Université Mohammed V- Agdal
Ecole Mohammadia d'Ingénieurs
Département Génie Informatique

Structures de Données

Par: Mr N.EL FADDOULI

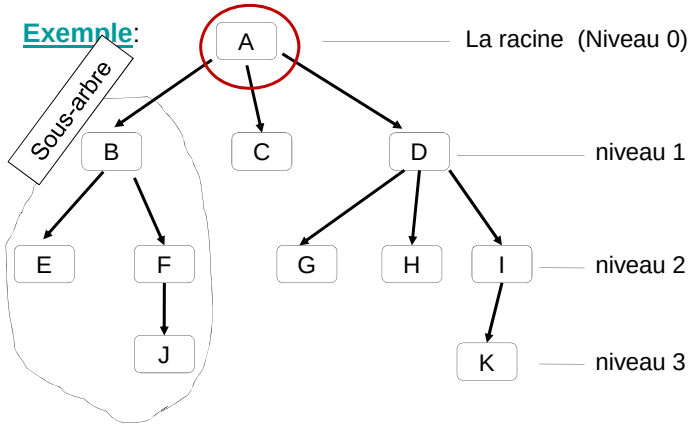
Année Universitaire:2013/2014

Les arbres: Définition

« Un arbre est composé d'un **ensemble de nœuds**, d'un **ensemble d'arcs** reliant les nœuds, et d'un nœud particulier appelé **racine** de l'arbre.

Un arc (n1,n2) établit une relation entre le nœud n1, appelé **nœud parent**, et n2, appelé **nœud enfant** de n1. L'ensemble des arcs doit être tel que chaque nœud, sauf la racine, a exactement un parent »

Exemple:



Les arbres: Caractéristiques

- Trois types de nœuds:

- **Racine:** le seul nœud de l'arbre qui n'a pas de parent.
- **Feuilles**: les nœuds qui n'ont pas d'enfants.
- **Nœuds internes:** ni des feuilles ni la racine.

- Chaque nœud possède:

- un **degré: le nombre de ses enfants.**
- un **niveau (profondeur): nombre d'arcs qu'il faut remonter pour** atteindre la racine.

• A chaque nœud est aussi associé un élément qu'on appelle ***contenu du nœud ou valeur du nœud***

• Le **degré (l'arité) de l'arbre est le plus grand degré qu'on trouve**.

Les arbres: Caractéristiques

• On peut fixer un degré maximum pour un arbre pour qu'il soit: **unaire, binaire, ternaire...**

• La **hauteur de l'arbre est le plus grand niveau qu'on** trouve parmi ses nœuds (la profondeur maximale).

• Si l'ordre entre les sous-arbres enfant est pris en compte, on parlera d'**arbre ordonné.**

• **Les opérations sur les arbres**:

- Ajouter un élément.
- Déterminer si un élément existe dans l'arbre.
- Supprimer un élément de l'arbre.
- …

• Dans ce qui suit, on traitera les arbres binaires (degré=2). C'est le type le plus basique dont toutes les opérations sont valables pour un arbre n-aire.

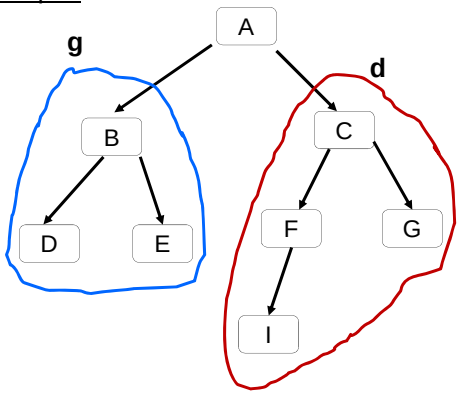
Les arbres binaires

Définition:

Un arbre binaire T est:

- Soit un arbre **vide**
- Soit un triplet (**r,g,d**) où:
  - **r** est la valeur de la racine de l'arbre
  - **g** est un sous-arbre gauche
  - **d** est un sous-arbre droite.
  - les deux sous-arbres peuvent être vides.

