7 - Exemples d'algorithmes opérant sur un arbre. Applications.

I- Les arbres

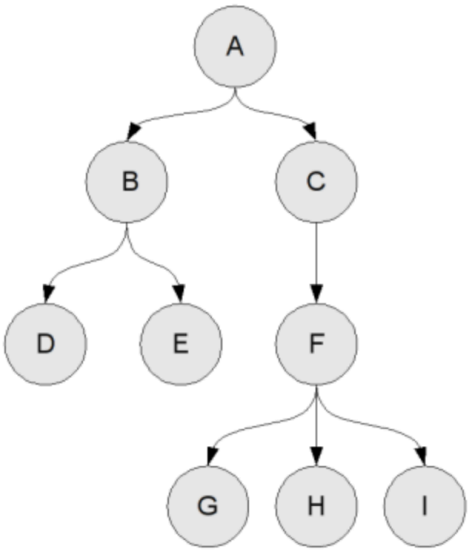
a- Généralité

On connaît les structures de données classiques tel que les tableaux, les listes, les files, les piles, … Ces structures sont dites linéaires, c'est-à-dire que les éléments sont stockés les uns derrière les autres.

Les arbres sont des structures non linéaires composées de nœuds qui possèdes ou non des enfants.

Il existe trois types de nœuds :

- La racine, c'est le seul nœud qui d'un arbre enraciné qui ne possède pas de parent
- Les feuilles, se sont des nœuds qui ne possède pas d'enfants
- Les nœuds internes sont des nœuds qui possèdent un parent et des enfants.



Un arbre sert avant tout a stocké un arbre, chaque nœud possède donc une valeur, ou peut être un objet choisit par le développer, l'arbre est donc étiqueté.

Définition :

B est parent de E

B est enfant de A

A est la racine

D et E sont des nœuds frères (siblings) et sont aussi des feuilles.

Le lien entre 2 nœuds est appelé une branche

On nome les arbres en fonction du nombre maximum d'enfant que peut avoir un nœud. Exemple un arbre possédant au plus 2 nœuds est appelé arbre binaire, 3 nœuds : arbre ternaire, 4 nœuds : arbre quaternaire (quadtree), n nœuds : arbre n-aire.

Nous allons nous intéresser plus particulièrement aux arbres binaires durant cette leçon. Il existe plus type d'arbre binaire :

- arbre dégénéré : les nœuds ont 1 seul enfant, c'est donc une liste chainée, à éviter.
- Arbre complet : toutes les feuilles on la même profondeur

On appelle également la profondeur d'un nœud la distance en terme de nœud par rapport à l'origine. Par convention, la racine est de profondeur 0.

Le degré d'un nœud est le nombre d'enfants qu'il possède.

La hauteur d'un arbre est la feuille possédant la plus grand profondeur. La hauteur d'un arbre est très importante, en effet beaucoup d'algorithme dépende de cette profondeur.

La taille d'un arbre est sont nombre de nœuds.

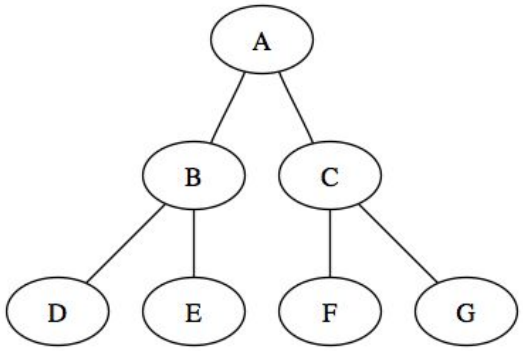
b- Représentation d'un arbre binaire en python

Avec une liste : celle ci sera de taille n égal au nombre de nœuds, chaque case du tableau comportera 3 éléments : l'étiquette du nœud, l'indice dans le tableau de sans fils gauche et celui de son fils droit.

Ex :

A	B	C	D	E	F
1	3	5	/	/	/
2	4	6	/	/	/

Mais avec l'arbre binaire complet ci-contre on peut aussi le représenter avec une liste à une dimension où le fils gauche du nœud à l'indice i sera à l'indice 2i+1 et le droit à l'indice 2i+2. Le parent d'un nœud quant à lui serait [(i-1)/2]



Avec des classes :

- Un classe Nœud avec comme attributs : string : valeur, Nœud : fils droit, Nœud : fils gauche
- Une classe Arbre_Binaire avec comme attribut : Nœud : racine
  Cette classe comportera aussi toute les méthodes (taille, hauteur,…)

II- Algorithme de base

Hauteur d'un arbre :

```
fonction hauteur ( node ) renvoie un entier
        si node est None
                renvoyer 0
        sinon
                renvoyer 1 + max (hauteur (node.G) hauteur(node.D)
        fin si
```

