Semaine 14
Arbres - 2/2

1 Arbres binaires de recherche (ABR) - Compléments

La structure d'ABR présente entre autre l'avantage de permettre une recherche très efficace : on peut descendre directement de la racine vers le sommet que l'on recherche en choisissant de descendre par la gauche ou par la droite selon le résultat de la comparaison entre la valeur cherchée et la valeur du sommet visité. Ainsi, il n'est pas nécessaire de parcourir la totalité des sommets, et le nombre de comparaisons est au plus égal à la profondeur de l'ABR + 1.

Remarque : l'élément minimum se trouve au bout de la branche gauche et l'élément maximum au bout de la branche droite.

Exercice 41 : Recherche dans un ABR
Écrire une fonction qui recherche un élément dans un ABR, et qui retourne l'adresse de cet élément s'il est présent dans l'ABR, NULL sinon.
Réponse :

```
Fonction rechercheABR( E A : TArbBin ; Elém : TInfo ) : TAdresse
// retourne l'adresse de Elém dans A, NULL si Elém n'est pas dans A
var Adr : TAdresse
    Val : TInfo
    Trouvé : booléen
Début
    Adr <-- adresseRacine( A )
    Trouvé <-- Faux
    Tant Que Adr <> NULL et non Trouvé
    Faire Début
            Val <-- valeurSommet( A, Adr )
            Si Val = Elém
            Alors Trouvé <-- Vrai
            Sinon Si Elém < Val
                  Alors Adr <-- adresseFilsGauche( A, Adr )
                  Sinon Adr <-- adresseFilsDroit( A, Adr )
          Fin
    Retourner( Adr )
Fin
```

Remarque : en terme de complexité, si n est le nombre de sommets de l'ABR, la recherche est en $O(logn)$ s'il est bien équilibré, et en $O(n)$ dans le cas le pire (un chemin "rectiligne" gauche ou droite).

2 Parcours en largeur : applications

A partir de maintenant, on ne considère plus de ABR, mais des arbres binaires généraux.

Exercice 42 : Afficher les éléments d'un même niveau
On considère un arbre généalogique (binaire), et on souhaite afficher à l'écran les noms des personnes d'une même génération donnée.
Réponse : il faut enfiler en plus de l'adresse de l'élément son niveau dans l'arbre pour savoir où on en est. On arrête le parcours de l'arbre dès qu'on a parcouru le niveau voulu.

```
Type Sommet = entite( adr : TAdresse ; lvl : entier )

Action memeGeneration(E A : TArbBin, k : entier)
var s, sf : Sommet
    F : File de Sommet
Début
    s.adr <-- adresseRacine( A )
    s.lvl <-- 0
    Si s.adr <> NULL    // pas utile ici car arbre non vide
    Alors Début
            créerFile( F )
            enfiler( F, s )
            Tant Que Non FileVide( F )
            Faire Début
                    s <-- valeurPremier( F )
                    défiler( F )
                    // Traitement
                    Si s.lvl = k
                    Alors ecrire( valeurSommet( A, s.adr ) )
                    Sinon Debut  // on n'enfile plus si on arrive au niv k
                            Si adresseFilsGauche( A, s.adr ) <> NULL
                            Alors Debut
                                    sf.lvl <-- s.lvl + 1
                                    sf.adr <-- adresseFilsGauche( A, s.adr )
                                    enfiler( F, sf )
                                  Fin
                            Si adresseFilsDroit( A, s.adr ) <> NULL
                            Alors Debut
                                    sf.lvl <-- s.lvl + 1
                                    sf.adr <-- adresseFilsDroit( A, s.adr )
                                    enfiler( F, sf )
                                  Fin
                          Fin
                  Fin
          Fin
Fin
```

3 Parcours en profondeur (version itérative)

Les méthodes classiques de parcours en profondeur non récursif utilisent une pile. Pour le préfixe, on peut écrire une version simple (idem largeur avec file $\leftrightarrow$ pile, et droit $\leftrightarrow$ gauche pour un parcours à main gauche). Mais si on veut une version générale pour les trois (préfixe, infixe, postfixe), il faut travailler un peu plus.

Version préfixe (simple) :

```
Action ParcoursProfondeurPrefixe(E A : TArbBin)
var Adr : TAdresse
    P : Pile de TAdresse
Début
    Adr <-- adresseRacine( A )
    Si Adr <> NULL    // pas utile ici car arbre non vide
    Alors Début
            créerPile( P )
            empiler( P, Adr )
            Tant Que Non pileVide( P )
            Faire Début
                    Adr <-- valeurSommet( P )
                    dépiler( P )
                    Traiter( A, Adr )
                    // droit puis gauche pour un parcours à main gauche
                    Si adresseFilsDroit( A, Adr ) <> NULL
                    Alors empiler( P, adresseFilsDroit( A, Adr ) )
                    Si adresseFilsGauche( A, Adr ) <> NULL
                    Alors empiler( P, adresseFilsGauche( A, Adr ) )
                  Fin
          Fin
Fin
```

Version générale (MM) : Il est facile de descendre en privilégiant la gauche, mais lorsqu'on est bloqué les primitives ne nous permettent pas de remonter ; d'autre part, quand on remonte sur un sommet, il faut parfois redescendre à droite (2° passage), parfois continuer la remontée (3° passage).

```
Type TDirection = (gauche, droit)
Type TSituation = entité ( adresse : TAdresse
                           direction : TDirection )

Action Parcours_iter( E A : TArbBin, Adr : TAdresse )
var P : Pile de TSituation
    situation : TSituation
Début
    créerPile( P )
    situation.direction <-- gauche
    situation.adresse <-- Adr
```

```
    Répéter
      Si situation.direction = gauche
      Alors Début
              Tant Que situation.adresse <> NULL
              Faire Début
                      Traitement1
                      empiler( P, situation )
                      situation.adresse <-- adresseFilsGauche( A, situation.adresse )
                    Fin
        Fin
      Si non pileVide(P)
      Alors Début
              situation <-- valeurSommet( P )
              dépiler( P )
              Si situation.direction = gauche
              Alors Début
                      Traitement2
                      situation.direction <-- droite
                      empiler( P, situation )
                      situation.direction <-- gauche
                      situation.adresse <-- adresseFilsDroit( A, situation.adresse )
                    Fin
              Sinon Traitement3
            Fin
    Jusqu'à pileVide( P )
Fin
```