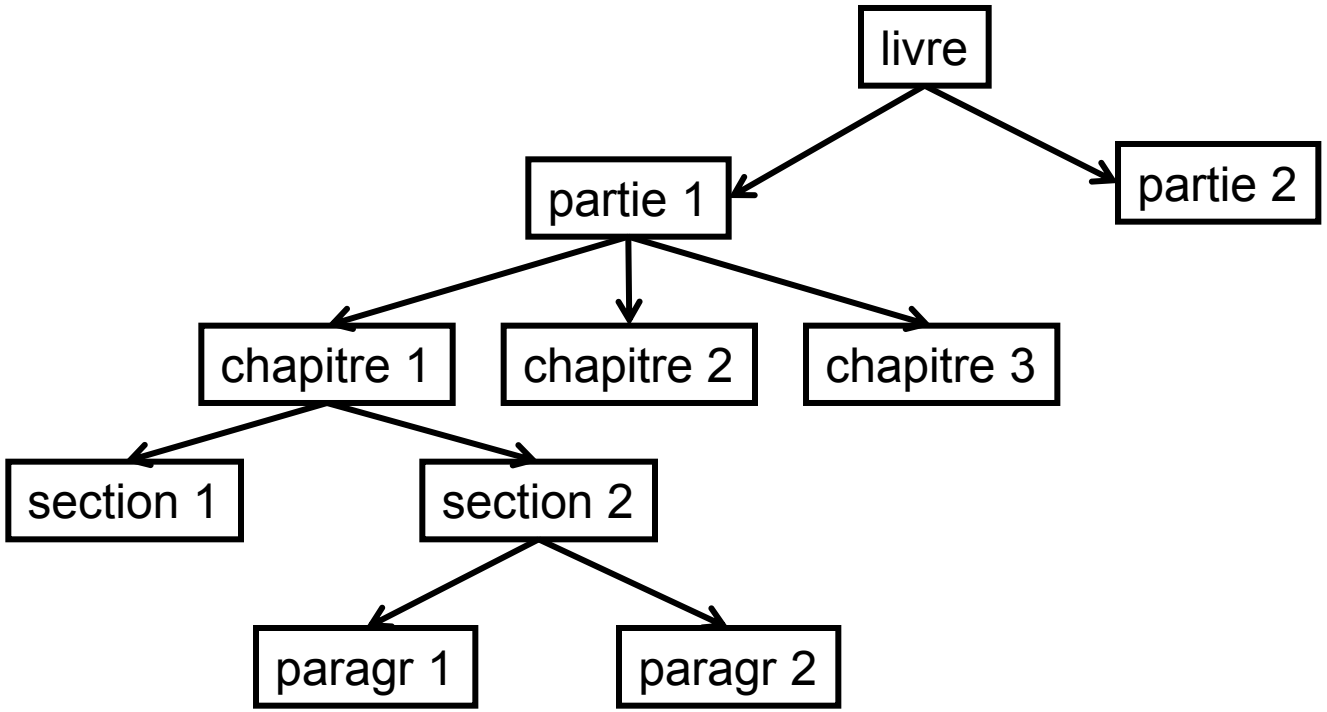
- La figure suivante présente deux exemples d'ABR qui possèdent la même collection de valeurs.

- L'ABR résultat dépend de l'ordre d'insertion des nœuds dans l'arbre.



4- L'usage des arbres

- Les arbres sont utiles pour représenter les données de manière hiérarchique. Exemples :
  - découpage d'un livre en parties, chapitres, sections, paragraphes, etc,
  - hiérarchies des répertoires: dossiers, sous dossiers, fichiers.
  - etc.



Traitement sur les arbres binaires de recherche

Plan

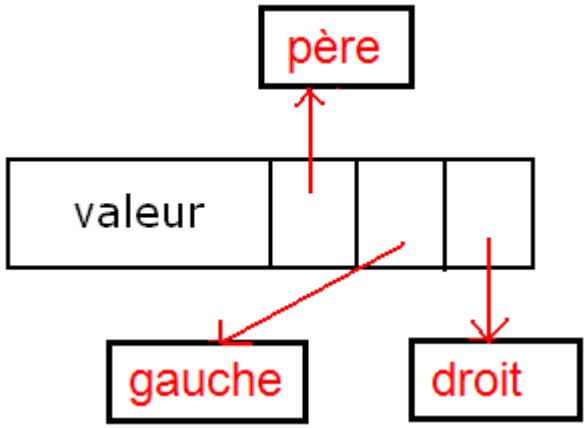
- Nous allons traiter un ABR d'entiers.

1) **Déclarer** un nœud de l'ABR.
2) **Créer la racine** d'un ABR.
3) **Insérer** un nœud dans un ABR.
4) **Rechercher** un nœud dans un ABR.
5) **Afficher** les nœuds d'un ABR: Les modes de parcours des nœuds d'un ABR.

1- Déclaration d'un nœud d'un ABR

- La déclaration d'un nœud de l'arbre est la suivante:

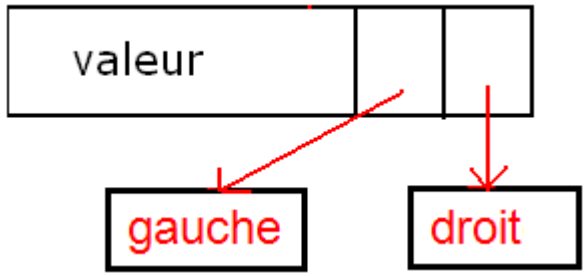
```
typedef struct Noeud{
    int valeur ;
    struct Noeud * père;  //optionnel
    struct Noeud * gauche ;
    struct Noeud * droit ;
} Noeud;
```



- Le pointeur vers le nœud père est optionnel. Il sert simplement à simplifier l'écriture et la complexité de certains algorithmes.

Ecriture simplifiée d'un nœud d'un ABR

```
typedef struct Noeud{
    int valeur ;
    struct Noeud * gauche ;
    struct Noeud * droit ;
} Noeud;
```



2- Création de la racine d'un ABR d'entiers

- La création de la racine d'un ABR d'entiers revient à donner des valeurs initiales aux différents champs de la structure Noeud.

