1 Hauteur d'un arbre binaire

On appelle hauteur d'un arbre le plus long chemin de la racine à une de ses feuilles.

Exercice 1 *Étant donné un arbre de hauteur h, quel est le nombre minimal d'éléments qu'il peut contenir ?*

Exercice 2 *Étant donné un arbre de hauteur h, quel est le nombre maximal d'éléments qu'il peut contenir ?*

Exercice 3 *Inversement, quelle est la hauteur minimale d'un arbre contenant n éléments ?*

Exercice 4 *On considère maintenant uniquement des arbres binaires dont les noeuds ont soit deux fils (on parle alors de* nœud interne*), soit aucun fils (on parle de* feuille*). Démontrer que le nombre de nœuds interne est égal au nombre de feuilles moins un.*

2 Arbres binaires de recherche

2.1 Définition

On définit un *arbre binaire de recherche* comme un arbre binaire dont les clés appartiennent à un ensemble totalement ordonné et vérifient la propriété suivante :

> *Toute clé d'un nœud de l'arbre est supérieure ou égale à celles des descendants de son fils gauche, et inférieure ou égale à celle des descendants de son fils droit.*

L'arbre représenté sur la figure 1 est donc un arbre binaire de recherche.

Exercice 5 *Montrer que tout sous-arbre d'un arbre binaire de recherche est lui-même un arbre binaire de recherche.*

Exercice 6 *Montrer qu'un parcours infixé d'un arbre binaire de recherche imprime les clés dans l'ordre croissant. En déduire un nouvel algorithme de tri.*

2.2 Opérations sur les arbres binaires de recherche

Exercice 7 *Écrire le pseudo-code d'une fonction recherchant un élément dans un arbre binaire de recherche.*
Quelle est la complexité de la recherche d'une clé, dans le cas où cette recherche est un succès et dans le cas où c'est un échec ?

Exercice 8 *Écrire le pseudo-code d'une fonction insérant un élément dans un arbre binaire de recherche.*
Quelle est la complexité de l'opération d'insertion ?

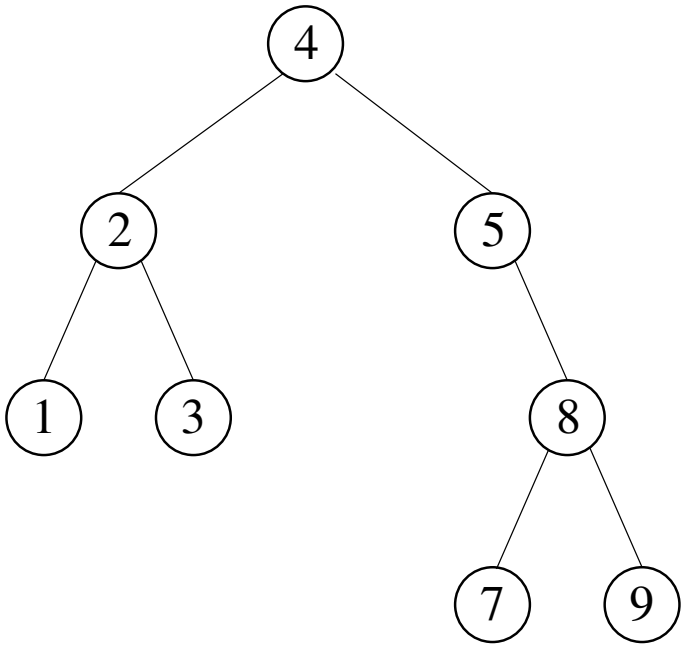

FIGURE 1 – Exemple d'arbre binaire de recherche obtenu en ajoutant successivement les entiers 4, 5, 8, 9, 2, 3, 1 et 7 dans un arbre initialement vide

Exercice 9 *En partant d'un arbre binaire de recherche initialement vide, on ajoute successivement les entiers suivants :*

$$6, 8, 13, 7, 4, 1, 5, 11, 3, 14, 10, 2, 9, 12$$

Quel arbre obtient-on finalement ?

Exercice 10 *Dans quel ordre s'affichent les éléments pour chacun des trois parcours*

- *préfixé*
- *infixé*
- *postfixé*

L'**opération de suppression** d'un élément dans un arbre binaire de recherche est plus subtile. Supposant donc vouloir supprimer d'un arbre le nœud contenant une clé donnée, on commencera par la rechercher en retournant le sous-arbre dont la racine contient la clé cherchée plutôt que l'information associée. Si la recherche est fructueuse, on sera alors amené à distinguer trois cas selon que le nœud à supprimer a deux fils, un fils ou aucun fils.

