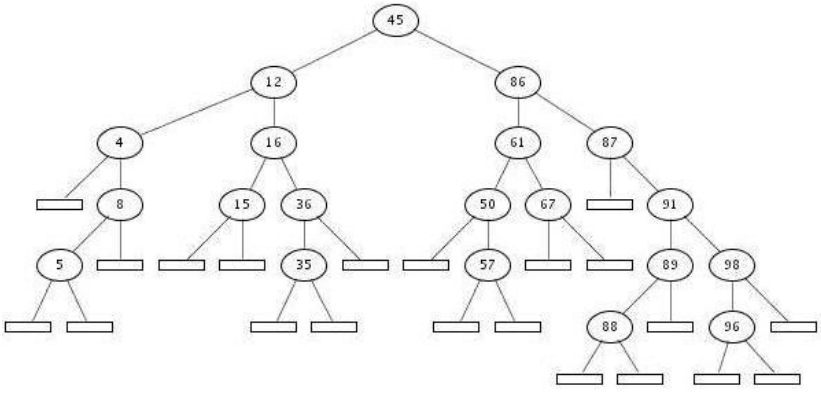
TD 7 : Algorithmique fonctionnelle
Arbres binaires.

Nga Nguyen - Stefan Bornhofen - Peio Loubière

1 Parcours d'arbre

Question 1 : Soit l'arbre suivant, dont les nœuds sont supposés contenir des clés entières :



- Donner la taille, la hauteur et les longueurs de cheminement interne et externe de cet arbre.
- Donner le parcours en largeur (gauche - droite).
- Donner les parcours en profondeur préfixe, infixe et postfixe.

Solution :

- taille = 20
  hauteur = 5
  lci = 30
  lce = 28
- (a) Parcours préfixe : 45, 12, 4, 8, 5, 16, 15, 36, 35, 86, 61, 50, 57, 67, 87, 91, 89, 88, 98, 96.
  (b) Parcours infixe : 4, 5, 8, 12, 15, 16, 35, 36, 45, 50, 57, 61, 67, 86, 87, 88, 89, 91, 96, 98.
  (c) Parcours postfixe : 5, 8, 4, 15, 35, 36, 16, 12, 57, 50, 67, 61, 88, 89, 96, 98, 91, 87, 86, 45.
- Parcours en largeur : 45, 12, 86, 4, 16, 61, 87, 8, 15, 36, 50, 67, 91, 5, 35, 57, 89, 98, 88, 96

2 Propiétés des arbres binaires

Question 2 : Prouver par récurrence que le nombre de feuilles F d'un arbre binaire non vide de taille N vérifie la relation : $1 \leq F \leq (N+1)/2$.

Solution :
Étude de la borne inférieure :
La propriété est vraie pour l'arbre réduit à une racine, et l'on peut garder le même nombre de feuilles, par exemple en ajoutant toujours les éléments du même côté. On obtient alors un arbre complètement dégénéré appelé chaîne. Supposons alors qu'on a construit un arbre binaire a_N de N éléments avec une seule feuille. Dans ces conditions, l'abre binaire a_{N+1}, construit en plaçant l'arbre vide à son fils gauche, a_N à son fils droit et un élément quelconque à sa racine, possède bien une taille de N + 1 et n'a toujours qu'une unique feuille.
Étude de la borne supérieure :
La propriété est encore vérifiée pour l'arbre binaire réduit à sa racine. Supposons maintenant qu'elle soit vraie pour tout arbre de taille t, inférieure à N. Elle est donc vraie pour les deux fils potentiels de l'arbre. Appelons respectivement N, Ng, Nd, F, Fg et Fd la taille de l'arbre, celle de son fils gauche, de son fils droit, le nombre de feuilles de l'arbre, celui de son fils gauche et de son fils droit. On peut donc écrire :

$$\begin{cases} N = Ng + Nd + 1 \\ F = Fg + Fd \end{cases}$$

La remarque précédente conduit à appliquer la relation de récurrence sur les deux fils :

$$\begin{cases} Fg \leq (Ng + 1)/2 \\ Fd \leq (Nd + 1)/2 \end{cases}$$

d'où l'on tire :

$$\begin{cases} F = Fg + Fd \\ \leq (Ng + 1 + Nd + 1)/2 \\ = ((Ng + Nd + 1) + 1)/2 \\ = (N + 1)/2 \end{cases}$$

ce qui prouve la propriété.