- Nous allons suivre les étapes suivantes:

1) Déclarer et allouer la mémoire pour le nœud racine.
2) Mettre le nombre que l'on veut ajouter dans le champ valeur.
3) Mettre les pointeurs gauche et droit à NULL. Au départ, il y'a un seul nœud.
4) Retourner la racine de l'ABR.

```
//Création de la racine d'un ABR
Noeud * creer_noeud(int n){
    Noeud * noeud= (Noeud *)malloc(sizeof (Noeud)); //étape 1
    noeud ->valeur = n ;                              // étape 2
    noeud ->gauche = NULL;                            // étape 3
    noeud ->droit = NULL;                             // étape 3
    return noeud;                                     // étape 4
}
```

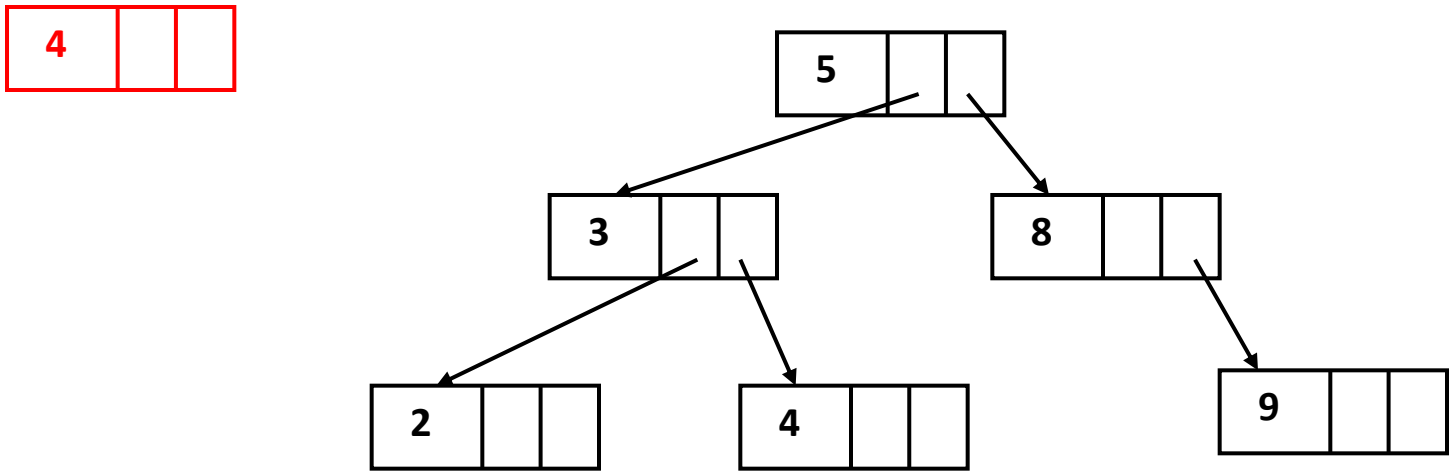
- n est la valeur du nœud.

Insertion d'un nœud dans un ABR

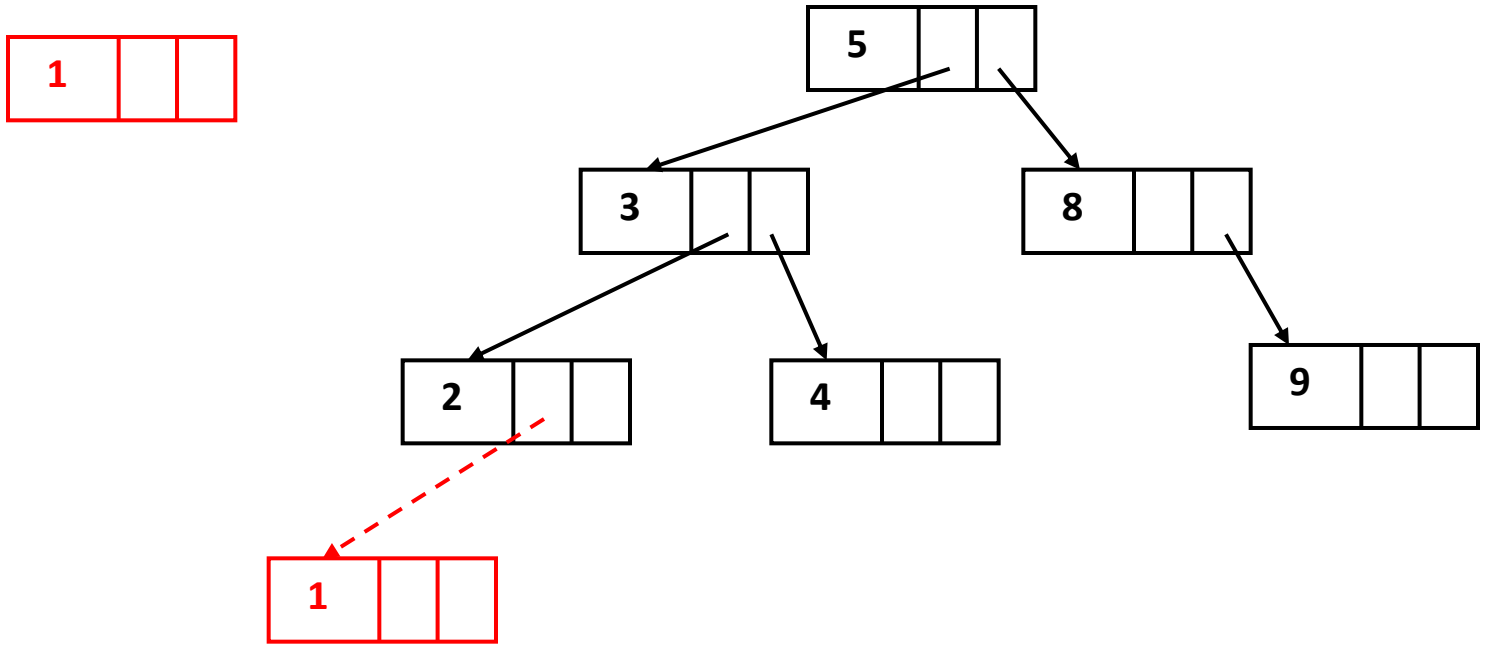
3- Insertion d'un nœud dans un ABR

- Nous allons traiter les 3 cas suivants:
  - **Cas 1:** La valeur du nouveau nœud se trouve déjà dans l'arbre.
  - **Cas 2:** Insérer un nouveau nœud à gauche de l'ABR.
  - **Cas 3:** Insérer un nouveau nœud à droite de l'ABR.

