# Algorithmique avancée

—o000o– TD5 / TP5 –o000o—

**Barrière d'abstraction des arbres binaires [en TP]** La première étape pour travailler avec les arbres binaires est de programmer un ensemble de fonctions de base pour les manipuler appelé barrière d'abstraction.

1. Proposer un constructeur similaire à celui qui a été étudié en cours pour la construction des arbres binaires.
2. Écrire une fonction `abVide(A)` qui prend en argument un arbre binaire `A` et retourne `True` si et seulement si `A` est vide.
3. Écrire une fonction `estFeuille(A)` qui prend en argument un arbre binaire `A` et retourne `True` si et seulement si `A` est une feuille.

**Questions de cours** Les questions suivantes ont été abordées en cours. Servez-vous en comme d'un échauffement et pour vérifier que vous avez bien compris la logique de parcours d'un arbre binaire avant d'aborder les exercices qui suivent.

1. Écrire une fonction `nbNoeuds(A)` qui prend en argument un arbre binaire et qui détermine son nombre de nœuds. On considère que le nombre de nœuds d'un arbre vide est 0 et qu'une feuille possède 1 seul nœud.
2. Écrire une fonction `hauteur(A)` qui prend en argument un arbre binaire et retourne sa hauteur. La hauteur d'un arbre binaire est équivalente à la longueur du plus long chemin entre la racine et une feuille.

**Liste infixe, préfixe et suffixe** Le parcours d'un arbre binaire peut donner lieu à 3 listes de valeurs différentes selon l'ordre dans lequel on considère la racine, l'arbre gauche et l'arbre droit.

1. À partir de votre cours, et sans regarder les implémentations Java proposées, écrire les fonctions `infixe(A)`, `prefixe(A)` et `postfixe(A)` qui prennent en argument un arbre binaire et retourne une liste chaînée représentée par une structure de type `Noeud`.

**Affichage d'un arbre binaire** Afin de pouvoir visualiser le résultat de vos algorithmes, il peut être utile de disposer d'une fonction d'affichage d'un arbre binaire. Cette question est à faire en fin de séance (ou chez vous) pour vous aider dans votre travail, et est donc optionnelle.

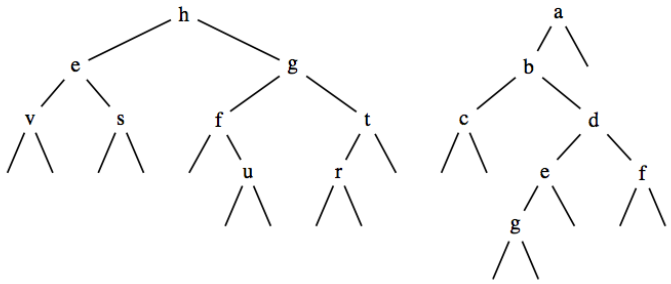



Figure 1: Illustration des arbres binaires à reproduire. Gauche : `arbre1` et droite : `arbre2`. (Illustration issue du cours de LI 101 - UPMC)

1. Écrire une fonction `decalage(n)` qui retourne une chaîne de caractères composée de $n$ caractères " " suivis des caractères `|->`. Ainsi `decalage(3)` retourne "  `|->`".

2. Écrire une fonction récursive `affichage(ab, prof)` qui prend en argument un arbre binaire et une profondeur et qui affiche la racine de l'arbre décalée de `prof` caractères et ensuite relance l'affichage sur les sous-arbres gauche et droit avec une profondeur égale à `prof + 1`.

3. Écrire enfin une fonction `affichage(ab)` qui prend en argument un arbre binaire et l'affiche en mode texte. Cette fonction se limite à appeler la fonction `affichage(ab, 0)`.

**Arbres binaires de test [en TP]** On s'intéresse désormais à la génération d'arbres binaires de test.

1. Écrire les fonctions `arbre1()` et `arbre2()` qui permettent de générer les arbres binaires représentés dans la figure 1.