Question 3 : Soit A un arbre binaire complet de hauteur H. Quel est le nombre de feuilles de A ? Prouvez votre formule par récurrence.

Solution :
Soit nf(A) le nombre de feuilles de l'arbre A.
Pour H = 0, on a nf(A) = 1.
Pour H = 1, on obtient nf(A) = 2.
Pour H = 2, nf(A) = 4.
Essayons de prouver que, d'une façon générale, on a nf(A) = 2^h.
La proposition est vraie pour les valeurs 0, 1 et 2 de H. Nous allons la supposer vraie jusqu'à la valeur $H-1 > 1$ et montrer qu'elle est encore vraie pour H.

Si A est complet de hauteur H, alors filsGauche(a) est complet de hauteur H - 1. On A donc nf(filsGauche(A)) = 2^{h-1}. Le même raisonnement s'applique au sous-arbre droit de A.
Comme le nombre de feuilles d'un arbre (non réduit à sa racine) est la somme des feuilles de ses deux sous-arbres, il vient :
nf(A) = nf(filsGauche(A)) + nf(filsDroit(A))
nf(A) = $2^{h-1} + 2^{h-1}$
nf(A) = 2^h

3 Fonctions sur les arbres binaires

Question 4 : Écrire une fonction qui calcule le nombre de feuilles d'un arbre binaire.
Solution :

```
1  Fonction nbFeuilles(A : ArbreBinaire) : Entier
2  Début
3    Si estVide(A) Alors
4      retourner 0
5    SinonSi estVide(filsGauche(A)) et estVide(filsDroit(A))
6      retourner 1
7    Sinon
8      retourner nbFeuilles(filsDroit(A)) + nbFeuilles(
           filsGauche(A))
9    FinSi
10 Fin
```

Question 5 : Écrire une fonction qui teste si un élément donné appartient à un arbre binaire donné.

Solution :

```
1  Fonction appartient(e : T, A : ArbreBinaire) : Booléen
2  Début
3    Si estVide(A) Alors
4      retourner faux
5    SinonSi e = racine(A)
6      retourner vrai
7    Sinon
8      retourner appartient(e, filsDroit(A)) ou appartient(e
           , filsGauche(A))
9    FinSi
10 Fin
```

Question 6 : Écrire une fonction qui calcule le nombre d'occurrences d'un certain élément dans un arbre binaire.
Solution :

```
1  Fonction nbOccurences(e : T, A : ArbreBinaire) : Entier
2  Début
3    Si estVide(A) Alors
4      retourner 0
5    SinonSi e = racine(A)
6      retourner 1 + nbOccurences(e, filsDroit(A)) +
           nbOccurences(e, filsGauche(A))
7    Sinon
8      retourner nbOccurences(e, filsDroit(A)) +
           nbOccurences(e, filsGauche(A))
9    FinSi
10 Fin
```

Question 7 : Écrire une fonction qui calcule la longueur de cheminement d'un arbre binaire non vide.

Solution :

```
1  Fonction lc(A : ArbreBinaire) : Entier
2  Début
3    Si !estVide(A) Alors
4      Si estVide(filsGauche(A)) et estVide(filsDroit(A))
           Alors
5        retourner 0
6      SinonSi estVide(filsGauche(A))
7                retourner lc(filsDroit(A)) + taille(
                     filsDroit(A))
8      SinonSi estVide(filsDroit(A))
9                retourner lc(filsGauche(A)) + taille(
                     filsGauche(A))
10     Sinon
11       retourner lc(filsDroit(A)) + taille(filsDroit(A))
             + lc(filsGauche(A)) + taille(filsGauche(A))
12     FinSi
13   FinSi
14 Fin
```

Question 8 : Écrire une fonction qui reçoit un arbre binaire et renvoie un arbre "miroir" où les sous-arbres gauche et droit de chaque noeud sont échangés.
Solution :

