5-Arbre binaire de recherche

- Arbre binaire de recherche
- Modification d'un arbre binaire de recherche
- Equilibrage

5.1- Arbre Binaire de Recherche

Définition 5.1 : Dans un arbre binaire de recherche, quel que soit x un sommet interne d'étiquette val(x), soit LG(x) (resp. LD(x)), l'ensemble des étiquettes du sous arbre gauche (resp. droit) de x. On a :

$$\forall y \in LG(x), \forall z \in LD(x), y \leq val(x) < z$$

Propriété 5.1: Si on parcourt un arbre binaire de recherche en ordre infixe, on obtient une séquence d'étiquettes triées en ordre croissant.

Corollaire 5.1: Si n est le nombre de sommets d'un arbre binaire de recherche, on obtient la liste triée en O(n).

On utilise les primitives des arbres binaires.

5.1-Recherche d'un élément dans un ABR

```
fonction recherche(val x:sommet, val e:objet):sommet;
   var tmp:objet;
   debut
      si x==NIL alors
        retourner(NIL)
      sinon
        tmp= getValeur(x);
        si tmp==e alors
          retourner(x);
        sinon
          si e <=tmp alors
            retourner(recherche(filsGauche(x),e));
          sinon
            retourner(recherche(filsDroit(x),e));
          finsi
        finsi
      finsi
   fin
```

Complexité
minimum :
maximum :

5.1-Recherche d'un élément dans un ABR

```
fonction recherche(val x:sommet, val e:objet):sommet;
   var tmp:objet;
   debut
      si x==NIL alors
        retourner(NIL)
      sinon
        tmp= getValeur(x);
        si tmp==e alors
          retourner(x);
        sinon
          si e <=tmp alors
            retourner(recherche(filsGauche(x),e));
          sinon
            retourner(recherche(filsDroit(x),e));
          finsi
        finsi
      finsi
   fin
```

Complexité
minimum :
maximum :

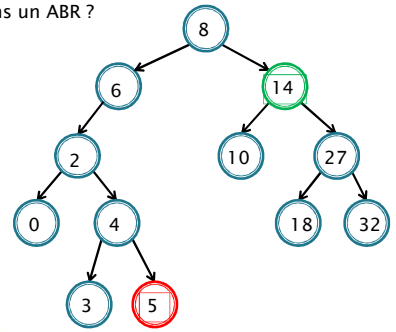
5.1- Recherche du plus petit élément d'un ABR

```
fonction chercheMinimum(val x:sommet):sommet;
  debut
    tantque filsGauche(x)!=NIL faire
      x=filsGauche(x);
    fintantque
    retourner(x);
  fin
```

Complexité
minimum :
maximum :

5.1- Recherche de l'élément suivant une valeur présente dans un ABR

Dans l'ordre croissant, où se trouve le suivant d'un élément x dans un ABR ?



5.1- Recherche de l'élément suivant une valeur présente dans un ABR

```
fonction chercheSuivant(val x: sommet,
      val e : objet):sommet;
  var p:sommet;
  debut
    x=cherche(x,e);
    si x==NIL alors
      retourner(NIL);
    sinon
      si filsDroit(x)!=NIL alors

return(cherchePlusPetit(filsDroit(x))

sinon
        p=pere(x);
        tantque p!=NIL faire
 si filsGauche(p)==x alors
            retourner(p)
          sinon
            x=p;
            p=pere(p);
          finsi
        fintantque
        retourner(NIL);
      finsi
    finsi
  fin
```

Complexité
minimum :
maximum :

5.2- Modification d'un ABR

Les primitives ajouter et supprimer des objets permettent de faire évoluer un ABR.

```
fonction ajouter(ref x:sommet,
            val e:objet):vide;
  var s:sommet;
  debut
     si e <= valeurSommet(x) alors
       s=filsGauche(x);
     si s==NIL alors
       ajouterFilsGauche(x,e);
     sinon
        ajouter(s,e);
     finsi

sinon
    s=filsDroit(x);
      si s==NIL alors
         ajouterFilsDroit(x,e);
      sinon
        ajouter(s,e);
     finsi
  finsi
fin
```

