Structures de données en C

TD/TP 4 – Les arbres

Exercice 1 : Arbres binaires

1. Implanter la spécification ABin en utilisant les déclarations suivantes :

```
typedef ... arbre;
arbre empty();
int is_empty(const arbre);
arbre sad(arbre);
arbre sag(arbre);
```

2. Écrire les fonctions **taille** et **hauteur.**

On dit qu'un arbre est **filiforme** (*dégénéré*) si chaque nœud a au moins un fils vide :

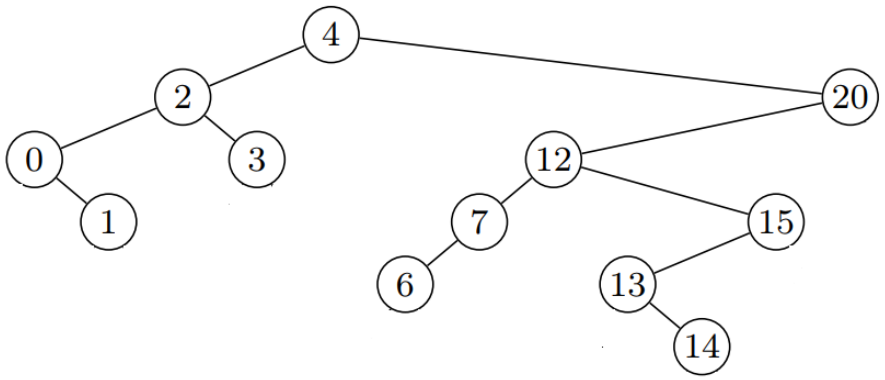
3. Dessiner tous les arbres filiformes de taille 1, 2 et 3.
4. Combien y a-t-il d'arbres filiformes de taille n ?
5. Écrire une fonction **est_filiforme**.
6. Montrer que pour tout arbre filiforme **t** on a l'égalité **taille(t) = hauteur(t)**

On dit qu'un arbre est un **peigne gauche** si le fils droit de chaque nœud est vide.
On définit de même les peignes droits.

7. Écrire une fonction **peigne(n)** qui retourne le peigne gauche de taille **n** donné.
8. Exécuter **est_filiforme(peigne(3))**. Qu'en pensez-vous ?

Exercice 2 : Parcourt d'arbres binaires

1. 45 15 65 98 78 43 23 35 2 87 9 19 20 14 61
2. Donner l'ordre de parcours des nœuds de l'arbre suivant, pour l'ordre préfixe. Quel est l'affichage si l'on affiche les étiquettes dans cet ordre ?



3. Même question pour les ordres infixe, et postfixe.
4. Même question pour un parcours en largeur ?
5. Écrire un algorithme qui étant donné un arbre retourne la liste des étiquettes de cet arbre dans l'ordre de parcours préfixe.

Exercice 3 : Arbres binaires de recherches, suppression

Pour chacun des algorithmes suivants, on donnera la complexité.

1. Écrire une fonction qui étant donné un **ABR** non vide retourne la plus petite valeur de cet **ABR**.
2. Écrire une fonction qui étant donné un **ABR** non vide supprime et retourne la plus petite valeur de cet **ABR**.
3. Écrire une fonction qui étant donné un **ABR a** supprime la racine de cet **ABR**. Si **a** et les deux sous arbres gauche et droit sont tous non vide, on remplacera la racine par le plus petit nœud du sous arbre gauche de **a**. Pourquoi ce choix est-il correct ?
4. Écrire une fonction qui étant donné un élément ***e*** et un arbre **a** supprime **e** de **a** s'il est dans **a** et sinon laisse **a** inchangé.

Exercice 4 : Effectuer un calcul sur un arbre

Écrire une fonction qui retourne la moyenne des éléments positifs et celle des éléments négatifs d'un arbre (0 est considéré ici comme un positif).