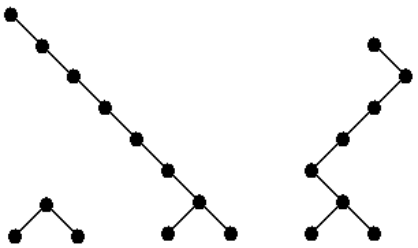
```
1  Fonction torsion(A : ArbreBinaire) : ArbreBinaire
2  Début
3    Si estVide(A) Alors
4      retourner arbreVide
5    Sinon
6      retourner cons(racine(A), torsion(filsDroit(A)),
           torsion(filsGauche(A)))
7    FinSi
8  Fin
```

Question 9 : Écrire une fonction qui renvoie la moyenne du nombre de fils par sommet. La fonction renvoie 0 pour l'arbre vide.

Solution :

```
1  Fonction nbFils(A : ArbreBinaire) : Entier
2  Début
3    Si estVide(A) ou Alors
4       retourner 0
5    SinonSi estVide(filsGauche(A)) et estVide(filsDroit(A))
6       retourner 0
7    SinonSi estVide(filsGauche(A))
8       retourner 1 + nbFils(filsDroit(A))
9    SinonSi estVide(filsDroit(A))
10      retourner 1 + nbFils(filsGauche(A))
11   Sinon
12      retourner 2 + nbFils(filsGauche(A)) + nbFils(
            filsDroit(A))
13   FinSi
14 Fin
15
16 Fonction nbFilsMoy(A : ArbreBinaire) : Réel
17 Début
18   Si estVide(A)
19     retourner 0
20   Sinon
21     retourner nbFils(A)/taille(A)
22   FinSi
23 Fin
```

Question 10 : Écrire une fonction qui vérifie si l'arbre binaire est "fourchu", c'est-à-dire qu'il est filiforme (un seul fils par sommet) sauf au dernier niveau, par exemple



Solution :

```
1  Fonction fourchu(A : ArbreBinaire) : Booleén
2  Début
3    Si hauteur(A) ≤ 0 Alors
4      retourner faux
5    SinonSi hauteur(A)=1
6      retourner not (estVide(filsGauche(A))) et not (
           estVide(filsDroit(A)))
7    SinonSi estVide(filsDroit(A))
8      retourner fourchu(filsGauche(A))
9    SinonSi estVide(filsGauche(A))
10     retourner fourchu(filsDroit(A))
11   Sinon
12     retourner faux
13   FinSi
14 Fin
```

Question 11 : Écrire une fonction qui teste si un arbre binaire est localement complet.

Solution :

```
1  Fonction localementComplet(A : ArbreBinaire) : Booléen
2  Début
3    Si estVide(A) Alors
4      retourner vrai
5    SinonSi estVide(filsGauche(A)) et estVide(filsDroit(A))
6      retourner vrai
7    SinonSi estVide(filsGauche(A)) ou estVide(filsDroit(A))
8      retourner faux
9    Sinon
10     retourner localementComplet(filsGauche(A)) et
           localementComplet(filsDroit(A))
11   FinSi
12 Fin
```

Question 12 : Écrire une fonction qui teste si un arbre binaire est complet.

Solution :

```
1  Fonction complet(A : ArbreBinaire) : Booléen
2  Début
3    Si estVide(A) Alors
4      retourner vrai
5    Sinon
6      retourner (hauteur(filsGauche(A)) = hauteur(
           filsDroit(A))) et complet(filsGauche(A)) et
           complet(filsDroit(A))
7    FinSi
8  Fin
```

Question 13 : Écrire une fonction qui teste si un arbre binaire est parfait.
Solution :

```
1  Fonction parfait(A : ArbreBinaire) : Booléen
2  Début
3    Si estVide(A) Alors
4      retourner vrai
5    SinonSi hauteur(filsGauche(A)) = hauteur(filsDroit(A)))
6      retourner complet(filsGauche(A)) et parfait(
           filsDroit(A))
7    SinonSi hauteur(filsGauche(A)) = hauteur(filsDroit(A)))
         + 1
8      retourner parfait(filsGauche(A)) et complet(
           filsDroit(A))
9    Sinon
10     retourner faux
11   FinSi
12 Fin
```