2. Proposer une fonction `genereAB(deep, skew)` qui construit un arbre binaire contenant des valeurs aléatoires comprises entre 0 et 100. La hauteur maximale de l'arbre est indiquée par l'argument `deep`. Si `deep` vaut 0, l'arbre retourné est vide et si `deep` vaut 1 alors l'arbre retourné est une feuille contenant une valeur aléatoire. Pour les hauteurs supérieures, on construit récursivement un arbre de hauteur $n$ en générant une racine aléatoirement et en y adjoignant 2 sous-arbres gauche et droit de hauteur $n-1$. L'argument `skew` est un pourcentage (exprimé par exemple entre 0 et 100) qui indique la chance pour un sous-arbre qu'il soit effectivement construit comme indiqué précédemment. Il suffit pour cela d'effectuer un tirage aléatoire entre 0 et 100 et si la valeur est inférieure à `skew` alors on construit l'arbre sinon on retourne un arbre vide. Note : pour générer un nombre aléatoire entre 0 et 100 on peut utiliser la fonction `randint(0,100)` du module `random`.

## À propos des feuilles dans un arbre binaire

1. Écrire la fonction `nbFeuilles(ab)` qui prend en argument un arbre binaire et retourne son nombre de feuilles. On pourra utiliser la fonction `estFeuille(ab)` de la barrière d'abstraction écrite précédemment.

2. Écrire la fonction `estVoyelle(lettre)` qui prend en argument un caractère et retourne vrai (`True`) si et seulement si le caractère `lettre` est une voyelle ('a', 'e', 'i', 'o' , 'u', 'y').

3. Écrire enfin la fonction `nbFeuillesVoyelle(ab)` qui prend en argument un arbre binaire dont les étiquettes sont des caractères et retourne son nombre de feuilles contenant une voyelle.

**Branches gauche et droite, chemin** On s'intéresse dans cet exercice aux branches gauche et droite d'un arbre binaire, c'est-à-dire la liste des étiquettes obtenue lorsque l'on parcourt récursivement l'ensemble des sous-arbres gauches ou droits d'un arbre binaire. Par exemple, pour l'arbre binaire `arbre1`, la branche gauche est `(h, e, v)` et la branche droite est `(h, g, t)`.

1. Écrire la fonction `brancheGauche(ab)` qui retourne la branche gauche de l'arbre binaire `ab`.

2. Écrire la fonction `brancheDroite(ab)` qui retourne la branche droite de l'arbre binaire `ab`.

3. Écrire la fonction `chemin(ab, C)` qui retourne la liste des étiquettes rencontrées dans l'arbre `ab` en suivant les indications contenues dans le chemin `C`. Ce chemin `C` prend la forme d'une liste qui ne contient que des symboles `G` ou `D`. Ceux-ci indiquent s'il faut suivre le sous-arbre gauche ou le sous-arbre droit respectivement. Dès lors que le chemin n'est plus correct (l'arbre n'existe plus dans la direction demandée) ou que la liste `L` est vide on retourne la liste vide.

**Nombre de Strahler d'un arbre binaire** Dans le cas d'un arbre binaire représentant une expression arithmétique, le nombre de Strahler est le nombre minimal de registres nécessaires pour évaluer cet arbre au moyen d'une calculatrice avec laquelle on peut faire trois types d'opérations :

- mettre une valeur dans un registre
- mettre le résultat de `reg1 op reg2` dans `reg3` (un même registre pouvant apparaître plusieurs fois)
- afficher le contenu d'un registre

Le nombre de Strahler est défini récursivement comme suit :

- si l'arbre est vide, son nombre de Strahler vaut 0 ;

- si les sous-arbres gauche et droit ont le même nombre de Strahler `S`, alors l'arbre aura pour nombre de Strahler la valeur `S + 1` ;
- sinon, le nombre de Strahler de l'arbre binaire est égal au maximum du nombre de Strahler de ses sous-arbres gauche et droit.