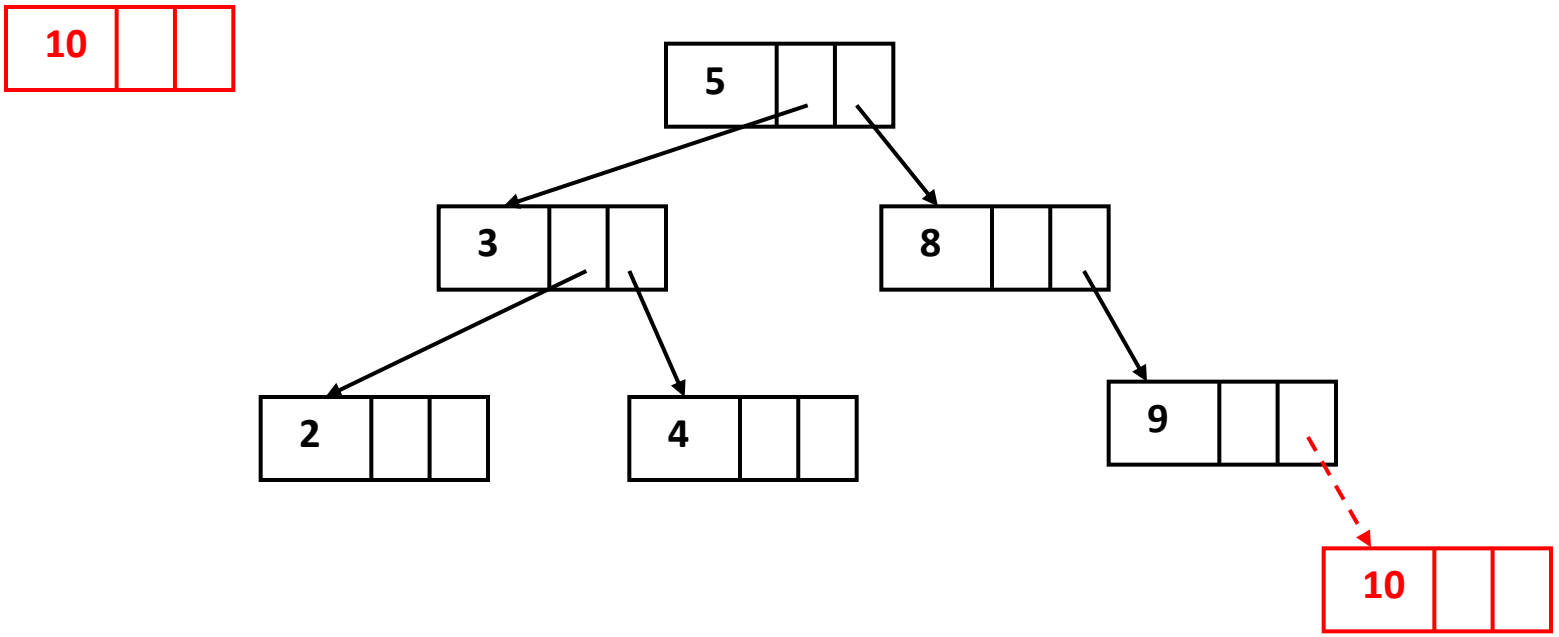
Cas 1: La valeur du nouveau nœud se trouve déjà dans l'arbre.



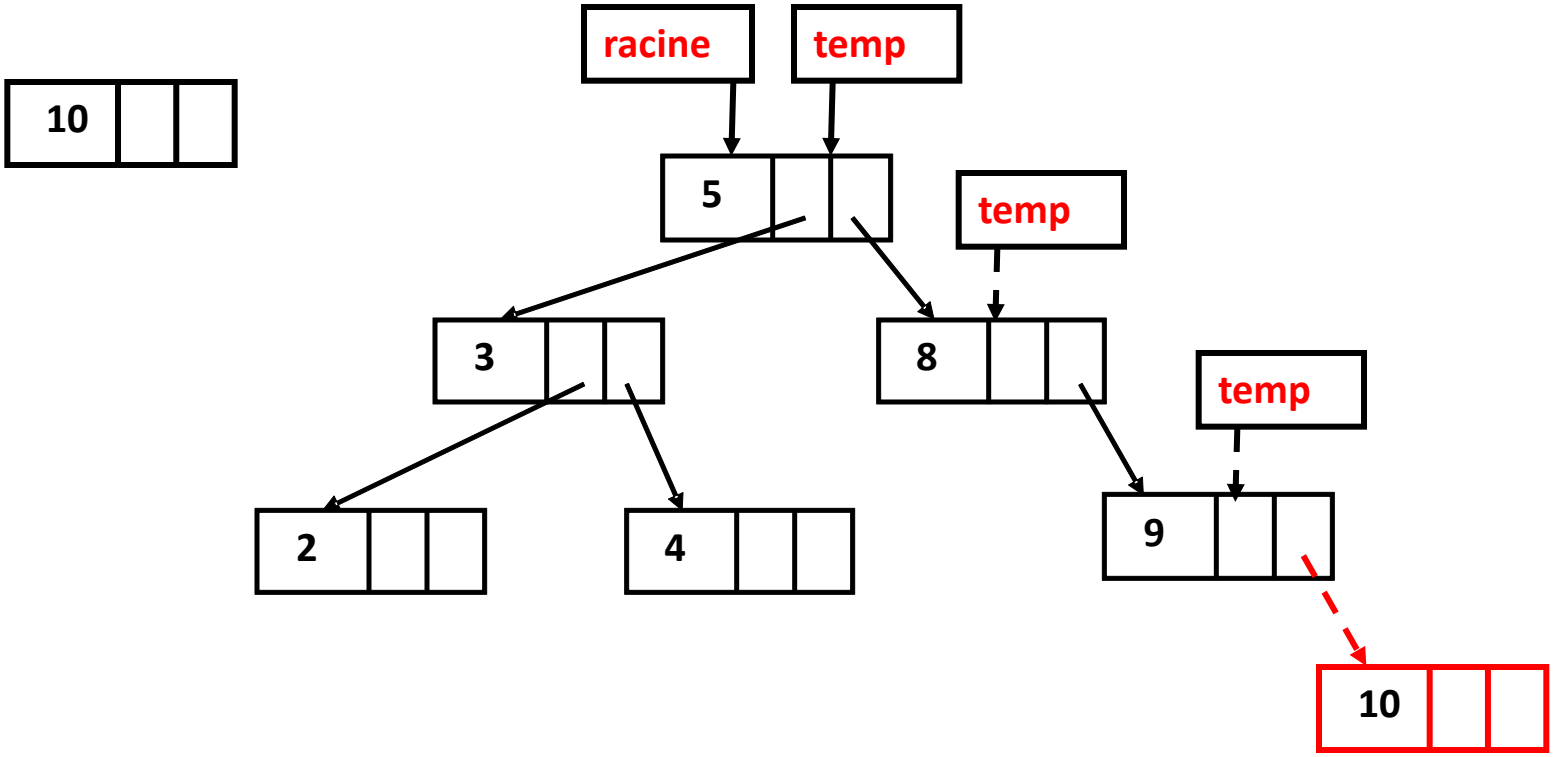
Cas 2: Insérer un nouveau nœud à gauche de l'ABR:



