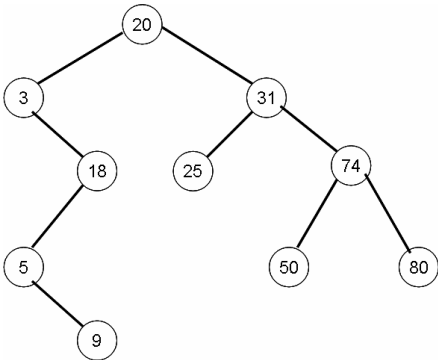
Algorithmique et programmation procédurale

TD No 6 - CORRIGE

Parcours et rotations des arbres binaires

Exercice 1 : Soit l'arbre :



a) Donner les valeurs suivantes :

- sa taille ;
- sa hauteur ;
- sa longueur de cheminement ;
- ses parcours infixe, postfixe et prefixe.

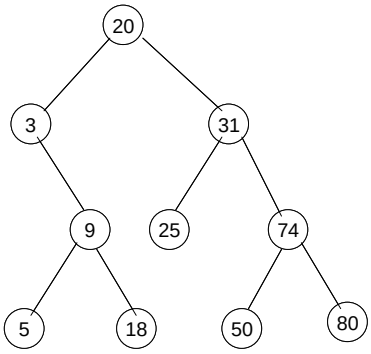
b) Faire des rotations nécessaires pour le rendre équilibre. Expliquer chaque étape de rotation.

Corrigé :

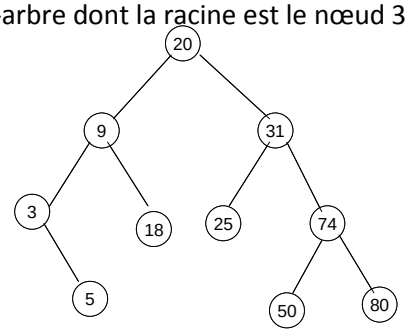
- sa taille = 10
- sa hauteur = 4
- sa longueur de cheminement = 21
- parcours infixe : 3, 5, 9, 18, 20, 25, 31, 50, 74, 80
- parcours postfixe : 9, 5, 18, 3, 25, 50, 80, 74, 31, 20
- parcours prefixe : 20, 3, 18, 5, 9, 31, 25, 74, 50, 80

· Etape 1 : Rotation gauche-droite sur le sous-arbre dont la racine est le nœud 18

.



Etape 2 : Rotation gauche sur le sous-arbre dont la racine est le nœud 3



Exercice 2 : On considère un arbre binaire de recherche, contenant des nombres entiers et construit par adjonctions successives aux feuilles, avec la suite :

347, 621, 219, 382, 924, 330, 2, 266, 911, 401, 344, 398, 252, 898, 220, 244, 258, 363,397, 362.

a) Construire l'arbre avec la suite complète. Est-il équilibré ?
b) Construire l'arbre pas à pas et calculer à chaque instant la fonction de déséquilibre. A chaque occurrence d'un déséquilibre, utiliser les rotations vues en cours pour restaurer l'équilibre : combien de rotation sont nécessaires ? De quel type sont-elles ?

Exercice 3 :

a) Écrire en pseudo-code l'algorithme itératif qui permet de parcourir en profondeur un arbre binaire en utilisant une pile (aide : inspirez vous du parcours en largeur vu en cours).
b) Écrire une fonction qui reçoit un tableau et renvoie l'arbre parfait pour lequel la suite des valeurs de ce tableau correspond au parcours en largeur.

Corrigé

a)
```
Fonction parcoursProfondeur(A : ArbreBinaire)
Variables p : Pile, ab : ArbreBinaire
Début
        Si  (!estVide(A)) Alors
                p <- pileVide()
                empiler(p, A)
                Tantque (! estVide(p))
                        ab <- depiler(p)
                        traiter(racine(ab))
                        Si  (!estVide(fD(ab))) Alors
                                empiler(p, fD(ab))
                        Finsi
                        Si  (!estVide(fG(ab))) Alors
                                empiler(p,fG(ab))
                        Finsi
                FinTantque
        Finsi
Fin
```

b) Fonction creer (tab : Tableau de T, n : Entier, i : Entier) : ArbreBinaire de T

```
Debut
  Si i <= n Alors
        retourner  cons (tab[i], creer (tab, n, 2*i), creer (tab, n, 2*i + 1))
  Sinon
        retourner arbreVide
  FinSi
Fin
```

```
Fonction Tableau2ArbreParfait(tab : Tableau de T, n : Entier) : ArbreBinaire de T
Début
  Retourner creer (tab, n, 1)
Fin
```

Exercice 4 : Écrire en pseudo-code les fonctions de rotation suivantes :

- rotGauche
- rotDroite
- rotGaucheDroite
- rotDroiteGauche

N'oubliez pas les préconditions nécessaires !

Corrigé

a) Fonction rotGauche(A : ABR) : ABR

```
Début
        Si  (!estVide(A) et !estVide(fD(A))) Alors
                return  cons(   racine(fD(A)),
                                cons(racine(A), fG(A), fG(fD(A))),
                                fD(fD(A)))
        Sinon
                Erreur
        Finsi
Fin
```

b) Fonction rotDroite(A : ABR) : ABR

```
Début
        Si  (!estVide(A) et !estVide(fG(A))) Alors
                return  cons(   racine(fG(A)),
                                fG(fG(A)),
                                cons(racine(A), fD(fG(A)), fD(A)))
        Sinon
                Erreur
        Finsi
Fin
```

c) Fonction rotGaucheDroite(A : ABR) : ABR

```
Début
        Si  (!estVide(A) et !estVide(fG(A)) et !estVide(fD(fG(A)))) Alors
```

```
            return  cons(   racine(fD(fG(A))),
                            cons(racine(fG(A)), fG(fG(A)), fG(fD(fG(A)))),
                            cons(racine(A), fD(fD(fG(A))), fD(A)))
        Sinon
            Erreur
        Finsi
    Fin
```

d) **Fonction** rotDroiteGauche (A : ABR) : ABR

```
    Début
        Si  (!estVide(A) et !estVide(fD(A)) et !estVide(fG(fD(A)))) Alors
            return  cons(   racine(fG(fD(A))),
                            cons(racine(A), fG(A), fG(fG(fD(A)))),
                            cons(racine(fD(A)), fD(fG(fD(A))),fD(fD(A))))

        Sinon
            Erreur
        Finsi
    Fin
```