Université Paris Cité

L2 informatique - Année 2023–2024

TD d'Éléments d'Algorithmique n° 9

*** Les exercices marqués d'une étoile sont à faire à la maison.**

Dans la suite nous allons utiliser la classe `Arbre` et la classe `Noeud` qui representent respectivement un arbre et un noeud de l'arbre.

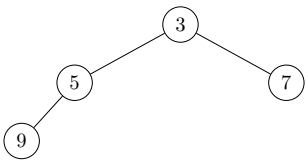
```
class Arbre{
  Noeud racine;
}
class Noeud{
  Noeud G;
  Noeud D;
  int valeur;
}
```

Exercice 1. *Un tout petit peu de combinatoire.*

1. Donnez une borne supérieure sur le nombre de nœuds dans un arbre binaire en fonction de la hauteur de l'arbre.
2. On appelle *arête* un lien entre un nœud et ses enfants (les traits que l'on représente sur le dessin). Si un arbre binaire a n nœuds, combien a-t-il d'arêtes ?

Exercice 2. *La structure.*

Donnez le code permettant de créer l'arbre binaire dessiné ci-dessous



Exercice 3. *Fonctions d'Arbres.*

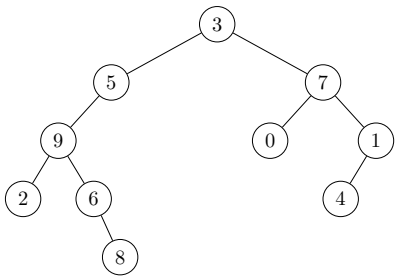
1. Écrivez les fonctions *récursives* qui renvoient respectivement la taille, la hauteur, le nombre de feuilles et la valeur maximale des nœuds d'un arbre binaire.
2. Un peigne gauche est un arbre binaire dans lequel l'enfant droit de chaque nœud interne est une feuille. Écrivez une fonction qui prend un arbre binaire en paramètre et retourne 1 si c'est un peigne gauche, 0 sinon.

Exercice 4. *Parcourir un Arbre.*

1. Rappelez le code récursif du parcours préfixe d'un arbre binaire.
2. Le parcours *infixe* d'un arbre de racine `v` est défini récursivement par :
   (a) parcours (infixe) de `v.G`
   (b) visite de `v`
   (c) parcours (infixe) de `v.D`

   et le parcours *postfixe* par :
   (a) parcours (postfixe) de `v.G`
   (b) parcours (postfixe) de `v.D`
   (c) visite de `v`

   Donnez le code récursif des parcours infixe et postfixe d'un arbre binaire.



3. Lorsque l'on visite un nœud, on affiche sa valeur. Qu'obtient-on sur l'arbre ci-dessus pour les trois parcours ?
4. Un autre arbre pourrait-il avoir le même parcours préfixe ? Même question pour les parcours et infixe et postfixe.
5. Examinez le code itératif suivant (remarquez que les valeurs stockées dans la pile `p` sont des noeuds.).

```
1  fonction(Arbre a){
2     p= new Pile()
3     push(p,a.racine)
4     while(p non vide){
5          Noeud n = pop(p)
6          print(n.val)
7          if n.D!=nil    {push(p,n.D)}
8          if n.G!=nil    {push(p,n.G)}
9     }
10 }
```

   Que produit cette fonction ?
6. Supposons qu'on remplace la pile `p` par une file dans l'algorithme de la question précédente. Quel affichage obtient-on dans l'arbre donné sur la figure ? Quelle propriété possède le parcours obtenu ?

Exercice 5. *Arbres binaires de recherche.*

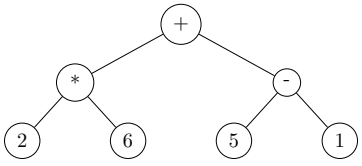
1. Dessinez étape par étape l'arbre binaire obtenu en insérant successivement

$$8, 1, 1, 8, 6, 3, 9, 1, 1, 8.$$

2. Donnez le parcours infixe de l'arbre obtenu. Que remarquez vous ?
3. En déduire un algorithme de tri prenant une liste d'entiers en paramètre et qui renvoie la liste triée.
4. Quel est le pire des cas pour la complexité ?

Exercice 6. *Notation Polonaise Inverse le retour.*

Dans cet exercice, les nœuds de l'arbre ont comme valeurs possibles un entier (s'il s'agit d'une feuille) ou un caractère parmi `+,-,*,/` (si c'est un nœud interne). Comme vu à un TD précédant, on peut représenter une expression algébrique comme `2*6+(5-1)` par la notation `26*51-+`, ou par l'arbre suivant :



1. À quel parcours de l'arbre la notation polonaise inverse correspond-elle ?
2. * Écrivez une fonction qui, à partir d'une expression en notation polonaise inverse (vue comme une liste), construit l'arbre représentant l'expression. On pourra utiliser une pile pour construire l'arbre.
3. * Écrivez une fonction récursive qui prend l'arbre et construit l'expression parenthésée usuelle, avec toutes les parenthèses, même celles qui ne sont pas nécéssaires (par exemple `((2*6)+(5-1))`).