Cas 3: Insérer un nouveau nœud à droite de l'ABR:



Explications



Etapes :

1) Créer un nouveau nœud et lui allouer la mémoire.
2) Rechercher de façon récursive le bon emplacement de la valeur: Comparer la valeur à ajouter avec la valeur de temp.
   a. Si nouveau->valeur<temp->valeur: chercher dans le sous arbre gauche.
   b. Si nouveau->valeur>temp->valeur: rechercher dans le sous arbre droit.
   c. Si nouveau->valeur==temp->valeur: on affiche le message: "Valeur existe déjà"
   d. Si la valeur ne figure pas dans l'arbre, on l'ajoute.

Ajouter un nouveau nœud

```
Noeud * ajouter_noeud ( int n , Noeud * racine ){
    Noeud * nouveau = creer_noeud (n ) ;              // étape 1
    return ajout_recu (nouveau, racine, racine) ;      //étape 2
}
```

```
Noeud * ajout_recu(Noeud * nouveau , Noeud * racine, Noeud * temp ){
  if ( racine== NULL)            // arbre vide
     return nouveau ;
  if ( nouveau->valeur == temp->valeur )  {   // la valeur se trouve à la racine
     printf("\n La valeur existe déjà \n");
     return racine;
  }

  else if ( nouveau->valeur <temp->valeur ){   // recherche au sous-arb gauche
     if ( temp->gauche == NULL)
        temp->gauche = nouveau ;
     else
        ajout_recu ( nouveau , racine , temp->gauche ) ;
  }
```

```
  else {    //nouveau->valeur > temp->valeur
     if ( temp->droit == NULL)
        temp->droit = nouveau ;
     else
        ajout_recu( nouveau , racine , temp->droit) ;
  }
  return racine ;
}
```

Recherche d'une valeur dans un ABR

4- Recherche d'une valeur dans un ABR

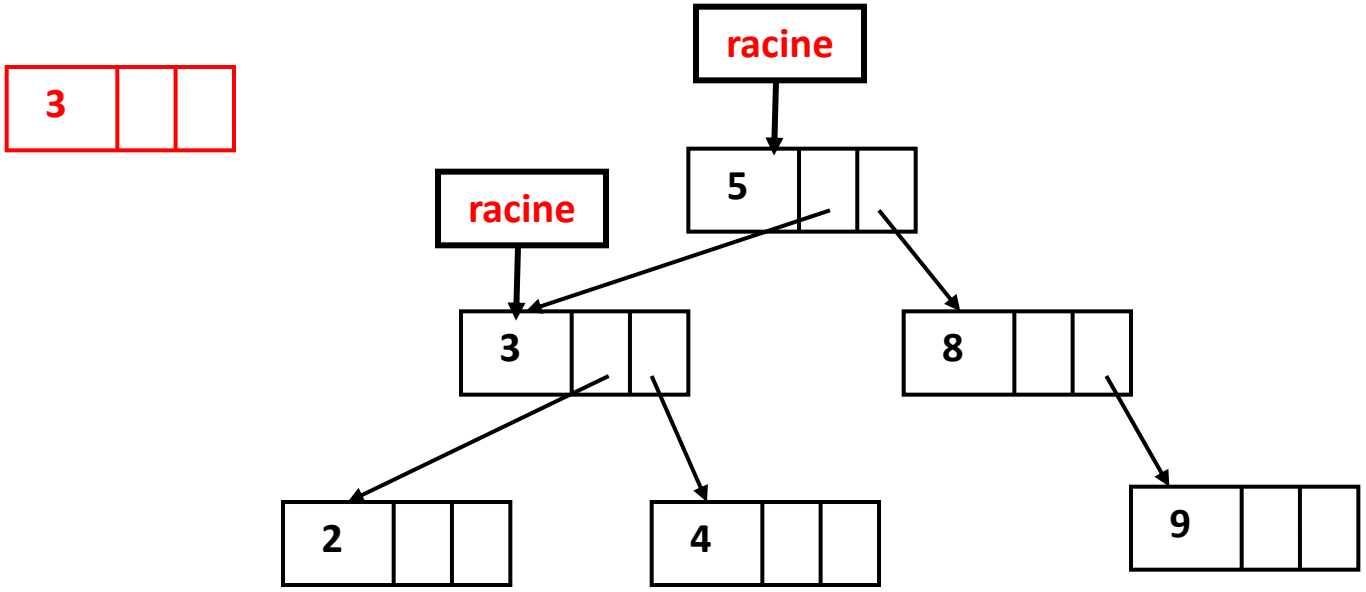
- Nous pouvons réaliser la recherche dans l'ABR de façon itérative ou bien récursive.

- Les 2 fonctions retournent l'adresse du nœud qui contient la valeur recherchée et NULL sinon.