1. Écrire la fonction `strahler(ab)` qui prend en argument un arbre binaire et retourne son nombre de Strahler selon la méthode décrite précédemment.

**Dénombrement à un certain niveau**

1. On s'intéresse tout d'abord à l'écriture d'une fonction pour déterminer le nombre de nœuds à un niveau k (entier positif) dans un arbre binaire B. On considérera que la racine d'un arbre binaire non vide est au niveau 1 et que de manière général, le niveau d'un nœud est égal à celui de son père augmenté de 1.
2. Écrire maintenant une fonction qui permet de déterminer le nombre de feuilles à un niveau donné k (entier positif) en suivant les mêmes contraintes que précédemment.

**Égalité entre deux arbres binaires** On s'intéresse maintenant à la définition d'une fonction pour déterminer si deux arbres binaires sont égaux. On considère ici que 2 arbres binaires sont égaux s'ils possèdent les mêmes valeurs d'étiquettes et la même structure : si l'un possède (resp. ne possède pas) un fil (gauche ou droit) l'autre doit (resp. ne doit pas) en avoir un. On pose qu'un arbre vide est égal à un autre arbre vide et que deux feuilles sont égales si elles possèdent la même étiquette.

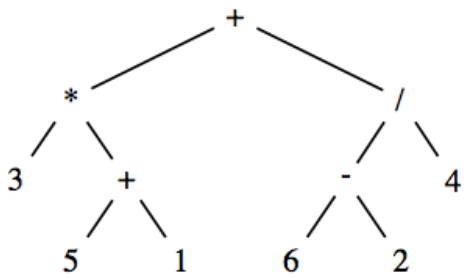
**Miroir d'un arbre binaire** L'arbre miroir d'un arbre binaire B est défini récursivement de la manière suivante :

- il possède la même racine que l'arbre binaire B,
- son fil gauche est l'arbre miroir du fil droit de B,
- son fil droit est l'arbre miroir du fil gauche de B.

Le miroir d'un arbre vide est un arbre vide et le miroir d'une feuille est elle-même.

1. Écrire une fonction qui prend en argument un arbre binaire et retourne son arbre miroir. Vous proposerez différents tests pour vérifier votre fonction. Notamment que vaut l'arbre miroir de l'arbre miroir d'un arbre binaire ?
2. Écrire une fonction prenant en argument 2 arbres binaires quelconques A et B et retourne `true` si et seulement si B est l'arbre miroir de A et `false` sinon.
3. Vérifier que le miroir du miroir d'un arbre est égal à cet arbre.

**Expression arithmétique [en TP]** Le but de cet exercice est de représenter une expression arithmétique par un arbre binaire. Par exemple, l'expression `(+ (* 3 (+ 5 1)) (/ (- 6 2) 4))` sera représentée par l'arbre binaire suivant (exemple tiré du cours LI101 UPMC) :




1. Donner la liste chaînée correspondante à l'expression précédente et la construire en utilisant le type `Liste` de liste chaînée vu lors des séances précédentes. L'idée est qu'une liste décrivant une expression arithmétique contienne soit une seule valeur qui est une constante numérique, soit 3 valeurs exactement : un opérateur binaire ("+", "-", ...) et ses deux opérandes. Les opérandes peuvent être soit des nombres, soit des listes correspondant aux expressions plus internes.

2. Écrire une fonction d'affichage d'une telle expression arithmétique sous forme de liste chaînée. Vous devrez dissocier les cas où la valeur de la liste expression est une chaîne de caractères (`str`), un entier (`int`) ou une liste chaînée.

3. Écrire la fonction calculant l'arbre associé à une expression sous forme de liste comme définie précédemment. On supposera ici que l'expression de départ est une expression arithmétique bien formée, composée uniquement d'opérateurs binaires et de constantes numériques.

