3.8- Implémentation par une liste_SC avec un pointeur sur le dernier élément

modification

```
fonction detruireFile(ref F:file de objet):vide;
  debut
    tantque !fileVide(F) faire
      defiler(F)
    fintantque
  fin
finfonction
```

4-Arbre

- Arbre et arborescence
- Arbres binaires
- Parcours d'un arbre binaire
- Implémentation d'un arbre binaire
- Retour sur les arborescences
- Parcours d'un arbre planaire
- Implémentation d'un arbre planaire

4.1-Arbre et arborescence

Définition 4.1: Un arbre est un graphe connexe sans cycle.
Un sous arbre est un sous graphe d'un arbre.

Propriété 4.1: Si un arbre a n sommets alors il a n-1 arêtes.

Idée de la démonstration: ceci s'appuie sur les deux propriétés suivantes des graphes connexes.

- Tout graphe connexe ayant n sommets a au moins n-1 arêtes.
- Tout graphe connexe ayant n sommet et au moins un cycle a au minimum n arêtes.

La **taille** d'un arbre est le **nombre de sommets** de l'arbre.

4.1-Arbre et arborescence

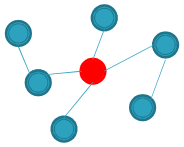
Propriété 4.2: Entre deux sommets quelconques d'un arbre, il existe une unique chaîne les reliant.

Idée de la démonstration : Pour deux sommets quelconques

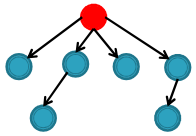
- Il ne peut exister deux chaines différentes les reliant sinon il y aurait un cycle dans l'arbre.
- Il existe au moins une chaine puisque un arbre est un graphe connexe.

4.1-Arbre et arborescence

Définition 4.2: Une **arborescence** est définie à partir d'un arbre en choisissant un sommet appelé **racine** et **en orientant** les arêtes de sorte qu'il existe un chemin de la racine vers tous les autres sommets.

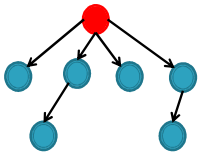


Arbre



Arborescence

4.1-Arbre et arborescence



Définitions 4.3 :

- On appelle **fils** d'un sommet s tout sommet s' tel que **(s,s') est une arête** de l'arbre.
- On notera qu'une **arborescence** est un exemple **d'ensemble partiellement ordonné** (relation d'ordre "fils de").
- On appelle **feuille de l'arbre** un sommet qui n'a pas de successeur . Tout autre sommet est appelé **sommet interne.**
- On appelle **hauteur d'un sommet** de l'arbre la longueur du chemin de la racine à ce sommet.

4.1-Arbre et arborescence

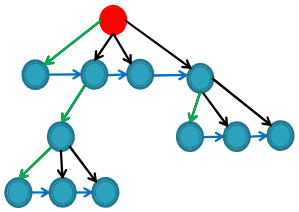
Définition 4.4: Un **arbre planaire** est défini en ordonnant les arêtes sortantes de chaque sommet. On notera qu'un arbre planaire est un exemple d'ensemble totalement ordonné (relation "est fils ou est frère à droite").

Définition 4.5: Un **arbre binaire** est un arbre planaire dont chaque sommet a au plus deux fils.

Définition 4.6: Un arbre binaire **complet** est un arbre binaire dont chaque sommet interne a exactement deux fils. .

4.1-Arbre et arborescence

Arbre planaire



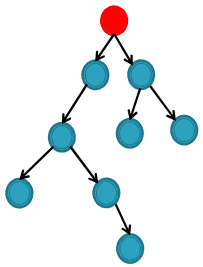
Racine

Premier Fils

Frère droit

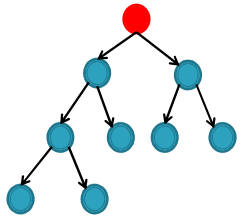
4.1-Arbre et arborescence

Arbre Binaire



4.1-Arbre et arborescence

Arbre Binaire complet

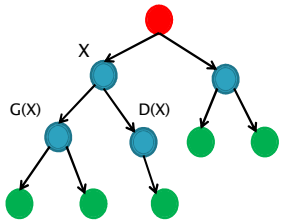


4.1-Arbre et arborescence

Propriété 4.3: Tout sommet x d'un arbre binaire vérifie l'une des deux propriétés suivantes:

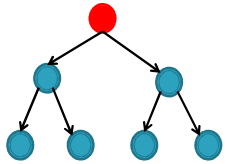
- x est une feuille,
- x a un sous arbre binaire dit *gauche* de racine G(x) et un sous arbre binaire *droit* de racine D(x).

feuille



4.1-Arbre et arborescence

Définition 4.7: Un arbre binaire **parfait** est un arbre binaire complet dans lequel toutes les feuilles sont à la même hauteur dans l'arbre.



