CPI1 2009-2010

Algorithmique fonctionnelle

Nga Nguyen – Stefan Bornhofen – Peio Loubière

TD8: Arbres binaires de recherche

1 Insertion et suppression

1. Insérer successivement dans un arbre binaire de recherche (ABR) vide: 4,7,2,6,12,3,5,1.

2. Y supprimer successivement: 12, 4, 5, 7, 3.

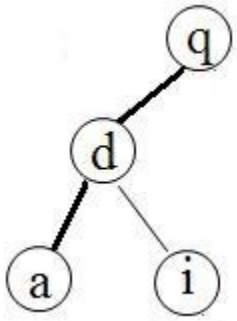
2 Réflexions

1. Quel est le critère d'une série d'insertions dans un arbre vide pour que l'ABR résultant soit filiforme?

Corrigé: Soit s1, s2, …. sN la série d'insertions d'un ABR à N éléments. L'ABR sera filiforme si tous les élements suivant si (i=1...N) sont soit plus grands, soit plus petits que si.

2. Supposons une branche d'ABR qui va de la racine jusqu'à une feuille donnée. On note respectivement A, B et C l'ensemble des valeurs situées à gauche, le long et à droite de cette branche. Si $a \in A, b \in B, c \in C$, est-ce que nous avons nécessairement a<b<c?

Corrigé: Non.
Contre-exemple: $\mathrm{q} \in B, \mathrm{i} \in C, mais \mathrm{q} > \mathrm{i}$



3. Montrer (par récurrence) que la connaissance du parcours préfixe d'un ABR suffit pour reconstituer l'arbre initial. Est-ce vrai pour le parcours postfixe?

Corrigé:
Prouvons cette équivalence par récurrence.
N=0: La propriété est naturellement vraie pour l'arbre vide.
N>0: Supposons qu'elle soit vraie pour tout ABR de taille inférieure à N. Soit a un arbre de taille N. La donnée du parcours préfixe de a donne, dans l'ordre, sa racine, les éléments de son fils gauche puis ceux de son fils droit. On sait donc où commence le parcours préfixe du fils gauche (juste après la racine). Pour savoir où il se termine, il suffit de repérer le premier élément listé dont la clé est supérieure à celle de la racine :

$$\underbrace{e_1}_{rac}, \underbrace{e_2, \ldots, e_i}_{fg}, \underbrace{e_{i+1}, \ldots, e_N}_{fd}$$

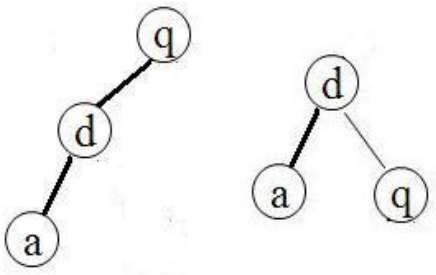
avec $e_i < e_1$ et $e_{i+1} > e_1$.

La donnée du parcours e2...ei du fils gauche permet par hypothèse (le fils gauche et le fils droit ont une taille inférieure N) de reconstituer ce sous-arbre, de même que la donnée du parcours ei+1...eN permet de reconstituer le fils droit, d'où la preuve de la propriété.

La preuve de l'équivalence dans le cas du parcours postfixe est rigoureusement identique, il suffit de remarquer que le fils droit commence dès qu'un élément parcouru possède une clé supérieure à celle du dernier élément parcouru (c'est-à-dire la racine).

4. Montrer que la connaissance de la donnée du parcours infixe ne suffit pas pour reconstituer l'arbre initial.

Corrigé: Contre-exemple



5. Les séquences suivantes peuvent-elles correspondre à une suite d'éléments dans un ABR visités lors de la recherche de la valeur 2048 ?

(a) 100, 140, 4096, 150, 3000, 2048	**Corrigé:**	ok
(b) 100, 150, 4096, 140, 3000 ,2048		X
(c) 4096, 100, 140, 150, 3000 ,2048		ok
(d) 100, 150, 140, 4096, 3000 ,2048		X
(e) 3000, 4096, 140, 150, 100 ,2048		X

3 Définir en pseudo-code les fonctions

1. **estABR** qui teste si un arbre binaire donné est un arbre binaire de recherche. *Attention*: l'information comme quoi le fils gauche est inférieur au noeud et le fils droit est supérieur au noeud est insuffisante (cherchez un contre-exemple)!

```
Fonction estABR(A: ArbreBinaire): Booléen
Début
        Si hauteur(A)<=0 Alors
                Retourner vrai
        SinonSi estVide(fG(A))
                Retourner estABR(fD(A)) et racine(A)<min(fD(A))
        SinonSi estVide(fD(A))
                Retourner estABR(fG(A)) et racine(A)>max(fG(A))
        Sinon
                Retourner estABR(fD(A)) et racine(A)<min(fD(A)) et
                        estABR(fG(A)) et racine(A)>max(fG(A))
        FinSi
Fin
```

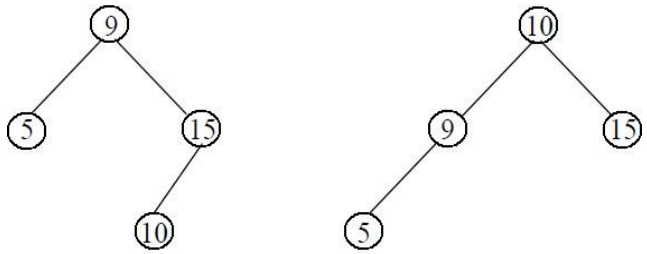
2. **rechercher** qui recherche un élément donné dans un ABR et renvoie l'ABR dont la racine contient cet élément;

```
Fonction rechercher (x: T, A: ABR):  ABR
Début
        Si estVide(A) Alors
                Retourner A
        SinonSi x=racine(A)
                Retourner A
        SinonSi x<racine(A)
                Retourner rechercher(x,fG(A))
        Sinon
                Retourner rechercher(x,fD(A))
        FinSi
Fin
```

3. **rechercheValide** qui reçoit une liste d'éléments et détermine si cette liste est une recherche valide dans un ABR (voir l'exercice 2.5);

```
Fonction rechercheValide (L: Liste):  Booléen
Début
        Si estVide(L) Alors
          Retourner faux
        SinonSi longueur(L)=1
          Retourner vrai
        SinonSi premier(L)>premier(reste(L))
          Retourner premier(L)>max(reste(L)) et rechercheValide(reste(L))
        Sinon { premier(L)<premier(reste(L)) }
          Retourner premier(L)<min(reste(L)) et rechercheValide(reste(L))
        FinSi
Fin
```

4. **équivalent** qui teste l'équivalence de deux ABR. Deux ABR sont dits équivalents lorsqu'ils comportent les mêmes valeurs, par exemple:




```
// tester si A1 contient les éléments de A2
Fonction contient (A1, A2: ABR):  Booléen
Début
        Si estVide(A2) Alors
          Retourner vrai
        SinonSi appartient(racine(A2), A1)
          Retourner contient(A1,fG(A2)) et contient(A1,fD(A2))
        Sinon
          Retourner faux
        FinSi
Fin

Fonction équivalent (A1, A2: ABR):  Booléen
Début
        Retourner contient(A1,A2) et contient (A2,A1)
Fin
```

4 Insertion à la racine

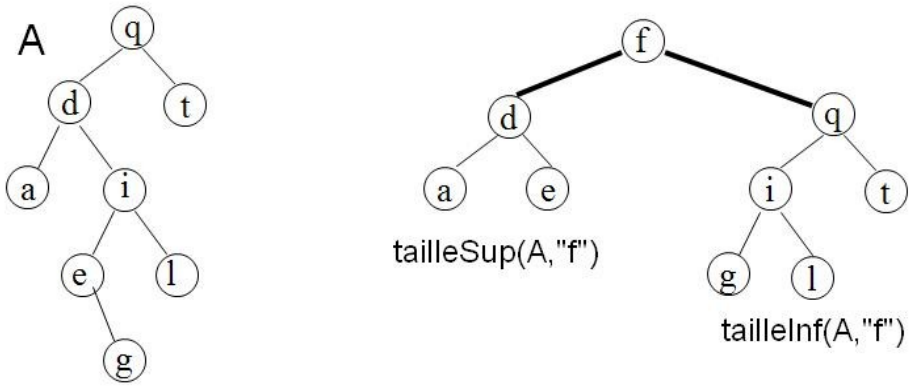
Dans cette méthode d'insertion, on ajoute un nouvel élément non pas en tant que feuille, mais à la racine de l'arbre. Ceci permet de retrouver plus rapidement les éléments récemment insérés. Le principe consiste à construire un nouvel arbre dont

- la racine est x
- le sous-arbre gauche est une copie de l'arbre A où l'on a taillé tous les éléments > x
- le sous-arbre droit est une copie de l'arbre A où l'on a taillé tous les éléments < x

Ecrire les fonctions

- **taillerSup (A,x)** qui renvoie une copie de l'arbre A où l'on a taillé tous les éléments > x
- **taillerInf (A,x)** qui renvoie une copie de l'arbre A où l'on a taillé tous les éléments < x
- **insérerRacine(A,x)** qui compose les deux arbres précédents et l'élément x à insérer.

Exemple: Insertion de « f » à la racine de l'arbre A.




```
Fonction taillerSup(x: T, A: ABR):  ABR
Début
        Si estVide(A) Alors
                Retourner A
        SinonSi  x > racine(A)
                Retourner cons(racine(A),fG(A), taillerSup(x, fD(A)))
        Sinon { x < racine(A) }
                Retourner taillerSup(x, fG(A))
        FinSi
Fin

Fonction taillerInf(x: T, A: ABR):  ABR
Début
        Si estVide(A) Alors
                Retourner A
        SinonSi  x < racine(A)
                Retourner cons(racine(A), taillerInf(x, fG(A)), fD(A))
        Sinon { x > racine(A)     }
                Retourner taillerInf(x, fD(A))
        FinSi
Fin

Fonction insérerRacine(x: T, A: ABR):  ABR
Début
        Retourner cons(x, taillerSup(x, A), taillerInf(x, A))

Fin
```