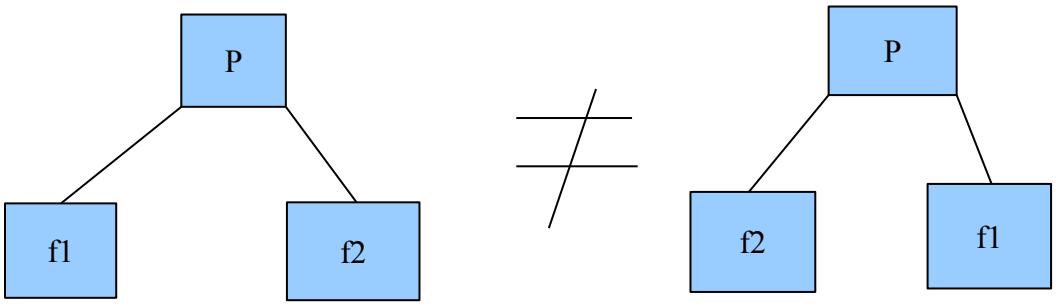
Les Arbres

La notion d'arbre permet de résoudre de nombreux problèmes de manière très élégante.

I. Exemples :
   - Système de fichier : voir « explorer »
   - Généalogie : Genealogie.JPG

II. Définitions :
   - Un arbre est un modèle d'une **hiérarchie** entre différents objets.
   - Les éléments de l'arbre s'appelle des nœuds.
   - L'élément unique le plus haut dans la hiérarchie s'appelle la racine (root).
   - Les éléments les plus bas dans la hiérarchie s'appellent des feuilles.
   - Un chemin entre 2 nœuds est la suite des nœuds permettant de passer de l'un à l'autre. La longueur du chemin est le nombre de nœuds qu'il contient moins un.
   - Un arbre est ordonné si les deux arbres suivants sont considérés comme différents.
     P le père, f1 le fils de droite et f2 le fils de gauche.
     P le père, f2 le fils de droite et f1 le fils de gauche.



   - Sinon l'arbre est non ordonné.

III. Applications :

1. Analyse d'expression algébrique :

Représenter l'expression algébrique suivante sous la forme d'un arbre :

(x + 4)*(5*(2x +1) +9)

2. La 3D : le CSG
   permet de décrire un objet complexe à partir d'un certain nombre de primitive.
3. Algorithme de tri : tri rapide.
   Les ABR : arbres binaires de recherche, construit ainsi :

   on insert les nombres : le sous arbre gauche du nœud x ne contient que des éléments de valeur inférieure à x.

   exemple : Si y < x alors insérer y comme fils gauche de x
   sinon insérer y comme fils droite de x.

4. Codage de Huffman : compression de fichier de texte à partir des fréquences d'apparition des caractères.

5. Application au jeu :

   - Le morpion : à partir d'une situation de jeu construire l'arbre des situations possibles.
   - les échecs : même principe.

IV. Opérations sur les arbres :

1. Insertions et suppressions :
   Insertions et suppression en feuille, c'est le plus simple. (exemple)
   Ailleurs cela entraine une restructuration de l'arbre : compliqué.
2. Les explorations ou parcours d'un arbre :
   « pour chaque nœud de l'arbre faire »

   - parcours préfixé : on commence par la racine puis on entreprend le parcours préfixé du premier sous-arbre puis le parcours préfixé du deuxième sous-arbre, etc...
   - parcours infixé : on commence par le parcours infixé du premier sous-arbre, puis on passe par la racine puis on effectue le parcours infixé du deuxième sous-arbre, etc...
   - parcours postfixé : on commence par le parcours postfixé du premier

sous-arbre, puis on effectue le parcours postfixé du deuxième sous-arbre, etc... et on termine par le racine.

Ces trois parcours peuvent se traduire plus facilement à l'aide d'algorithmes récursifs.

3. Restructuration : par exemple l'opération qui consiste à équilibrer un ABR.
4. ensembliste : par exemple fusion de deux arbres.
5. Relation de parenté : par exemple : est-il l'ancêtre de ?

V. Les implémentations :

- Avec des tableaux :
  - tableau des pères : Chaque élément du tableau contient l'index du père.
  - Arbre binaire : on sait qu'il y a au maximum deux fils donc :

- Avec des références ou pointeurs: procédé similaire aux listes chaînée.
  - Soit tableau de listes des fils.
  - Soit liste de listes des fils.
  - Cas de l'arbre binaire.

VI.Parlons récursivité :

Algorithme qui consiste en l'appel de la fonction f depuis la même fonction f.
Cette technique est fortement liée à la notion d'arbre.

Parcours prefixé d'un arbre :

```
procédure Prefixe( n: noeud)
début
afficher n
pour chaque fils f de n , s'il y en a,
depuis le plus à gauche jusqu'au plus à droite faire
        Prefixe( f)
fin
```

Parcours infixé d'un arbre :

```
procédure Infixe( n: noeud)
début
si n est une feuille alors
  afficher n
 sinon
  début
  Infixe( fils le plus à gauche de n)
  afficher n
  pour chaque fils f de n, à l'exception du fils le plus à gauche
     depuis le plus à gauche jusqu'au plus à droite faire
       Infixe( f)
  fin
fin
```

Parcours Postfixé d'arbre :

```
procédure Postfixe( n : noeud)
début
pour chaque fils f de n , s'il y en a,
       depuis le plus à gauche jusqu'au plus à droite faire
       Postfixe( f);
afficher n
fin
```