Exemple:



Les arbres binaires

Définition TAD:

Type arbre:

Utilise élément, booléen , entier

Opérations:

estvide	: arbre	→ booléen
arbrevide	:	→ arbre
racine	: arbre	→ élément
arbregauche	: arbre	→ arbre
arbredroit	: arbre	→ arbre
hauteur	: arbre	→ entier

existeelement: arbre, élément → booléen

….

Fin Type

Exercice:

- Définir la fonction **existeelement** de manière récursive en fonction de: **estvide**, **racine**, **arbregauche** et **arbredroite**.

Les arbres binaires

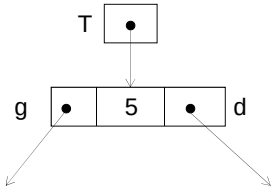
Booléen existeelement (arbre A, element e)
Début

Fin
Entier Hauteur(arbre A)
Début

Fin
Entier NbNoeuds(arbre A) .....
Entier NbFeuilles(arbre A) .....
Entier NbFeuillesIn(arbre A) .....

Les arbes binaires

Implémentation chainée:




```
typedef   struct nœud  *   arbre;
struct nœud { int  val;
                  struct nœud *g;
                  struct  nœud  *d;   };
```

Créer un arbre vide: arbre T = NULL;

Les arbes binaires

Opérations:

int **EstVide**(arbre);

arbre **FilsGauche** (arbre);

arbre **FilsDroit** (arbre);

int **EstFeuille**(arbre);

int **EstNoeudInterne**(arbre);

int **Hauteur**(arbre);

int **NbreNoeud**(arbre);

int **NbreFeuille**(arbre);

int **nbreNoeudInterne**(arbre);

Les arbes binaires

Opérations:

int **EstVide**(arbre T) {

}

arbre **FilsGauche** (arbre T) {

}

arbre **FilsDroit** (arbre T) {

}

int **EstFeuille**(arbre T) {

}

int **EstNoeudInterne**(arbre T) {

}

Les arbes binaires

Opérations:

int **Hauteur**(arbre T) {

}

int **NbreNoeud**(arbre) {

}

int **NbreFeuille**(arbre) {

}

int **nbreNoeudInterne**(arbre) {

}

Les arbes binaires

Parcours en profondeur:

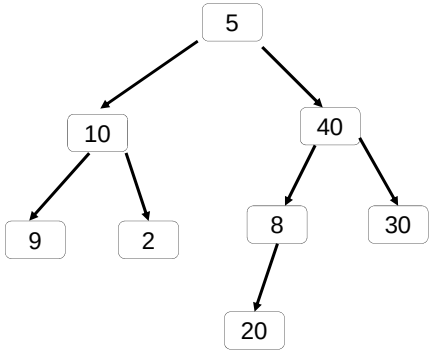
Préfixe, infixe, suffixe

Parcours préfixe:

Pour un arbre T:

- On traite la **racine** de l'arbre T (affichage, …)
- On passe ensuite aux sous-arbres **gauche** et **droit** → C'est un parcours **RGD** ou **RDG**.

Exemple: Affichage préfixe de l'arbre suivant:



5, 10, ……..

Les arbes binaires

Parcours préfixe:

```
void  Parcours_Pref( arbre T)
{




}
```


Les arbes binaires

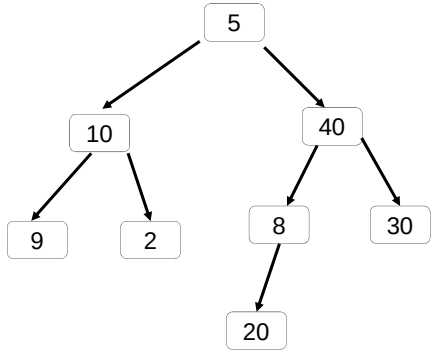
Parcours infixe:

Pour un arbre T:

- On traite le sous-arbre **gauche**
- On traite la **racine** de l'arbre T (affichage, …)
- On passe ensuite au sous-arbre **droit**.