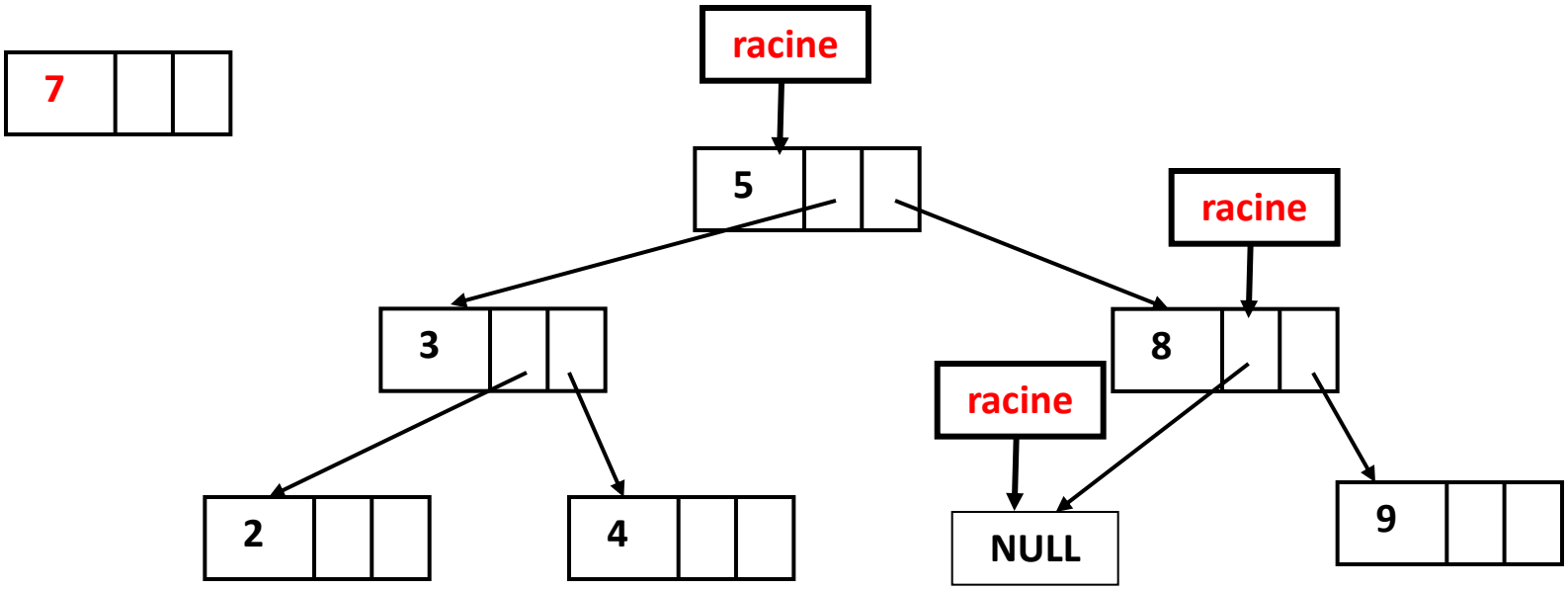
Recherche itérative dans un ABR

- La fonction itérative utilise une boucle tant que pour parcourir tous les nœuds de l'arbre et comparer la valeur recherchée avec les valeurs des nœuds à partir de la racine. La fonction :
  - retourne l'adresse de la racine qui contient la valeur;
  - continue la recherche de la valeur avec le fils droit ou gauche en fonction du résultat de la comparaison de la valeur recherchée avec le nœud racine.
  - retourne NULL si la valeur ne se trouve pas dans l'arbre.

- **Exemple 1:** Rechercher la valeur 3 dans l'arbre.



- **Exemple 2:** Rechercher la valeur 7 dans l'arbre:



- Fonction de recherche itérative:

```
Noeud* rechercher_valeur_iter ( int n, Noeud* racine ) {
  while (racine != NULL) {
     if ( racine->valeur == n)
        return racine;
     else if (n < racine->valeur )
        racine=racine->gauche;
     else
        racine=racine->droit;
  }
  return NULL;
}
```

Recherche récursive dans un ABR

- La fonction **récursive** consiste à comparer la valeur de la racine avec la valeur recherchée. Le résultat de la comparaison permettra de:
  - **retourner l'adresse** de la racine correspondante;
  - **appeler la même fonction** de recherche avec une modification: remplacer la racine par le fils gauche ou bien par le fils droit du nœud racine.
  - **retourner NULL** si la valeur ne se trouve pas dans l'arbre.

- Fonction de recherche récursive

```
Noeud* rechercher_valeur_rec ( int n , Noeud* racine)  {
    if ( racine->valeur == n)
        return racine;
    else if (n < racine->valeur ){
        if ( racine->gauche != NULL)
            return rechercher_valeur_rec (n, racine->gauche);
        else
            return NULL;
    } else {
        if ( racine->droit!= NULL)
            return rechercher_valeur_rec (n, racine->droit);
        else
            return NULL;
    }
}
```

Affichage des nœuds d'un ABR

5- Affichage des nœuds d'une arborescence

- Pour afficher les valeurs des nœuds d'une arborescence, nous pouvons choisir l'un des 2 parcours suivants:
  - Parcours en largeur.
  - Parcours en profondeur (le plus utilisé).

Parcours en largeur

- Le parcours en largeur consiste à explorer un arbre de gauche à droite puis de haut en bas.

Parcours en profondeur

- Le parcours en profondeur consiste à explorer un arbre de haut en bas puis de gauche à droite.
- On distingue 3 types de parcours en profondeur :
  - le parcours préfixe.
  - le parcours infixe.
  - le parcours postfixe.

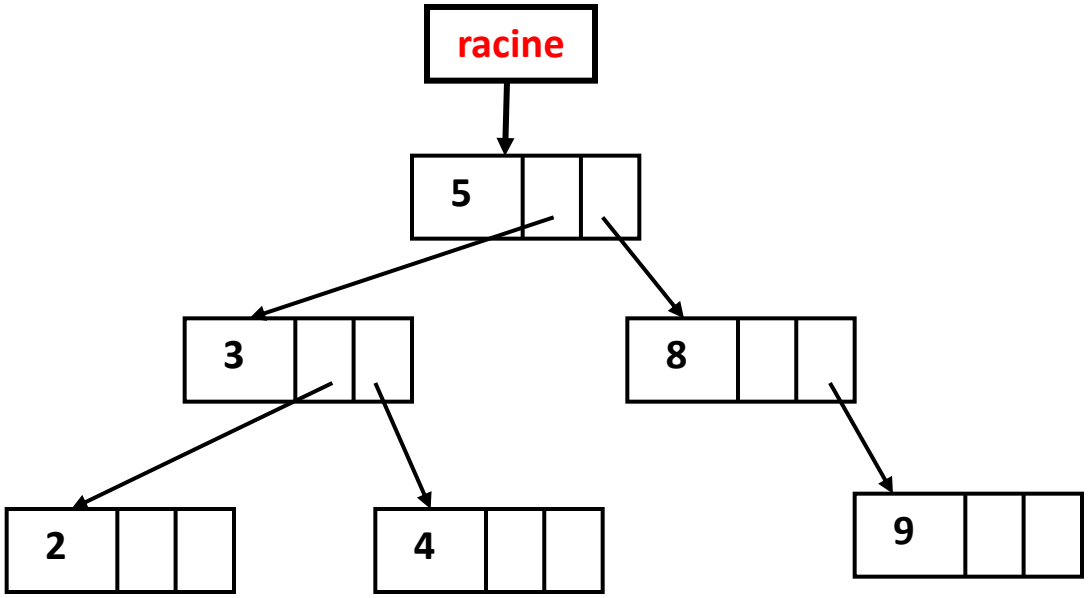
Parcours préfixe :

