

Université Bordeaux 1

Licence Semestre 3 - Algorithmes et structures de données 1

Dernière mise à jour effectuée le 1 Septembre 2013

Arbre binaire de recherche

- [Arbre binaire de recherche](#)
 - [Modification d'un arbre binaire de recherche](#)
 - [Equilibrage](#)
-

1. Arbre binaire de recherche

Définition 8.1. Soit x un sommet interne d'étiquette $l(x)$, $L_G(x)$ (resp. $L_D(x)$), l'ensemble des étiquettes du sous arbre gauche (resp. droit) de x . On a

$$\forall y \in L_G(x), \forall z \in L_D(x), y \leq l(x) < z.$$

Propriété 8.2 Si on parcourt un arbre binaire de recherche en ordre infixe, on obtient une séquence d'étiquettes triées en ordre croissant.

Corollaire 8.3 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.

Recherche d'un élément dans un sous arbre

```
fonction recherche(x:sommet, val e:objet):sommet;
  var tmp:objet;
  début
    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 : $O(1)$
- maximum : $O(h(x))$

Recherche du plus petit élément d'un sous arbre

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

Complexité

- o minimum : $O(1)$
- o maximum : $O(h(x))$

Recherche de l'élément suivant une valeur présente dans un sous-arbre

```
fonction rechercheSuivant(val x: sommet, val e:objet):sommet;
var p:sommet;
début
  x=cherche(x,e);
  si x==NIL alors
    retourner(NIL);
  sinon
    si filsDroit(x)!=NIL alors
      retourner(recherchePlusPetit(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
  fin
```

Complexité

- o minimum : $O(1)$
 - o maximum : $O(h(x))$
-

2. Modification d'un arbre binaire de recherche

Un ABR est un conteneur. 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;
début
  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);
    finssi
finsi
fin

```

Complexité

- o minimum : $O(1)$
- o maximum : $O(h(x))$

```

fonction supprimer(ref x:sommet):booléen;
var p,f,y:sommet;
début
    si estFeuille(x) alors
        p=pere(x);
        si filsGauche(p)==x alors
            supprimerFilsGauche(p)
        sinon
            supprimerFilsDroit(p)
        finssi
    sinon
        f=filsDroit(x);
        si f!=NIL
            y=cherchePlusPetit(f);
        sinon
            f=filsGauche(x);
            y=cherchePlusGrand(f);
        finssi
        v=getValeur(y);
        supprimer(y);
        setValeur(x,v);
    finssi
fin

```

Complexité

- o minimum : $O(1)$
- o maximum : $O(h(x))$

3. Equilibrage

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.

```

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

```

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;  
  début  
    si s!=NIL alors  
      detruireArbreBinaire(filsGauche(s));  
      supprimerFilsGauche(s);  
      detruireArbreBinaire(filsDroit(s));  
      supprimerFilsDroit(s);  
    fin si  
  fin
```

```
fonction equilibre(ref A:arbreBinaire de objet):vide;  
  var N:entier;  
  début  
    N=tailleArbre(A);  
    var T:tableau[1..N]d'objet;  
    lister(A,T,0);  
    detruireArbreBinaire(A);  
    delete(A);  
    A=construire(T,1,N)  
  fin
```

```
fonction construire(ref T:tableau[1..N]d'objet,  
                  ref d,f:entier):sommet;  
  
  var m:entier;  
  var c,s:sommet;  
  début  
    si d≤f alors  
      m=partieEntiere((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;  
        fin si  
        s=construire(m+1,f);  
        c^.droit=s  
        si s!=NIL  
          s^.pere=c;  
        fin si  
      fin si  
      retourner(c);  
    sinon  
      retourner(NIL)  
    fin si  
  fin
```