Exercice 5 : Extraire une liste d'éléments d'un arbre

Écrire une fonction qui retourne la liste chaînée des éléments d'un arbre d'entiers, qui ne sont pas divisibles par leur parent et que leur parent ne divise pas.
Citez au moins deux façons de construire des arbres pour lesquels cet algorithme retourne nécessairement la liste de tous ses éléments.

Exercice 6 : Partitions d'arbres binaires de recherches

Écrire un programme qui étant données **Ar** un **ABR** et **e** un élément, retourne un **ABR Ar_e** qui ne contient que les éléments de **Ar** inférieur à la valeur à **e**.
Contrainte : L'algorithme ne doit parcourir qu'un seul chemin dans l'**ABR** sans comparer tous les nœuds à **e**.

Problème : Le triangle Chinois de Pascal

Isaac Newton a donné une formule générale du développement de la puissance n^{ième} d'un binôme :

$$(x+y)^n = \sum_{k=0}^{n} C_n^k x^k y^{n-k}$$

Ou C_n^k le « coefficient binomial », alternativement dénoté par $\binom{n}{k}$ se calcule par la formule :

$$C_n^k = \frac{n!}{k!\,(n-k)!}$$

En combinatoire, C_n^k décompte le nombre de combinaisons de k éléments parmi n.

Le triangle de Pascal, ou « triangle arithmétique Chinois » date du XIII[e] siècle et n'a donc pas été inventé par Blaise Pascal.

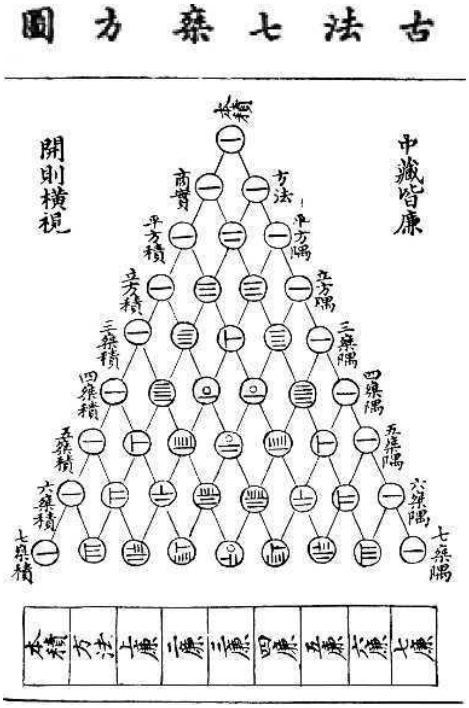


Figure 1 Triangle de Yang Hui – Source :
http://en.wikipedia.org/wiki/File:Yanghui_triangle.gif

Vous avez certainement déjà rencontré cet objet mathématique qui exploite une méthode de construction par récurrence pour produire un arrangement géométrique arborescent des coefficients binomiaux.
En effet, il est trivial de constater que : $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$ avec pour chaque ligne : $C_n^0 = \frac{n!}{0!(n-0)!} = 1$. En particulier : $C_0^0 = \frac{0!}{0!(0-0)!} = 1$.
Il vous est demandé de réaliser l'algorithme itératif qui construit le triangle sous forme d'un arbre binaire non équilibré, jusqu'à un niveau de profondeur n donné, dont la racine contient la valeur C_0^0, le fils gauche de la racine C_1^0 , le fils droit de la racine C_1^1, puis, dès la profondeur n = 2, dont les fils gauche et droit du fils le plus à gauche de profondeur n – 1 (contenant C_{n-1}^0) contiennent respectivement les valeursC_n^0 et C_n^1, et dont tous les noeuds situés à droite de ces deux fils (contenant $C_n^{k>1}$) soient les fils droits de leur père de niveau n – 1 (contenant $C_{n-1}^{k-1>0}$).

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
```

Figure 2: exemple

1. Écrire une fonction qui permet de générer un arbre contenant les éléments du triangle chinois.

2. Ecrire une fonction qui permet de calculer $(x + y)^n$ en utilisant l'arbre combinatoire généré.