Exercice 11 *Comment supprimer*

- *un nœud sans fils ?*
- *un nœud ayant un seul fils ?*
- *un nœud ayant deux fils (plus difficile) ?*

Quelle est la complexité de l'opération de suppression ?

Exercice 12 *Comment retirer la racine de l'arbre précédamment construit ? Quel arbre binaire de recherche obtient-on ?*

3 Dénombrement des arbres binaires

On souhaite dénombrer le nombre C_n d'arbres binaires de n noeuds (les entiers C_n sont appelés nombres de Catalan). Pour cela, on va utiliser une technique à base de séries formelles, sans chercher à la justifier totalement d'un point de vue mathématique.

Exercice 13 *Montrer que*
- $C_0 = 1$
- $C_1 = 1$
- $C_2 = 2$
- $C_3 = 5$

Que vaut C_4 ?

Exercice 14 *En considérant qu'un arbre de n éléments est formé d'une racine, d'un sous-arbre gauche de p éléments et d'un sous-arbre droit de q éléments (et que par conséquent $1+p+q = n$) trouver une relation de récurrence entre C_n et les C_i ($0 \leq i < n$).*

On pose $C(x) = \sum_{n\geq 0} C_n \times x^n$ (on ne cherchera surtout pas à essayer de justifier une quelconque convergence, cette écriture étant essentiellement formelle).

Exercice 15 *A partir de la relation de récurrence sur les C_n obtenue précédemment, montrer que l'on a*

$$C(x) = x \times C(x)^2 + 1$$

En déduire la valeur de $C(x)$ en fonction de x.

On admet le développement suivant, où l'on note $\left(\begin{array}{c} n \\ p \end{array} \right) = \frac{n!}{p!(n-p)!}$:

$$\sqrt{1-x} = 1 - \sum_{n=1}^{+\infty} \frac{2}{4^n(2n-1)} \left(\begin{array}{c} 2n-1 \\ n \end{array} \right) x^n$$

Exercice 16 *En développant $C(x)$ et en identifiant le coefficient de x^n à C_n, montrer que*

$$C_n = \frac{1}{2n+1} \left(\begin{array}{c} 2n+1 \\ n+1 \end{array} \right) = \frac{1}{n+1} \left(\begin{array}{c} 2n \\ n \end{array} \right)$$

Exercice 17 *On rappelle la formule de Stirling selon laquelle $n! \approx \frac{n^n}{e^n}\sqrt{2\pi n}$. En déduire que $C_n \approx \frac{1}{\sqrt{\pi}} 4^n n^{-\frac{3}{2}}$.*

Exercice 18 *Application : Étant donné un arbre binaire de n éléments, calculer la probabilité que l'un des sous-arbres droit ou gauche soit vide ?*

Exercice 19 *Application : Étant donné un arbre binaire de $2n+1$ éléments, calculer une valeur approchée de la probabilité que les deux sous-arbres droit et gauche aient chacun exactement n éléments.*

4 Pour aller plus loin : implémentation des arbres binaires de recherche

Exercice 20 *Compléter le squelette de programme suivant, implémentant divers algorithmes de gestion d'arbre binaire de recherche :*

```
#include <stdio.h>
#include <stdlib.h>

struct noeud
{
  int cle;
  struct noeud *gauche;
  struct noeud *droit;
};
typedef struct noeud *arbre;

// construit un arbre à partir de la valeur v de la clé de
// la racine et des sous-arbres droit (sad) et gauche (sag)
arbre branche(arbre sag, arbre sad, int v)
{ arbre nouv;
  nouv=(arbre) malloc(sizeof(struct noeud));
  nouv→cle=v;
  nouv→gauche=sag;
  nouv→droit=sad;
  return nouv;
}

// Impression infixée d'un arbre binaire
void imprime_infixe(arbre a){...}
// Recherche de v dans l'arbre binaire de recherche a
int recherche(arbre a, int v){...}
// Recherche du minimum dans un arbre
int minimum(arbre a){...}
// Recherche du maximum dans un arbre
int maximum(arbre a){...}
// Comptage du nombre d'éléments contenu dans l'arbre a
int nbnoeud(arbre a){...}
// Mesure de la hauteur de l'arbre a
int hauteur(arbre a){...}
// Insertion de v dans l'arbre a
void insertion(arbre *a, int v){...}
// Suppression de v dans l'arbre a
int supprime(arbre *a, int v){...}
// Libération complète de la mémoire occupée par un arbre
void free_arbre(arbre a){...}
```