Complexité
minimum :
maximum :

5.2- Modification d'un ABR

```
fonction supprimer(ref x:sommet):vide;
 /*Tous les elts sont différents */
  var p,f,y:sommet;
  debut
    si estFeuille(x) alors
      p=pere(x);
       /*traiter le cas racine*/
      si filsGauche(p)==x alors
        supprimerFilsGauche(p)
      sinon
        supprimerFilsDroit(p)
      finsi
```

Complexité
minimum :
maximum :

```
sinon
  f=filsDroit(x);
  si f!=NIL
    y=cherchePlusPetit(f);
  sinon
    f=filsGauche(x);
    y=cherchePlusGrand(f);
  finsi
var v : valElement;
  v=getValeur(y);
    supprimer(y);
    setValeur(x,v);
finsi
fin
```

5.2- Modification d'un ABR

Dans le cas général la fonction supprimer applique l'algorithme suivant :

Si le sommet à supprimer :

- est **une feuille** on l'enlève
- **a 1 fils** on le remplace par son fils
- **a 2 fils** on remplace sa valeur par la valeur précédente dans l'ordre croissant et on supprime le sommet qui portait cette valeur dans l'ABR.

5.3 : Equilibrage d'un ABR

La complexité des opérations sur un ABR dépendant de la hauteur de l'arbre, il est important qu'un ABR reste aussi proche que possible d'un arbre binaire parfait de manière à ce que la hauteur soit minimum.

L'équilibrage d'un ABR peut-être obtenu par un algorithme de type "diviser pour régner".

On récupère la liste des éléments triés dans un tableau T[1..N] où N est la taille de l'arbre de départ et on reconstruit l'arbre.

5.3 : Equilibrage d'un ABR

```
fonction lister(val x:sommet, ref T:tableau[1..N] d'objet,
                ref i:entier):vide
  debut
    si estFeuille(x)alors
       i=i+1;
       T[i]= getValeur(x);
    sinon
      si filsGauche(x)!=NIL alors
          lister(filsGauche(x),T,i);
      finsi
        i=i+1;
        T[i]= getValeur(x);
      si filsDroit(x)!=NIL alors
        lister(filsDroit(x),T,i);
      finsi
    finsi

fin
```

5.3 : Equilibrage d'un ABR

L'appel : `liste(A,T,0)` fournit, en utilisant l'ordre infixe, dans le tableau T la liste des valeurs dans l'ordre croissant de l'arbre A.

```
fonction detruireArbreBinaire(ref A:arbreBinaire
d'objet):vide;
  debut
    si s!=NIL alors
      detruireArbreBinaire(filsGauche(s));
      supprimerFilsGauche(s);
      detruireArbreBinaire(filsDroit(s));
      supprimerFilsDroit(s);
    finsi
  fin
```

5.3 : Equilibrage d'un ABR

```
fonction equilibre(ref A:arbreBinaire de objet):vide;
  var N:entier;
  var T:tableau[1..N]d'objet;

  debut
    N=tailleArbre(A);
    lister(A,T,0);
    detruireArbreBinaire(A);
    delete(A);
    A=construire(T,1,N))
  fin
```

5.3 : Equilibrage d'un ABR

```
fonction construire(ref T:tableau[1..N]d'objet,ref
                                        d,f:entier):sommet;
 var m:entier;
 var c,s:sommet;
  debut
    si d≤f alors
      m=(d+f)//2;
      new(c);
      setValeur(c,T[m]);
      si d==f alors
        c^.gauche=NIL;
        c^.droit=NIL;
        retourner(c);
      sinon
        s=construire(d,m-1);
        c^.gauche=s;
        si s!=NIL alors
          s^.pere=c;
        finsi
        s=construire(m+1,f);
        c^.droit=s
        si s!=NIL alors
           s^.pere=c;
        finsi
      finsi
      retourner(c);
   sinon
     retourner(NIL)
   finsi
fin
```