4. Proposer maintenant une fonction pour calculer la valeur associée à l'expression sous forme d'arbre binaire.

**Arbre binaire de recherche** On s'intéresse dans cet exercice aux arbres binaires de recherche numériques qui sont des arbres binaires étiquetés par des nombres tels que tout arbre de recherche binaire non vide possède les propriétés suivantes :

- l'étiquette de la racine est supérieure à toutes les étiquettes de son sous-arbre gauche,
- l'étiquette de la racine est inférieure à toutes les étiquettes de son sous-arbre droit,
- les sous-arbres gauche et droit sont des arbres binaires de recherche.

1. Écrire une fonction qui prend en argument un arbre binaire de recherche ainsi qu'une valeur entière et retourne `True` si et seulement si la valeur est présente dans l'arbre binaire de recherche et `False` sinon.

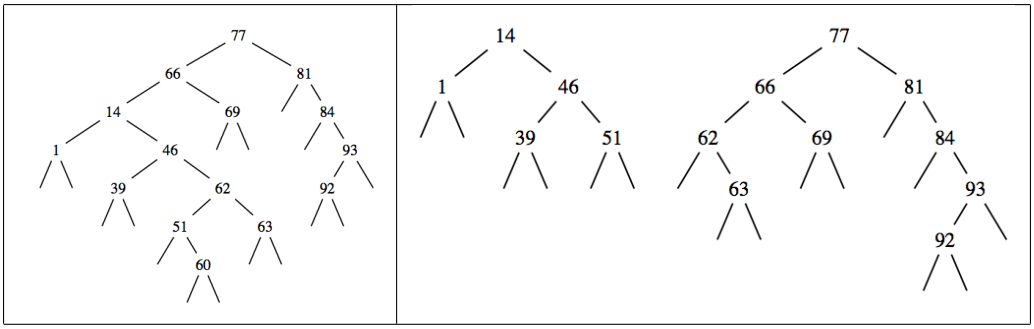



Figure 2: Gauche : arbre binaire de recherche initial. Droite : couple d'arbres obtenus par coupure sur la valeur 60.

2. Écrire une fonction qui étant donné un arbre binaire de recherche et une valeur entière retourne le nouvel arbre binaire dans le lequel la valeur a été insérée au niveau des feuilles. Si la valeur est déjà présente dans l'arbre binaire de recherche, celui-ci sera retourné tel quel.

3. Écrire désormais une fonction qui retourne un arbre binaire de recherche contenant k valeurs générées aléatoirement. Vous pouvez utilisez la fonction précédente d'ajout au niveau des feuilles.

4. Écrire désormais une fonction de coupure qui, étant donnés un nombre x et un arbre binaire de recherche A, renvoie un tableau contenant deux arbres binaires de recherche :

   - le premier est composé des éléments de ABR strictement inférieurs à x,
   - et le second est composé des éléments de ABR strictement supérieurs à x.

   La figure 2 illustre le mécanisme de coupure et son résultat attendu.

   **Indications :** cette fonction récursive est (très) difficile à coder sans avoir l'intuition de ce qu'il faut faire. Le cas le plus simple est celui de la coupure d'un arbre vide qui produit un tableau contenant 2 arbres binaires vides. Ensuite, le second cas qui est simple à considérer est celui dans lequel l'étiquette est la valeur de coupure recherchée. Dans ce cas, le tableau retourné contient comme premier élément le sous-arbre gauche et comme second élément le sous-arbre droit. Enfin, dans le cas général, il faut considérer la coupure soit du sous-arbre gauche, soit du sous-arbre droit en fonction de la valeur de coupure et ensuite reconstruire le nouveau couple d'arbres.

5. À partir de la fonction précédente, proposez une fonction pour l'insertion d'une valeur à la racine d'un arbre binaire de recherche.

6. Écrire maintenant un prédicat qui retourne `True` si et seulement si l'arbre binaire contenant des entiers passé en argument est un arbre binaire de recherche et `False` sinon.