- Le traitement du nœud courant est effectué avant les appels récursifs des nœuds gauche et droit.

```
void affiche_prefixe (Noeud* racine){
    printf ("%d ", racine->valeur) ;
    if (racine->gauche != NULL)
        affiche_prefixe(racine->gauche ) ;
    if (racine->droit != NULL)
        affiche_prefixe(racine->droit) ;
}
```

Exemple

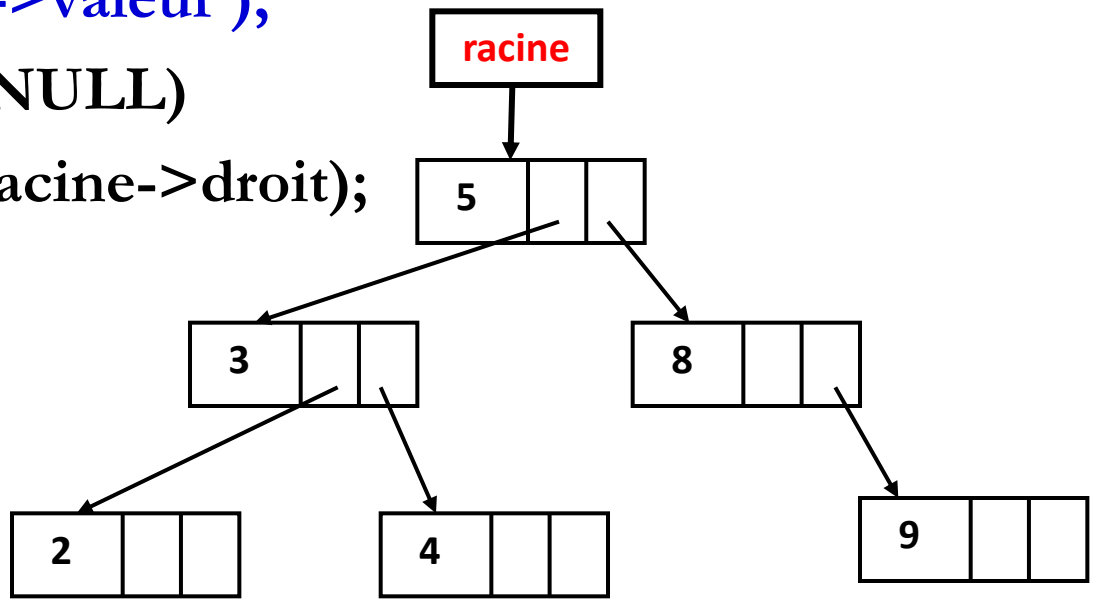
```
void affiche_prefixe (Noeud* racine){
    printf ("%d ", racine->valeur );
    if (racine->gauche != NULL)
        affiche_prefixe(racine->gauche );
    if (racine->droit != NULL)
        affiche_prefixe(racine->droit);
}
```



Parcours infixe :

- Traitement du sous-arbre gauche, puis le nœud courant et enfin le sous-arbre droit.

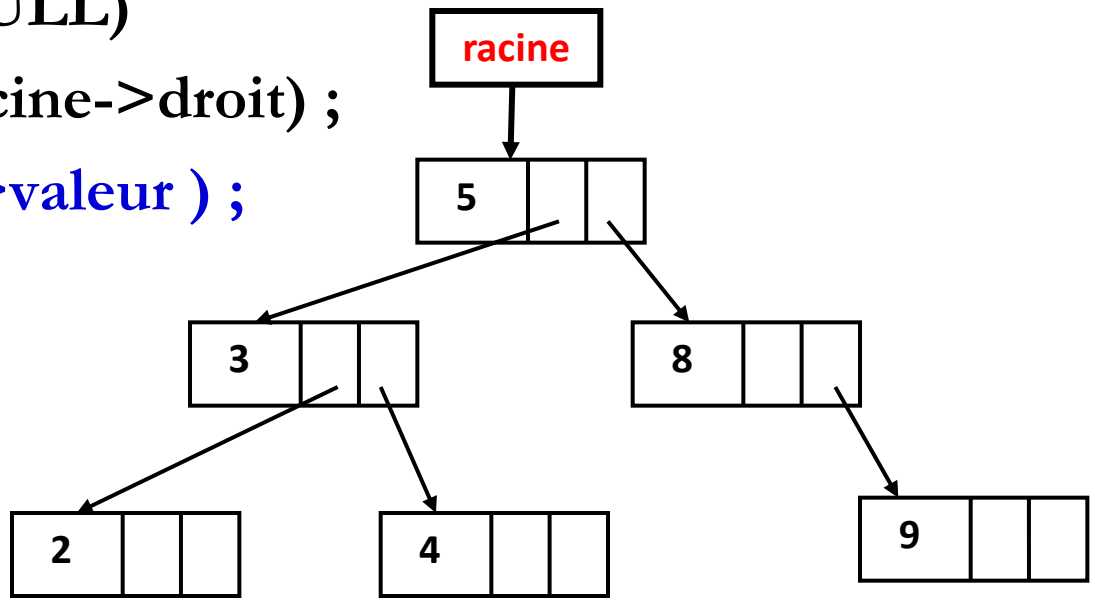
```
void affiche_infixe (Noeud* racine){
    if (racine->gauche != NULL)
        affiche_infixe(racine->gauche);
    printf("%d", racine->valeur );
    if (racine->droit != NULL)
        affiche_infixe(racine->droit);
}
```