Taille d'un arbre :

```
fonction Taille ( node ) renvoie un entier
        si node est None
                renvoyer 0
        sinon
                renvoyer 1 + Taille (node.G) + Taille(node.D)
        fin si
```

Nombre de feuilles

```
fonction NbFeuilles ( node ) renvoie un entier
        si node est None
                renvoyer 0
        sinon si node.G is None && node.D is None
                renvoyer 1
        sinon
                renvoyer NbFeuilles(node.G) + NbFeuilles (node.D)
        fin si
```

Nombre de nœuds internes

```
fonction NbFeuilles ( node ) renvoie un entier
        si node est None
                renvoyer 0
        sinon si node.G is None && node.D is None
                renvoyer 0
        sinon
                renvoyer 1 + NbFeuilles(node.G) + NbFeuilles (node.D)
        fin si
```

Recherche sur différent type d'arbre

Sur un arbre binaire : parcours des nœuds jusqu'à trouver le bon (O(n))

Introduction arbre binaire de recherche (dichotomie) $O(\log_2(n))$

Min (feuille tout à gauche) et max (feuille tout à droite) dans un ABR

III- Algorithme plus complexe

Les parcours :

- Parcours à gauche et parcours à droite
- Parcours préfixé : on traite la racine puis on trait les enfants
- Parcours infixé : on traite le fils gauche, puis la racine et enfin le fils droit
- Parcours post fixé : On traite les enfants et ensuite la racine (calcul algébrique post fixé où les feuilles sont des valeurs et les nœuds des opérations)

Le parcours en profondeur permet d'explorer l'arbre en explorant jusqu'au bout une branche pour passer à la suivante.

Exemple récursif de parcours en profondeur :

```
fonction parcours_prof ( node)
si node n'est pas None alors
      traiter_racine(node) # préfixé
      parcours_prof (node.G)
      traiter_racine(node) # infixé
      parcours_prof(node.D)
      traiter_racine(node) # post fixé
fin si
```

Nous allons aborder un type de parcours un peu plus compliqué, c'est le parcours en largeur. Il s'agit d'un parcours dans lequel, on traite les noeuds un par un sur un même niveau.

```
fonction parcours_largeur(node)
        Creer_File F
        F.ajouter(node)
        tant que F n'est pas vide faire
                X <- F.extraire()
                Traiter_racine(X)
                si node.G n'est pas None alors
                        F.ajouter(node.G)
                fin si
                si node.D n'est pas None alors
                        F.ajouter(node.D)
                fin si
        fin faire
```

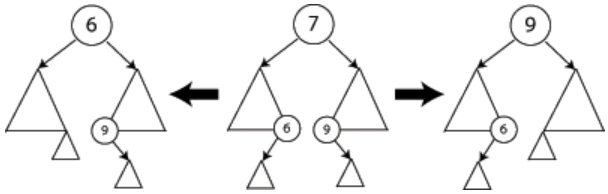
Pour un parcours en profondeur itératif, remplacer la file par une pile.

Ajout/ suppression d'un nœud dans un arbre binaire de recherche

L'insertion d'un nœud commence par une recherche : on cherche la clé du nœud à insérer ; lorsqu'on arrive à une feuille, on ajoute le nœud comme fils de la feuille en comparant sa clé à celle de la feuille : si elle est inférieure, le nouveau nœud sera à gauche ; sinon il sera à droite. La complexité est la même que pour la recherche : O(log n) dans le cas moyen et O(n) dans le cas critique.

Plusieurs cas sont à considérer, une fois que le nœud à supprimer a été trouvé à partir de sa clé :

- **Suppression d'une feuille** : Il suffit de l'enlever de l'arbre vu qu'elle n'a pas de fils.
- **Suppression d'un nœud avec un enfant** : Il faut l'enlever de l'arbre en le remplaçant par son fils.
- **Suppression d'un nœud avec deux enfants** : Supposons que le nœud à supprimer soit appelé *N* (le nœud de valeur 7 dans le graphique ci-dessous). On échange le nœud N avec son successeur le plus proche (le nœud le plus à gauche du sous-arbre droit - ci-dessous, le nœud de valeur 9) ou son plus proche prédécesseur (le nœud le plus à droite du sous-arbre gauche - ci-dessous, le nœud de valeur 6). Cela permet de garder une structure d'arbre binaire de recherche. Puis on applique à nouveau la procédure de suppression à *N*, qui est maintenant une feuille ou un nœud avec un seul fils.



Rotation gauche ? Rotation droite ? Double rotation ?

http://www.iro.umontreal.ca/~csuros/IFT2015/H07/materiel/abr.pdf

Hauteur ABR : $\log_2(n) <= h <= n - 1$

IV- Ouverture : les quadtree et la manipulation d'image bitmap

Représentation

Implémentation

Rotation

Symétrie