Théorème 4.1: Un arbre binaire de taille n a une hauteur moyenne de $log_2(n)$.

4.2-Arbre Binaire

Théorème 4.2: Il existe une bijection qui transforme un arbre planaire ayant *n* sommet en un arbre binaire complet ayant *2n+1* sommets.

Du fait de ce théorème, on ne considère dans un premier temps que le type arbre binaire que l'on nommera arbreBinaire.

4.2-Arbre Binaire

Chaque sommet permet d'accéder à deux sommets :

- le **fils gauche .**
- le **fils droit**.

Ce type sera nommé **sommet**.

Chaque sommet permet également d'accéder à l'objet qu'il stocke.

Un arbre binaire peut être vu comme un curseur indiquant le sommet racine. De la même manière un sommet est un curseur. On a donc :

```
arbreBinaire=curseur;
sommet=curseur;
```

4.2-Arbre Binaire : primitives

Le type **sommet** présente les primitives suivantes :

Accès :

```
fonction getValeur(val S:sommet):objet;
/* vaut NULL si le sommet n'existe pas /*

fonction filsGauche(val S:sommet):sommet;
/* vaut NIL si S n'a pas de fils gauche */

fonction filsDroit(val S:sommet):sommet;
/* vaut NIL si S n'a pas de fils droit */

fonction pere(val S:sommet):sommet;
/* vaut NIL si S est la racine de l'arbre */
```

4.2-Arbre Binaire : primitives

Modification :

```
fonction setValeur(ref S:sommet;val x:objet):vide;
/* affecte au sommet S la valeur x */
fonction ajouterFilsGauche(ref S:sommet,val x:objet):vide;
/* filsGauche(S)==NIL doit être vérifié */
fonction ajouterFilsDroit(ref S:sommet,x:objet):vide;
/* filsDroit(S)==NIL doit être vérifié */
fonction supprimerFilsGauche(ref S:sommet):vide;
/* filsGauche(S) est une feuille */
fonction supprimerFilsDroit(ref S:sommet):vide;
/* filsDroit(S) est une feuille */
fonction detruireSommet(ref S:sommet):vide;
/* S est une feuille */
```

4.2-Arbre Binaire : primitives

Modification :

```
fonction creerArbreBinaire(val Racine:objet):sommet;
fonction detruireArbreBinaire(ref A:arbreBinaire d'objet):vide;
```

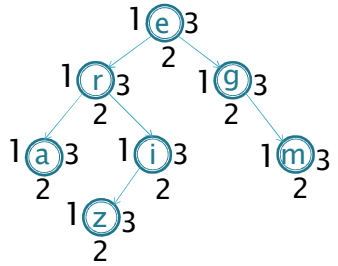
Détection de feuille

```
fonction estFeuille(val S:sommet):booléen;
  debut
    retourner(filsGauche(S)==NIL et filsDroit(S)==NIL)
  fin
```

4.2- Parcours d'un arbre binaire

Le parcours d'un arbre binaire consiste à donner une liste de sommets dans l'arbre.

Le *prototype* d'algorithme suivant permet d'effectuer les parcours selon les algorithmes associés aux *traitements* à partir d'un sommet de l'arbre.



4.2- Parcours d'un arbre binaire

```
fonction parcoursArbreBinaire(val A:arbreBinaire d'objet):vide;
//    Déclarations locales
    debut
//    traitement1;
      si estFeuille(A)alors
//      traitement2;
      sinon
//      traitement3;
        si filsGauche(A)!=NIL alors
//        traitement4;
          parcoursArbreBinaire(filsGauche(A));
//        traitement5;
        finsi
//      traitement6;
        si filsDroit(A)!=NIL alors
//        traitement7;
          parcoursArbreBinaire(filsDroit(A));
//        traitement8;
        finsi
//      traitement9;
      finsi
//    traitement10;
  fin
```

4.2- Parcours d'un arbre binaire

Complexité:

Si le `traitement`$_i$ a pour complexité $c_i(n)$, soit $c(n) = \sum_{i=1}^{10} c_i(n)$

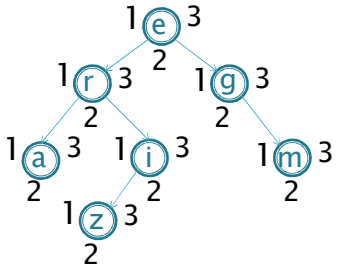
La complexité intrinsèque de l'algorithme est :

O(n c(n)).

On distingue quatre parcours qui conditionnent les algorithmes sur les arbres binaires.

- Le **parcours hiérarchique** ou en largeur
- **Parcours préfixe :** on affiche la racine, les sommets du sous arbre gauche, puis les sommets du sous arbre droit.
- **Parcours infixe**: on affiche les sommets du sous arbre gauche, puis la racine puis les sommets du sous arbre droit.
- **Parcours suffixe :** on affiche les sommets du sous arbre gauche, puis les sommets du sous arbre droit puis la racine.

4.2- Parcours d'un arbre binaire



Parcours hiérarchique : e,r,g,a,i,m,z
Parcours préfixe : e,r,a,i,z,g,m
Parcours infixe : a,r,z,i,e,g,m
Parcours postfixe : a,z,i,r,m,g,e

4.2- Parcours d'un arbre binaire

Affichage des valeurs des sommets pour un parcours donné
Soit un arbre étiqueté par des caractères. On considère que l'on dispose de la fonction qui affiche un caractère :

```
fonction afficher(val n:entier):vide;
```

Affichage dans le parcours préfixe

```
pas de déclarations locales.
traitement 2, 3 : afficher(valeur(A));
```

Affichage dans le parcours infixe :

```
pas de déclarations locales.
traitement 2, 6: afficher(valeur(A));
```

Affichage dans le parcours postfixe

```
pas de déclarations locales.
traitement 2, 9 : afficher(valeur(A));
```

4.2- Parcours d'un arbre binaire

Lister les étiquettes d'un arbre dans un tableau

```
fonction arbre2Tableau(val A:arbreBinaire d'entier;
      ref T:tableau[1..N] d'entier; ref i:entier):vide;
    debut
      i=i+1;
      T[i]=valeur(A);
      si !estFeuille(A)alors
         si filsGauche(A)!=NIL alors
          arbre2Tableau(filsGauche(A),T,i);
         finsi
         si filsDroit(A)!=NIL alors
          arbre2Tableau(filsDroit(A),T,i);
         finsi
      finsi
   fin
```

4.2- Parcours d'un arbre binaire

Hauteur d'un arbre binaire

```
fonction hauteurArbreBinaire(val s:sommet):entier
  debut
    si estFeuille(s)alors
      retourner(0)
    sinon
      var tmp1,tmp2:entier;
      tmp1=0;
      tmp2=0;
      si filsGauche(s)!=NIL alors
        tmp1= hauteurArbreBinaire(filsGauche(s));
      finsi
      si filsDroit(s)!=NIL alors
        tmp2=hauteurArbreBinaire(filsDroit(s));
      finsi
      retourner(1+max(tmp1,tmp2));
    finsi
  fin
```

4.2- Parcours d'un arbre binaire

Hauteur d'un arbre binaire : Autre version

```
fonction hauteurArbreBinaireSimp(val s:sommet):entier
  debut
    si s==NIL alors
      retourner(-1)
    sinon
       retourner(1+
             max(hauteurArbreBinaireSimp(filsGauche(s)),
                 hauteurArbreBinaireSimp(filsDroit(s))));
   finsi
 fin
```

Remarque : La fonction hauteurArbreSimp (même si elle est plus courte) a une complexité plus grande que la fonction hauteurArbreBinaire notamment en nombre d'appel récursif. Elle est moins performante.

4.2- Parcours d'un arbre binaire

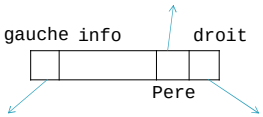
Taille d'un sous-arbre d'un arbre binaire complet.

```
fonction tailleArbreBinaire(val A: arbreBinaire):entier;

debut
    si estFeuille(A) alors
      retourner(1)
    sinon
      retourner(1+tailleArbreBinaire(filsGauche(A))
                 +tailleArbreBinaire(filsDroit(A))
    finsi
  fin
```

4.2- Arbre binaire implémentation

L'implémentation se fait par allocation dynamique. On définit

```
cellule=structure
         info:objet;
         gauche:sommet;
         droit:sommet;
         pere:sommet;
       finstructure

sommet=^cellule;
```



4.2- Arbre binaire implémentation

accès

```
fonction getValeur(val S:sommet):objet;
  debut
    retourner(S^.info);
  fin

fonction filsGauche(val S:sommet):sommet;
  debut
    retourner(S^.gauche)
  fin
```

4.2- Arbre binaire implémentation

modification

```
fonction creerArbreBinaire(val racine:objet):sommet;
      var tmp:sommet;
      debut
        new(tmp);
        tmp^.info=racine;
        tmp^.gauche=NIL;
        tmp^.droit=NIL;
        tmp^.pere=NIL;
        retourner(tmp)
      fin
```

4.2- Arbre binaire implémentation

modification

```
fonction ajouterFilsGauche(ref S:sommet,val x:objet):vide;
  var tmp:sommet;
  debut
    new(tmp);
    tmp^.info=x;
    tmp^.gauche=NIL;
    tmp^.droit=NIL;
    tmp^.pere=S;
    S^.gauche=tmp;
  fin
```

4.2- Arbre binaire implémentation

modification

```
fonction supprimerFilsGauche(ref S:sommet):vide;
  var tmp:sommet;
  debut
    tmp=S^.gauche;
    S^.gauche=NIL;
    delete(tmp);
  fin
```

4.2- Arbre binaire implémentation

modification

```
fonction detruireArbreBinaire(ref A:arbreBinaire d'objet):vide;
   debut
     si estFeuille(A)alors
       delete(A)
     sinon
       si filsGauche(A)!=NIL alors
         detruiresArbreBinaire(filsGauche(A));
       finsi
       si filsDroit(A)!=NIL alors
         detruireArbreBinaire(filsDroit(A));
       finsi
       delete(A);
     finsi
   fin
```

4.3- Arbre planaire : primitives

On peut définir un type abstrait sommetArbrePlanaire par les primitives suivantes:

accès

```
fonction getValeur(val S:sommetArbrePlanaire):objet;

fonction premierFils(val S:sommetArbrePlanaire):sommetArbrePlanaire;

fonction frere(val S:sommetArbrePlanaire):sommetArbrePlanaire;

fonction pere(val S:sommetArbrePlanaire):sommetArbrePlanaire;
```

4.3- Arbre planaire : primitives

modification

```
fonction creerArbrePlanaire(val racine:objet):
                                    sommetArbrePlanaire;
fonction ajouterFils(ref S:sommetArbrePlanaire,
                     val x:objet):vide;
/* ajoute un fils comme cadet */

fonction supprimerSommet(ref S: sommetArbrePlanaire):vide;
/* le sommet doit être une feuille */

fonction detruireArbrePlanaire(ref  S:
                               sommetArbrePlanaire):vide;
```

Un arbre planaire est de type sommetArbrePlanaire.
C'est un curseur.

4.3- Parcours d'un arbre planaire

Le parcours d'un arbre planaire consiste à donner une liste de tous les sommets.

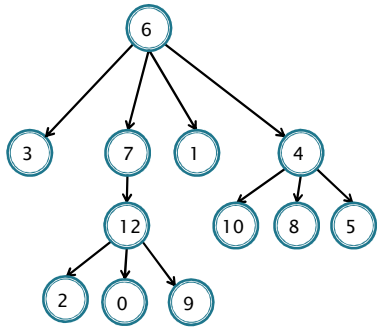
```
fonction parcoursArbrePlanaire(val A:sommetArbrePlanaire):vide;
// Déclarations locales
var f: sommetArbrePlanaire;
debut
// traitement1;
  f= premierFils(A);
  tant que f!=NIL faire
  // traitement2;
  parcoursArbrePlanaire(f);
  // traitement3;
  f=frere(f);
  // traitement4
fintantque
   // traitement5
fin
```

4.3- Parcours d'un arbre planaire

On distingue trois parcours qui conditionnent les algorithmes sur les arbres planaires :

- Le **parcours hiérarchique** qui s'effectue grâce à une file.
- Le **Parcours préfixe** : on liste la racine, les sommets de chaque sous arbre dans l'ordre où les sous arbres apparaissent.
- Le **Parcours suffixe** : on liste les sommets des sous arbres en ordre inverse puis la racine.

4.3- Parcours d'un arbre planaire



4.3- Implémentation du type arbrePlanaire

Implémentation dans le type arbreBinaire

L'implémentation dans le type `arbreBinaire` découle du théorème 4.2

On a alors
`arbrePlanaire=arbreBinaire.`

Pour un noeud donné :

- la primitive `filsGauche` donne accès au **premier fils** du noeud.
- la primitive `filsDroit` donne accès au **frère** du noeud.
- la primitive `pere` donne accès **soit au père du noeud soit à son frère précédent.**
  Il faut donc redéfinir la primitive `pere` pour les arbres planaires.

4.3- Implémentation du type arbrePlanaire

```
fonction pereArbrePlanaire(val S:
              sommetArbrePlanaire):sommetArbrePlanaire;
var T: sommetArbrePlanaire;
debut
  T=filsGauche(pere(S));
  tant que T!=S faire
    S=pere(S)
    T=filsGauche(pere(S));
  fintantque
  retourner(pere(S))
fin
```

En utilisant le théorème 4.1, cette dernière primitive à une complexité moyenne $O(\log_2 n)$.

4.3- Implémentation du type arbrePlanaire

Implémentation par allocation dynamique

Cette implémentation permet de diminuer le temps d'accès au père.

```
cellule=structure
       info:objet;
       premierFils:sommet;
       frere:sommet;
       pere:sommet;
Finstructure
```

Dans cette implémentation, on initialise parfois le champ frere du dernier frère à l'adresse du premier fils.
On peut vérifier aisément que les primitives sont toutes réalisables en O(1).

L'espace mémoire est le même que celui occupé par une implémentation dans le type arbreBinaire

4.3- Implémentation du type arbrePlanaire

Accès :

```
fonction getValeur(val s:sommetArbrePlanaire):objet;
  début
    retourner(s^.info)
  fin

fonction premierFils(val s:sommetArbrePlanaire):
                                    sommetArbrePlanaire;
  début
    retourner(s^.premierFils)
  fin
```

4.3- Implémentation du type arbrePlanaire

Accès :

```
fonction frere(val s : sommetArbrePlanaire) :
                                    sommetArbrePlanaire;
  début
    retourner(s^.frere)
  fin

fonction pere(val s : sommetArbrePlanaire):
                                    sommetArbrePlanaire;
  début
    retourner(s^.pere)
  fin
```

4.3- Implémentation du type arbrePlanaire

Modification :

```
fonction creerArbrePlanaire(val racine:objet):sommetArbrePlanaire;
  var tmp:^cellule;
  début
    new(tmp);
    tmp^.info=racine;
    tmp^.premierFils=NIL;
    tmp^.frere=NIL;
    tmp^.pere=NIL;
    retourner(tmp)
 fin
```

4.3- Implémentation du type arbrePlanaire

Modification :

```
fonction detruireArbrePlanaire(ref  s:sommetArbrePlanaire):vide;
  var tmp,f: sommetArbrePlanaire;
   début
      f= premierFils(s);
      tant que f!=NIL faire
          detruireArbrePlanaire(f);
          f=frere(f);
      fintantque
      supprimerSommet(s)
   fin
```

4.3- Implémentation du type arbrePlanaire

```
fonction ajouterFils(ref s:sommetArbrePlanaire,val x:objet):vide;
/* ajoute un fils comme cadet  cette fonction n'est pas en O(1) */
  var tmp:^celluleAP;
  début
    new(tmp);
    tmp^.info=x;
    tmp^.frere=NIL;
    tmp^.pere=s;
    tmp^.premierFils=NIL
     si s^.premierFils==NIL alors
      s^.premierFils=tmp;
    sinon
      r=premierFils(s);
      tantque frere(r)!=NIL faire
        r=frere(r)
      fintantque
      r^.frere=tmp
   fin
```

4.3- Implémentation du type arbrePlanaire

```
fonction supprimerSommet(ref s:sommetArbrePlanaire):vide;
/* le sommet doit être une feuille */
  var p,r,tmp:^celluleAP;
  début
    r=premierFils(pere(s));
    si r==s alors
      p=pere(s);
      p^.premierFils=s^.frere;
    sinon
      tantque frere(r)!=s faire
        r=frere(r);
      fintantque
      r^.frere=s^.frere;
    finsi
    delete(s)
  fin
```