Parcours postfixe :

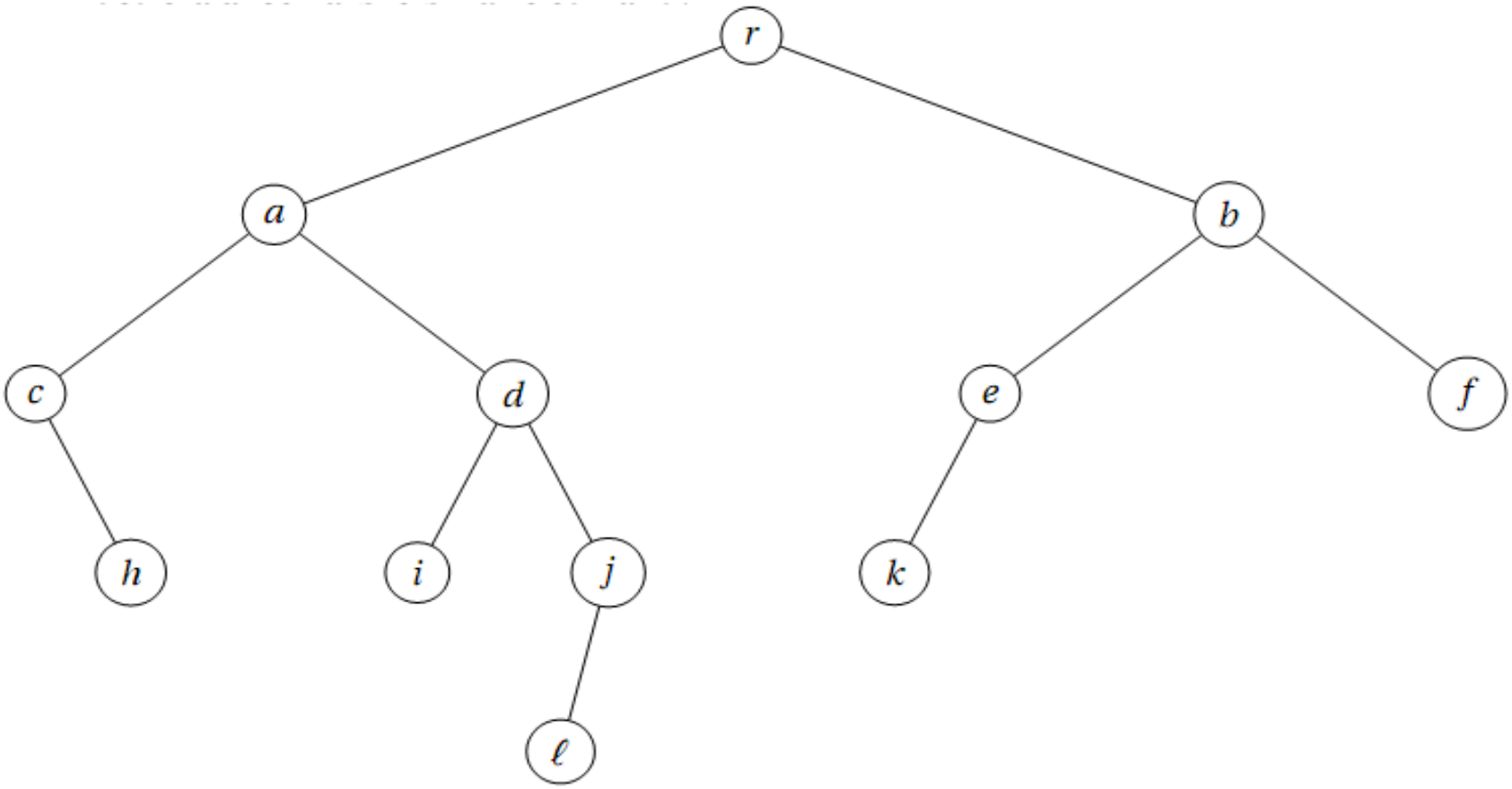
- Traitement du sous-arbre gauche, puis le sous-arbre droit et enfin le nœud courant.

```
void affiche_postfixe (Noeud* racine){
    if (racine->gauche != NULL)
        affiche_postfixe (racine->gauche ) ;
    if (racine->droit != NULL)
        affiche_postfixe (racine->droit) ;
    printf ("%d", racine->valeur ) ;
}
```



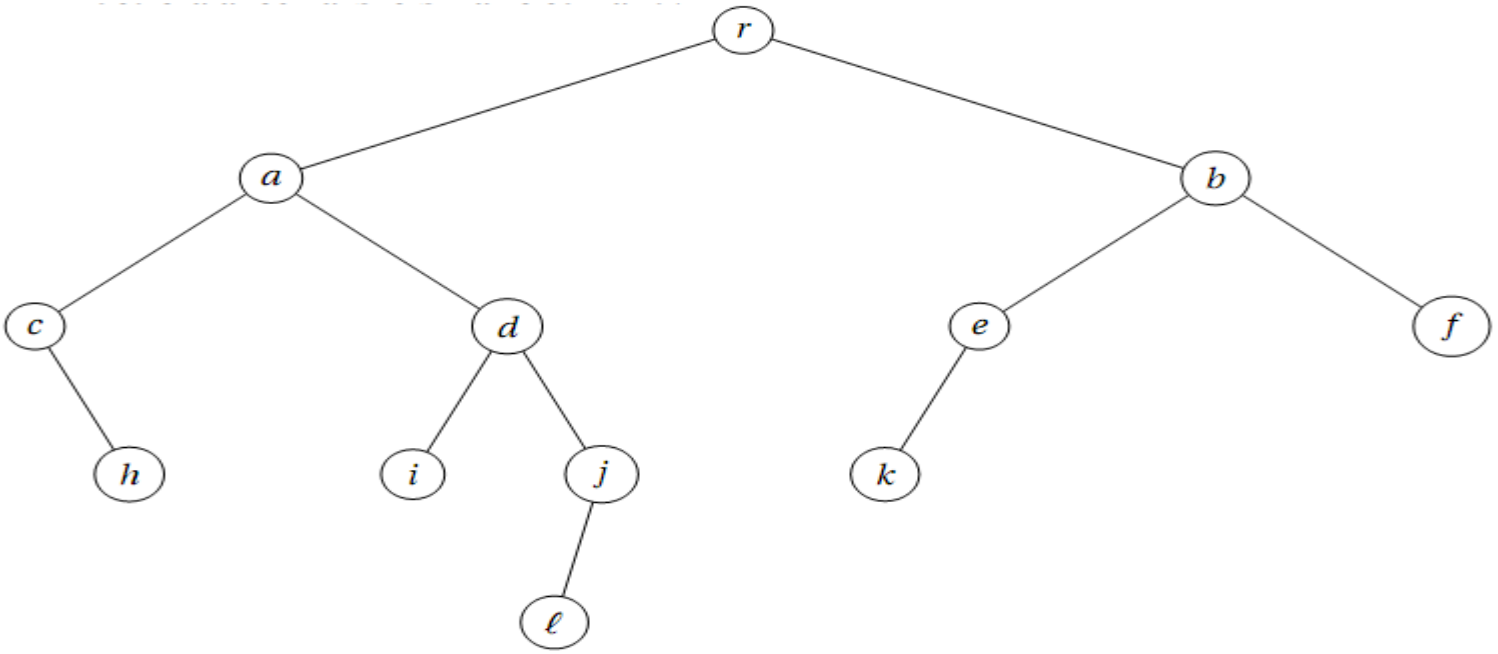
Exercice 1

- **Afficher les valeurs des noeuds de l'arbre suivant en utilisant les 3 types de parcours: préfixe, postfixe et infixe.**



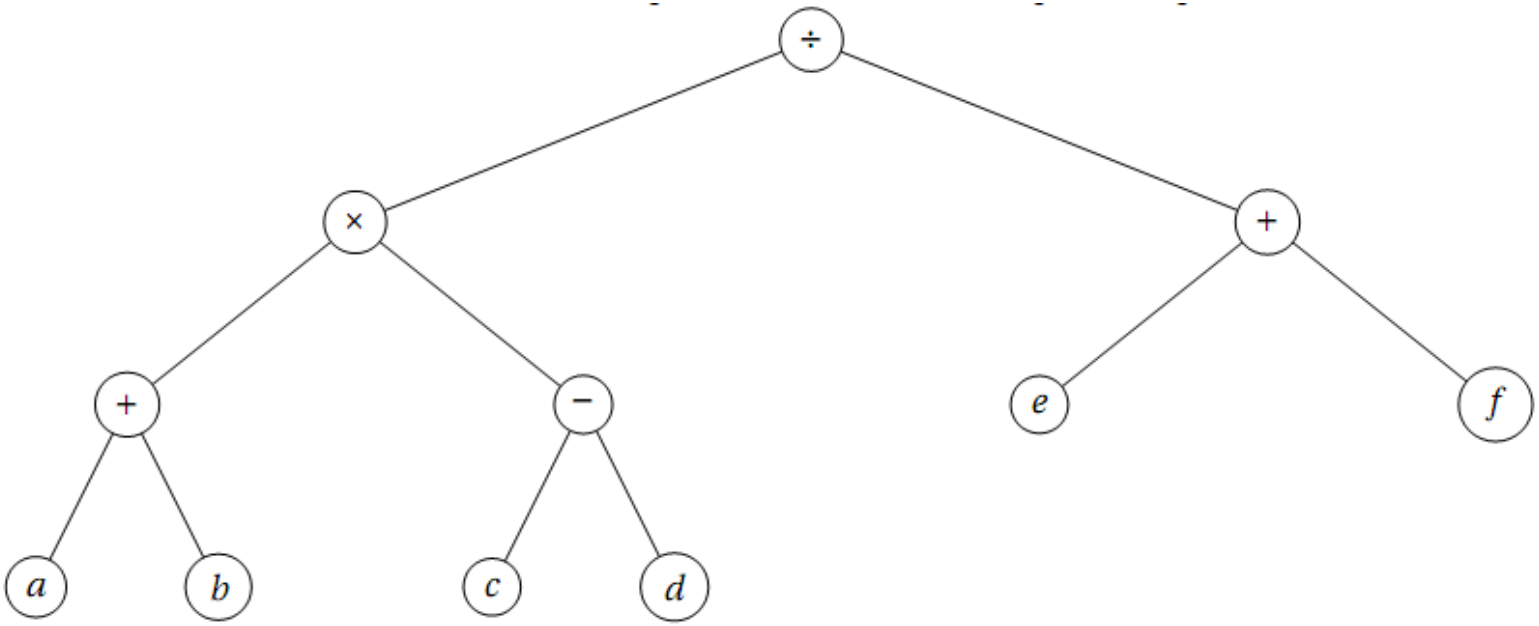
Solution

- Parcours préfixe : r, a, c, h, d, i , j, l, b, e, k, f.
- Parcours postfixe : h, c, i, l, j, d, a, k, e, f, b, r.
- Parcours infixe : c, h, a, i ,d, l, j, r, k, e, b, f.



Exercice 2

- **Écrire les sommets de l'arbre ci-dessous pour chacun des types de parcours: postfixe, préfixe et infixe :**



- **Pour le parcours infixe, on ajoute une parenthèse ouvrante à chaque fois qu'on rentre dans un sous-arbre et on ajoute une parenthèse fermante lorsqu'on quitte ce sous-arbre.**

Solution

- Parcours préfixe : ÷, *, +, a, b, -, c, d, +, e, f
- Parcours postfixe : a, b, +, c, d, -, *, e, f, +, ÷
- Parcours infixe : ((a+b)*(c-d)) ÷(e+f)

Suppression d'un nœud d'un ABR

```
Noeud * supprimer_noeud (Noeud* racine,  int val ) {
   Noeud *tmp;
   if(racine->valeur == val ) {
      if( racine->gauche != NULL)  {
         // on accroche arbre->droit au fils le plus à droite du fils gauche
         tmp = racine->gauche;
         while(tmp->droit != NULL)
            tmp = tmp->droit;
         tmp->droit = racine->droit;
         racine = racine->gauche;
      }
      else
         racine = racine->droit;
   }
```

```
    else {
        if( val < racine->valeur )
            racine->gauche = supprimer_noeud ( racine->gauche, val);
        else
            racine->droit = supprimer_noeud ( racine->droit, val);
    }
    return racine;
}
```

Avis

- **DS 2: La semaine du 02/05.**
- **Présentation du mini projet: la semaine du 09/05.**
- **Rattrapage: la semaine du 16/05.**