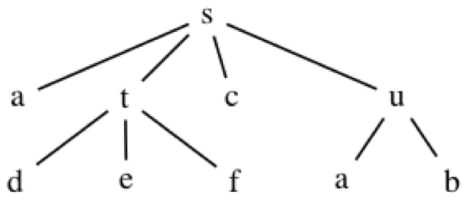



Figure 3: Exemple d'arbre général dont les étiquettes portent des lettres.

7. Écrire une fonction qui retourne la liste des éléments d'un arbre binaire de recherche obtenue par un parcours infixe de l'arbre. Celui-ci consiste à parcourir d'abord le sous-arbre gauche, puis l'étiquette et enfin le sous-arbre droit.

8. On s'intéresse enfin au tri par ordre croissant de valeurs entières contenues dans une liste chaînée. Pour cela, on se propose de parcourir la liste et de représenter ses valeurs sous la forme d'un arbre binaire de recherche. Il est ensuite possible de récupérer la liste des valeurs triées en construisant la liste infixe des valeurs de cet arbre binaire de recherche. Proposez la fonction qui réalise un tel traitement.

9. Implémenter la rotation gauche et la rotation droite d'un arbre binaire de recherche. Cette méthode a été étudiée en cours.

**Arbres généraux** On s'intéresse désormais aux arbre s généraux dans lesquels le nombre de fils de chaque nœud n'est pas limité à deux comme dans les arbres binaires. On rappelle les fonctions de la barrière d'abstraction des arbres généraux :

- pour un arbre général `A`, l'accès à la valeur de la racine à l'aide de la propriété `A.etiquette` ;
- pour un arbre général `A`, l'accès aux sous-arbres contenus dans sa forêt à l'aide de la propriété `A.foret` ;
- le constructeur `AG(etiquette, foret)` dans lequel `foret` désigne la liste chaînée contenant les arbres fils ;
- une fonction d'affichage d'un arbre général pour vérifier vos fonctions `affichageAG(A)`.

1. Soit l'arbre général de la figure 3. Proposez une fonction qui retourne cet arbre général

2. Écrire une fonction qui prend en argument l'arbre de la question précédente et retourne un arbre général de même structure contenant des entiers à la place des lettres. Les correspondances de valeurs entre caractères et entiers est l'indice de la lettre dans l'alphabet à savoir (en partant de 1 la numération) : a = 1, b = 2, c = 3, d = 4, e = 5, f = 6, r = 18, s = 19, t = 20 et u = 21. Votre fonction doit évidemment parcourir l'arbre initial pour construire son double "dynamiquement".

Il est possible d'automatiser la conversion d'un caractère `c` en `int` en Python à l'aide de la fonction `ord(c)`. Si l'on retire de cette valeur `ord('a') - 1` (pour que 'a' ait la valeur 1) on obtient le résultat escompté.

3. Écrire un prédicat `agFeuille(AG)` qui détermine si un arbre général est réduit à une feuille. Cette fonction permettra de compléter la barrière d'abstraction et pourra être utilisée dans les questions suivantes.

4. Écrire une fonction qui retourne une liste (Python !) contenant le nombre de noeuds et le nombre de feuilles d'un arbre général. La fonction devra ne réaliser qu'une seule passe sur l'arbre général pour produire son résultat.

5. On définit récursivement la profondeur d'un arbre `A` comme suit :

   - si `A` est une feuille alors sa profondeur est 1 ;
   - sinon la profondeur de `A` est égale au maximum de la profondeur de ses sous-arbres agmentée de 1.

   Écrire une fonction qui retourne la profondeur d'un arbre général. Vous pourrez proposer deux solutions, l'une utilisant un schéma de récursion avec 2 fonctions (une pour les arbres et l'autre pour les forêts) et la seconde qui utilise les fonctionnelles.