→ C'est un parcours **GRD**

Exemple: Affichage infixe de l'arbre suivant:



……

Les arbes binaires

Parcours infixe:

```
void  Parcours_Inf( arbre T)
{




}
```


Les arbes binaires

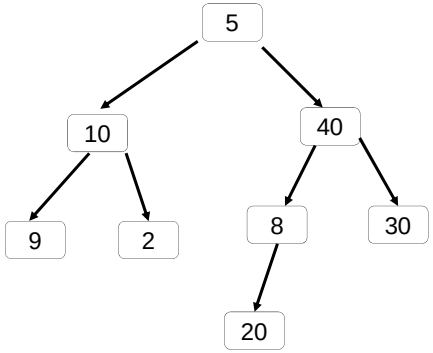
Parcours post-fixe:

Pour un arbre T:

- On traite le sous-arbre **gauche**
- On passe ensuite au sous-arbre **droit**.
- On traite la **racine** de l'arbre T (affichage, …)

→ C'est un parcours **GDR**

Exemple: Affichage postfixe de l'arbre suivant:



……

Les arbes binaires

Parcours post-fixe:

void Parcours_Post(arbre T)

{

}

Exercice:

- Fonction qui calcule le nombre d'occurrences d'un entier dans un arbre T.

Les arbes binaires

Création d'un arbre:

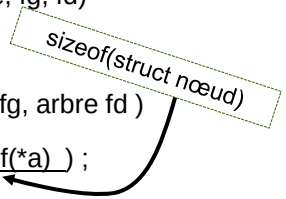
• Un arbre vide est un pointeur contenant NULL.

• Un arbre non vide est un pointeur sur la racine de l'arbre (*structure contenant trois champs: **valeur** de la racine, **sous-arbre gauche** et **sous-arbre droit***)

• La création d'un arbre se fait à partir d'une **racine**, un **sous-arbre gauche** et un **sous-arbre droit**:

arbre **CreerArbre**(val_racine, fg, fd)

En langage C: arbre d'entiers




```
arbre   CreerArbre (int v,  arbre fg, arbre fd )
{ arbre   a=(arbre) malloc( sizeof(*a)  ) ;
a -> val = v;
a ->g =fg;       a ->d = fd;
return a;
}
```

Les arbes binaires

Ajouter un élément dans un arbre:

• Ajouter dès que possible (rencontrer un fils vide)

• Ajouter l'élément de façon à garder un ordre entre les éléments de l'arbre.

• Ajouter de façon à avoir un arbre complet le plus possible (un arbre complet = arbre binaire dont les feuilles ont la même profondeur).

Ajout à la rencontre d'un fils vide:

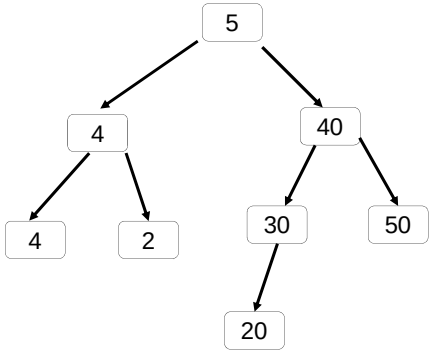
```
void  Ajouter( arbre  *t,  int  v)
{
if (*t==NULL)  *t = CreerArbre( v, NULL, NULL);
  else   if (  (*t)->g==NULL)
                    (*t)->g= CreerArbre( v, NULL, NULL);
         else if ( (*t)->d==NULL)
                     *t) ->d= CreerArbre( v, NULL, NULL);
               else Ajouter (&((*t)->g), v) ;
}
```


Les arbes binaires de recherche

- Les éléments du sous-arbre gauche sont inférieurs ou égales à la racine.
- Les éléments du sous-arbre droit sont supérieurs à la racine.
- Optimiser la recherches dans un arbre: parcourir seulement une branche

  → temps de recherche proportionnel à la hauteur de l'arbre.

