2 Arbres

Algorithmique et Programmation C++ 2

Principe

Une liste chaînée peut enregistrer dans ses maillons une information parfois très complexe, elle-même parfois composée de listes chaînées. Ainsi, on peut définir un arbre comme une liste de listes.

On peut de manière générale raccrocher un arbre à une **organisation hiérarchique** (organigramme, livre, etc.).

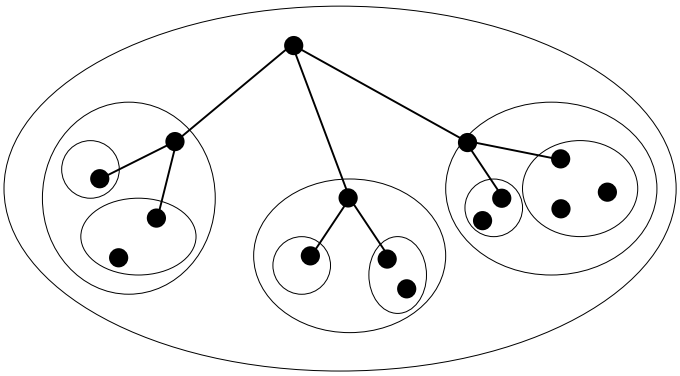
Remarque : On parle parfois en algorithmique de l'arbre des appels récursifs.

Chaque branche d'un arbre est elle-même un arbre, similaire au premier.

Définitions

Définition récursive

A est un **arbre** si et seulement si A est un ensemble qui contient :
- un élément particulier : la **racine** de A,
- dont tous les autres éléments sont **partitionnés**,
- dont un sous-ensemble de cet ensemble est lui-même un arbre, appelé **sous-arbre** de A.



Autre définition

A est un **arbre** si et seulement si :
- A est vide,
- ou $A=(I,(A_0,A_1,\ldots,A_n))$, avec I une information et $\{A_0,A_1,\ldots A_n\}$ des arbres.

Eléments de l'arbre

Un **nœud** est l'élément d'un arbre qui porte une information. On définit les relations hiérarchiques entre les nœuds : **Fils**, **Frère**, **Père**.

Une **feuille** est un nœud qui n'a pas de fils.

Le **degré** d'un nœud est le nombre de fils que possède ce nœud.

La **profondeur** d'un nœud est la longueur depuis la racine jusqu'au nœud.

La **hauteur** de l'arbre est le maximum de toutes les profondeurs.

Arbres binaires

Définition

Un **arbre binaire** est un arbre dont les nœuds ont au plus deux fils.

Structure de données

```
class Noeud
{
      int info;
      Noeud * fg, * fd;
public:
      .
      .
      .
};

Noeud::Noeud(int i=0, Noeud * g=NULL; Noeud * d=NULL)
{
      info=i;
      fg=g;
      fd=d;
}

class Arbre
{
      Noeud * racine;
public:
      Arbre() {racine=NULL;}
      .
      .
      .
};
```

Affichage

```
void Arbre::afficher()
{
      if(racine==NULL)
            cout << "Arbre vide" << endl;
      else
            racine->afficher();
}

void Noeud::afficher()
{
      cout << info << ';';          // (1)
      if(fg!=NULL) fg->afficher(); // (2)
      if(fd!=NULL) fd->afficher(); // (3)
}
```

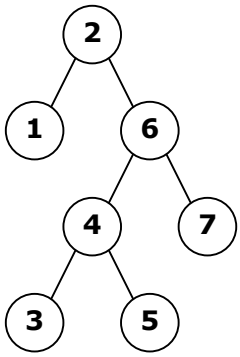
Selon l'ordre dans lequel sont placées les trois lignes, l'affichage sera différent. On distingue :
- 1,2,3 : Parcours préordre,
- 2,1,3 : Parcours inordre,
- 2,3,1 : Parcours postordre.

Application :

Préordre : On s'occupe du père avant les fils (PRE).
2 1 6 4 3 5 7

Postordre : On s'occupe des fils avant le père (POST).
1 3 5 4 7 6 2

Inordre : Fils gauche, père, puis fils droit (IN).
1 2 3 4 5 6 7



Arbre binaire ordonné

Chaque nœud ne porte que des valeurs plus petites que lui-même dans son sous-arbre gauche, et que des valeurs plus grandes dans son sous-arbre droit.

Cette structure permet de rechercher rapidement des données ordonnées (dichotomie).

Insertion d'une valeur dans un arbre ordonné

Méthode itérative :

```
void Arbre::ajouter(int i)
{
      Noeud * n, *nn;
      n=racine;
      nm=NULL;
      while(n!=NULL)
      {
            nn=n;
            if(i < n->info) n=n->fg;
            else if(i > n->info) n=n->fd;
                  else {cout << "Double !" << endl; return;}
      }
      if(nn=NULL) // cas ou l'arbre est vide :
            racine=new Noeud(i,NULL,NULL);
      else
            if(i < nn->info)
                  nn->fg=new Noeud(i,NULL,NULL);
            else
                  nn->fd=new Noeud(i,NULL,NULL);
}
```

Méthode itérative améliorée :

```
void Arbre::ajouter(int i)
{
      Noeud **n;
      n=&racine;
      while(*n!=NULL)
      {
            if(i < (*n)->info)
                  n=&((*n)->fg);
            if(i > (*n)->info)
                  n=&((*n)->fd);
            if(i==(*n)->info)
                  {cout << "Double !" << endl; return;}
      }
      *n=new Noeud(i,NULL,NULL);
}
```

Méthode par référence :

```
void Arbre::ajouter(int i)
{
      ajouterf(i,racine);
}

void Arbre::ajouterf(int i, Noeud * &n)
{
      if(n==NULL) n=new Noeud(i,NULL,NULL);
      else if(i < n->info) ajouterf(i,n->fg);
            else if(i > n->info) ajouterf(i,n->fd);
                  else cout << "Double !" << endl;
}
```

Méthode par adresse :

```
void Arbre::ajouter(int i)
{
      ajoutera(i,&racine);
}

void Arbre::ajoutera(int i, Noeud **n)
{
      if(*n==NULL)
            *n=new Noeud(i,NULL,NULL);
      if(i < (*n)->info)
            ajoutera(i,&((*n)->fg));
      else if(i > (*n)->info) ajoutera(i,&((*n)->fd));
            else cout << "Double !" << endl;
}
```