6. Proposer une fonction qui, étant donné un arbre général `A` et un entier strictement positif `k` représentant un niveau de profondeur dans cet arbre, retourne le nombre de nœuds dans `A` à la profondeur `k`.

**Strahler d'un arbre général** On définit récursivement le nombre de Strahler d'un arbre général comme suit :

- si l'arbre est réduit à une feuille son nombre de Strahler vaut 0,
- sinon le calcul du nombre de Strahler dépend des valeurs des nombres de Strahler de tous les arbres de la forêt sous-jacente :
- si tous les arbres de la forêt sous-jacente ont même nombre de Strahler S, alors le nombre de Strahler de l'arbre général est S + 1,
- sinon le nombre de Strahler de l'arbre général est égal au maximum des nombres de Strahler des arbres de la forêt sous-jacente.

Le but de cet exercice est de calculer le le nombre de Strahler d'un arbre général.

1. Écrire un prédicat qui prend en argument une liste chaînée et retourne `True` si et seulement si tous les éléments de la liste sont égaux et `False` sinon. On considérera qu'une liste vide ou une liste ne possédant qu'un seul élément retourne `True`.

2. Écrire maintenant une fonction qui prend en argument un arbre général et retourne un entier positif qui représente son nombre de Strahler.

   **Indications :** pour parvenir à calculer le résultat voulu, il faut que la fonction qui travaille sur la forêt ne retourne pas une valeur mais la liste des nombres de Strahler des arbres appartenant à cette forêt. Il est ensuite possible de déterminer si toutes les valeurs sont identiques ou bien d'en calculer le maximum au besoin en réutilisant les fonctions définies dans les questions précédentes.

### Liste des nœuds des arbres généraux

1. Écrire une fonction qui prend en argument un arbre général et retourne la liste de ses feuilles de gauche à droite.

2. Écrire une fonction qui prend en argument un arbre général et retourne la liste de ses nœuds en ordre préfixe. Cet ordre est défini comme suit pour un arbre général `A` :

   - si `A` est vide alors la liste est vide ;
   - si `A` est une feuille alors la liste préfixe ne contient qu'un seul élément qui est l'étiquette de l'arbre ;
   - sinon, la liste est égale à la concaténation de l'étiquette de la racine de `A` et des listes préfixes des sous-arbres de `A` dans leur ordre d'apparition dans la forêt.

3. Écrire maintenant une fonction qui prend en argument un arbre général et retourne la liste de ses nœuds en ordre suffixe. Cet ordre est défini comme suit pour un arbre général `A` :

   - si `A` est vide alors la liste est vide ;
   - si `A` est une feuille alors la liste préfixe ne contient qu'un seul élément qui est l'étiquette de l'arbre ;
   - sinon, la liste est égale à la concaténation des listes suffixes des sous-arbres de `A` dans leur ordre d'apparition dans la forêt avec la liste contenant l'étiquette de la racine de `A`.

4. Écrire une fonction qui étant donné un arbre général, retourne sa branche gauche sous la forme d'une liste chaînée. La branche gauche d'un arbre général `A` est définie récursivement comme suit :

   - si `A` est un arbre vide, sa branche gauche est la liste vide
   - si `A` est une feuille alors sa branche gauche est la liste contenant son étiquette
   - sinon la branche gauche de `A` est égale à la concaténation de l'étiquette de la racine de `A` et de la branche gauche du premier sous-arbre de `A`.

5. Écrire une fonction qui étant donné un arbre général, retourne sa branche droite sous la forme d'une liste chaînée. La branche droite d'un arbre général `A` est définie récursivement comme suit :

   - si `A` est un arbre vide, sa branche droite est la liste vide ;
   - si `A` est une feuille alors sa branche droite est la liste contenant son étiquette ;
   - sinon la branche droite de `A` est égale à la concaténation de l'étiquette de la racine de `A` et de la branche droite du dernier sous-arbre de `A`.