Exemple:



Les arbes binaires de recherche

Recherche d'un élément:

```
Booléen Recherche( arbre T, entier  x)
{
Si Estvide(T) retourner faux
Sinon
   Si   Racine(T) = X retourner vrai
   Sinon
      Si  Racine(T)>=x
         retourner  Recherche(Filsgauche(T), x)
      Sinon
         retourner   Recherche(Filsdroit(T), x)
}

int Recherche( arbre T, int  x)
{




}
```

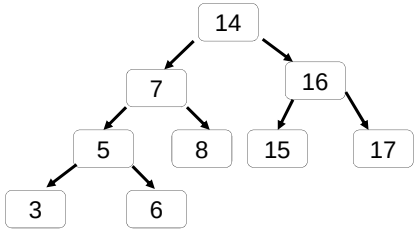

Les arbes binaires de recherche

Ajouter un élément comme feuille l'arbre:

```
void  AjouterElt( arbre  *T, int x)
{
if ( *T ==NULL)   *T = CreerArbre(x, NULL, NULL);
else   if (x > (*T) -> val)  AjouterElt(&((*T) ->d), x);
        else AjouterElt(&((*T) ->g), x);
}
```

Exercice: Ecrire un programme principal qui fait appel à cette fonction. La saisie des éléments de l'arbre se fera **par niveaux** (niveau 0, niveau 1, …)

Exemple: la saisie des éléments 14, 7, 16, 5, 8, 15, 17, 3, 6 permettra d'avoir l'arbre suivant:



Les arbes binaires de recherche

Insérer un élément:

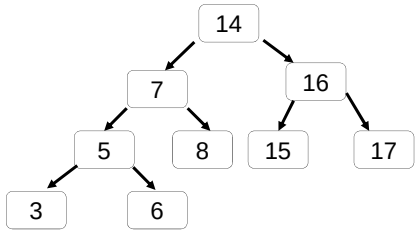
```
void  InsererElt( arbre  *T, int x)
{




}
```

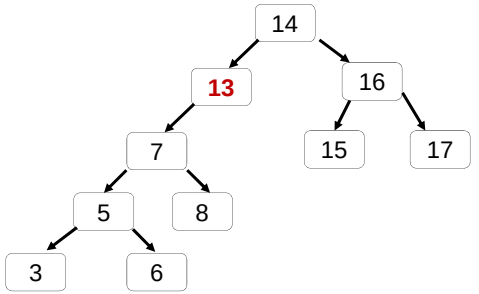
Exercice: Modifier le programme principal de l'exercice précédent pour faire appel à la fonction InsererElt afin d'insérer un nouvel élément.

Exemple: la saisie des éléments 14, 7, 16, 5, 8, 15, 17, 3, 6 permettra d'avoir l'arbre suivant:

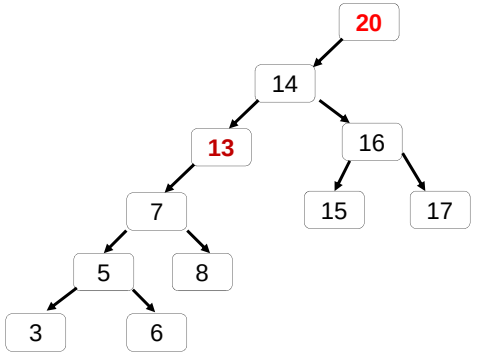


Les arbes binaires de recherche

L'insertion de 13 modifiera la liste comme suit:



L'insertion de 20 modifiera la liste comme suit:



TP

Exercice 1:

1.Définir la structure Nœud et le type Arbre (*voir cours*)

2.Définir la fonction **Arbre CreerArbre (int v, Arbre G, Arbre D)** qui crée un arbre de racine **v** et ayant les sous-arbres **G** et **D**.

3.Ecrire la fonction **void InsererElt(arbre *T, int X)** qui ajoute un nœud **X** dans un **arbre de recherche T**.

4. Ecrire la fonction **void Parcours_pref(Arbre T)** pour afficher l'arbre **T** en ordre préfixé

5. Ecrire un programme principal lire les valeurs d'un arbre par niveau et les **insère** dans un arbre **initialement vide** et **affiche** l'arbre créé.

Exemple: la saisie des éléments 14, 7, 16, 5, 8, 15, 17, 3, 6 permettra d'avoir: