R3.02 : Développement efficace

Structure en arbre 1

J-F. Kamp

Septembre 2024

Notion de table

Ensemble formé d'un nombre variable, éventuellement nul, de données sur lequel on peut effectuer les opérations suivantes :

- Ajout d'une nouvelle donnée (tuple = clé unique + info)
- Recherche d'une donnée par sa clé
- Consultation de l'info associée à une clé
- Suppression de la donnée d'après sa clé
- Test si la table est vide

Mise en œuvre par un tableau

Mise en œuvre par un tableau

- Tableau de n éléments de type association

- Éléments du tableau triés par ordre croissant sur les clés

Chaque tuple *Ti* (i de 1 à n) est un objet qui contient :

- une information (+ sieurs champs)
- une clé supposée entière

Opération de recherche

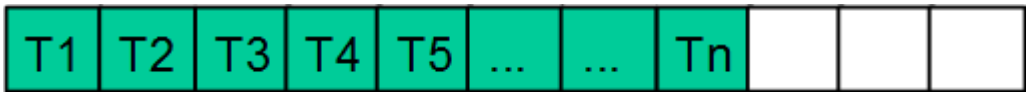
Opérations de consultation

EstDans ?
chercher la clé en utilisant un algorithme de dichotomie

Valeur
chercher la clé en utilisant un algorithme de dichotomie et retourner l'info associée à la clé

Opération de modification

Opérations de modification



Oter

chercher l'élément (dichotomie) et le supprimer en décalant tous les éléments de clé plus grande

Ajouter

parcourir le tableau jusqu'à trouver l'endroit d'insertion (par dichotomie) puis décaler les éléments suivants d'une position et placer l'élément à ajouter

Conclusion

Opérations de consultation (*estDans ?* et *Valeur*) sont efficaces car la recherche est rapide pour un grand tableau : complexité en $O(log_2(n))$ pour *n* éléments.

Opérations de modification sont très coûteuses : complexité en $O(n)$ pour *n* éléments.

Conclusion : solution à envisager lorsque la table conserve une taille relativement constante : peu d'insertions ou de suppressions pour beaucoup de recherches.

Mise en œuvre par une liste chaînée

Pas d'algorithme de recherche intéressant car on ne peut que parcourir la liste (complexité en *O(n)*).

Même les opérations de modification (*Oter* et *Ajouter*) sont coûteuses car nécessitent une recherche préalable dans la liste.

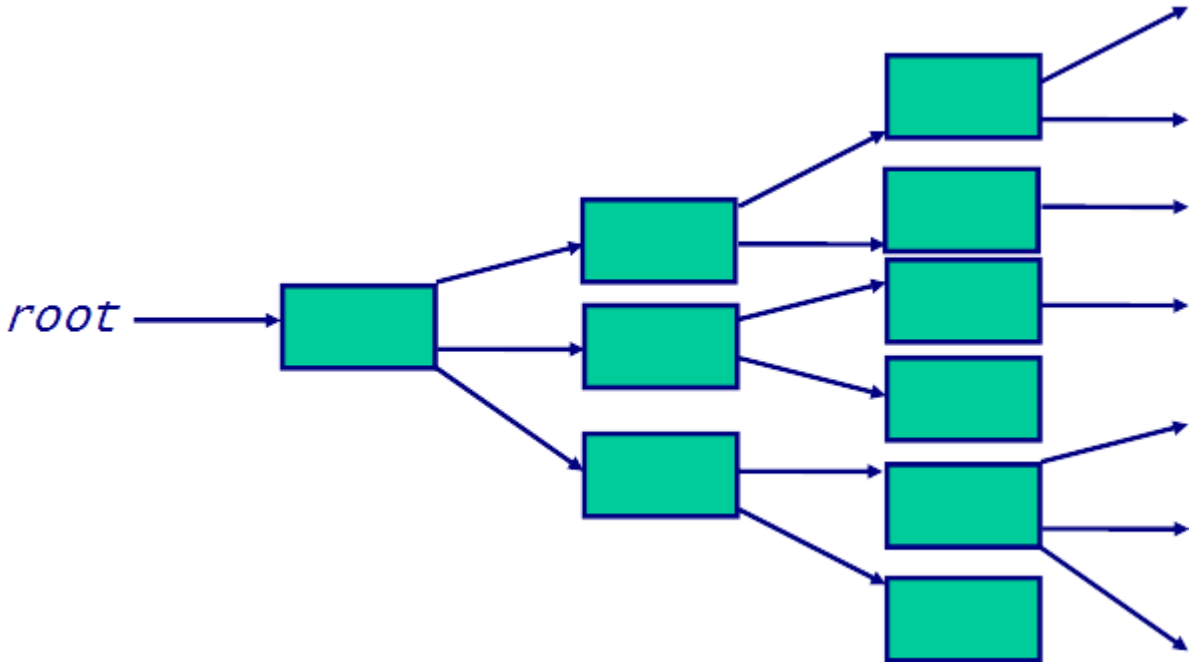
Solution à proscrire.

Mise en œuvre par un arbre

Mise en œuvre par un arbre

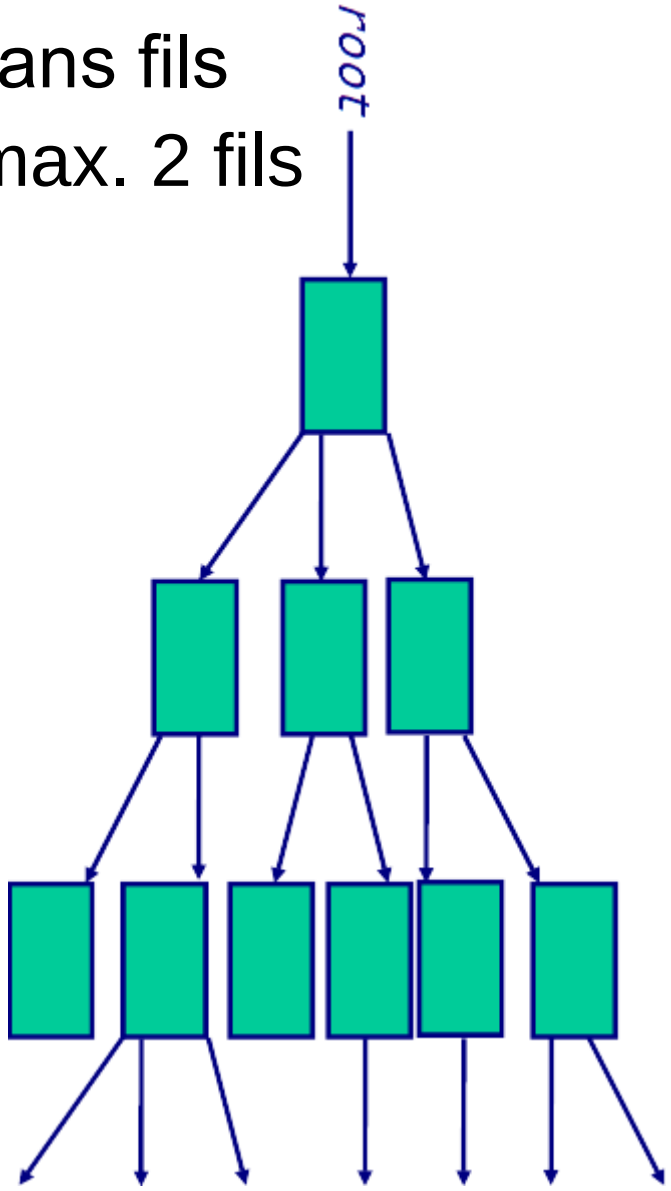
Un arbre est une liste chaînée ou chaque élément peut avoir +sieurs voisins à droite (fils) mais un seul voisin à gauche (père) :

- élément = nœud (*node*)
- le début = nœud (*root*) le seul nœud sans voisin à gauche



Arbre vertical : terminologie

- le début = nœud racine (*root*)
- voisin bas = fils (*son*)
- voisin haut = père (*father*)
- lien = branche père/fils
- feuille = nœud sans fils
- arbre binaire = max. 2 fils

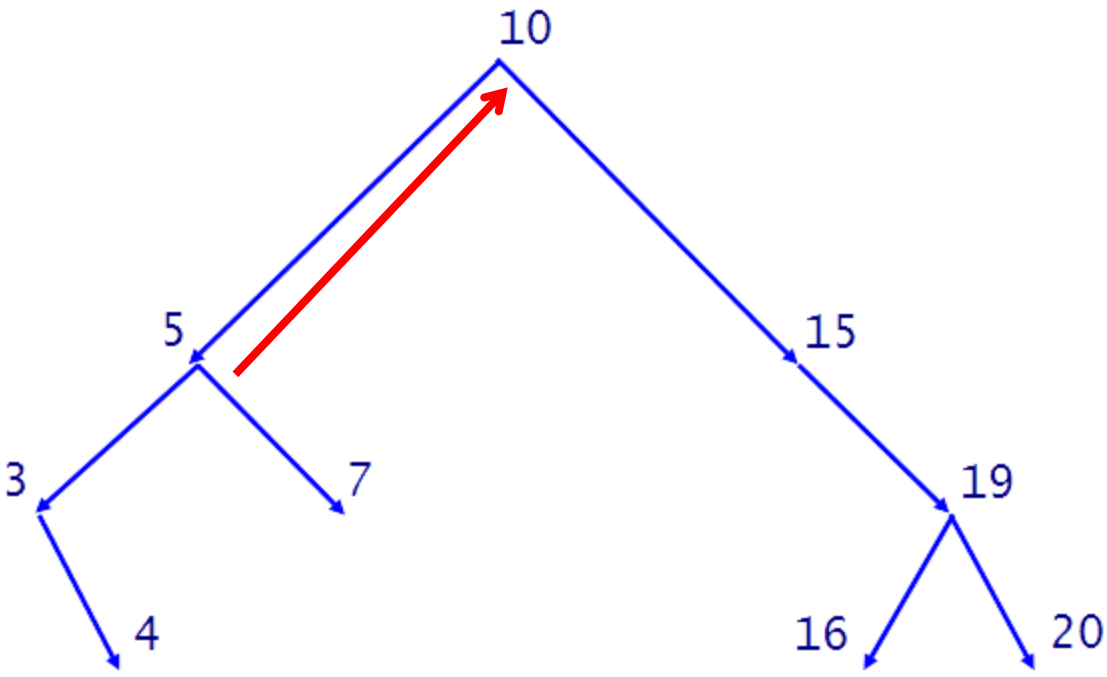


Arbre binaire de recherche

Un arbre binaire de recherche (ABR) est un arbre binaire (2 fils max.), éventuellement vide, qui possède les propriétés suivantes :

- chaque nœud est un tuple (info + clé unique)
- en chaque nœud : toutes les clés à sa gauche (sous arbre gauche) sont strictement inférieures à la clé du nœud
- en chaque nœud : toutes les clés à sa droite (sous arbre droit) sont strictement supérieures à la clé du nœud

Exemple d'ABR



- root (10) est le seul nœud sans père
- chaque nœud est aussi relié à son père (liaison **double** père/fils)
- un nœud peut avoir :
  - 2 fils
  - 1 seul fils
  - 0 fils (nœud feuille)

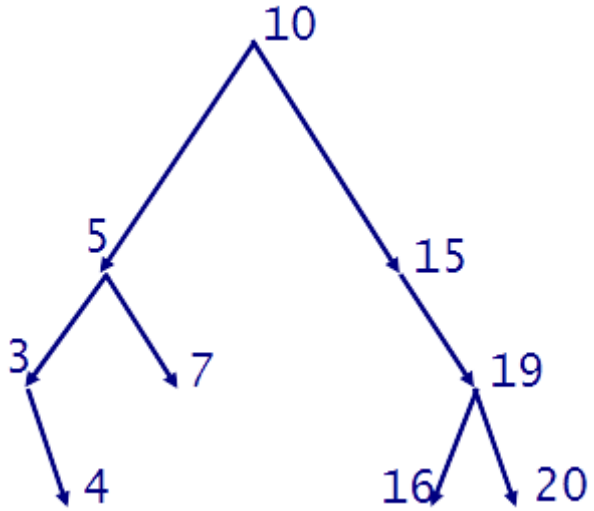
Recherche

ABR : recherche d'un tuple (nœud)

Exemple : recherche de la clé de valeur 16 à partir du nœud racine (noeud10).

Algorithme RECURSIF :

recherche (16, noeud10)
  retourner recherche (16, noeud15)
    retourner recherche (16, noeud19)
      retourner recherche (16, noeud16)
        retourner vrai

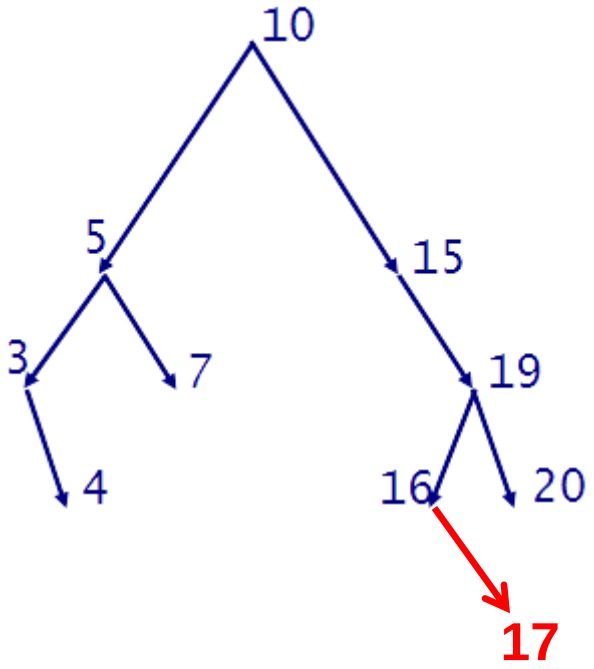


Insertion

ABR : ajout d'un tuple (nœud)

1. Créer le nœud (tuple = info + clé)
2. Rechercher la place de la clé dans l'ABR => le père auquel se raccrocher
3. Raccrocher le nouveau nœud à son père : à sa droite ou à sa gauche suivant la valeur de sa clé

Exemple : ajout noeud17 dans l'arbre ci-dessous



ABR : ajout d'un tuple (nœud)

```
//   Rôle
//   Recherche où insérer un noeud dans un arbre
//   Retourne le noeud qui servira de pere
//   Le code est-il complet ?

fonction rechercherPlace  ( cle ) : noeud

debut
    x := root;

    tant que (x!= vide) faire

        pere := x;

        si x.cle < cle
            alors x := x.left;
            sinon x := x.right;
        finsi

    fintantque

    retourner pere;
fin
```

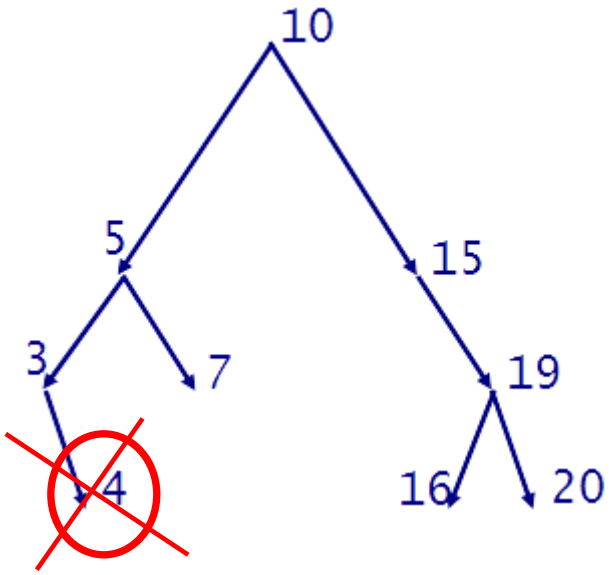
Suppression

Suppression d'un tuple : cas1

Principe :

1. rechercher le nœud à supprimer
2. supprimer le nœud (s'il existe)
   => 3 cas possibles

Cas 1 : le nœud est une feuille

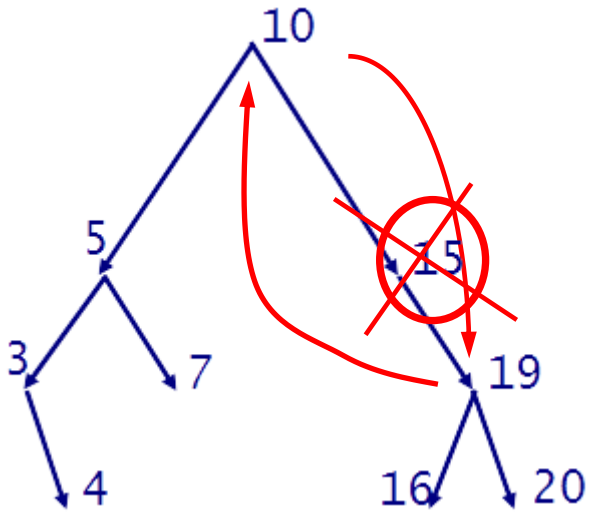


Suppression d'un tuple : cas2

Principe :

1. rechercher le nœud à supprimer
2. supprimer le nœud (s'il existe)
   => 3 cas possibles

Cas 2 : le nœud n'a qu'un seul fils



Suppression d'un tuple : cas3

Cas 3 : le nœud a 2 fils

Remplacer ce nœud à supprimer par le plus grand du SAG (ou le plus petit du SAD).

Ensuite supprimer le nœud de remplacement.