Université Paris Sud - Licence/IUP2 - Année 2000/2001

Projet d'algorithmique n° 3

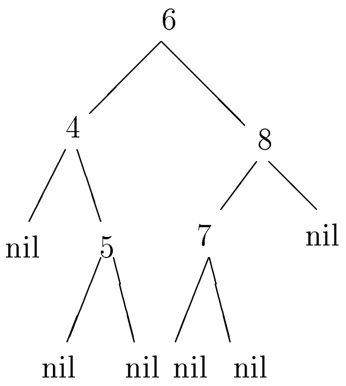
La recherche adaptative dans les arbres binaires de recherche

a) Les arbres binaires de recherche

Dans ce projet on représentera les *arbres binaires de recherche* avec la syntaxe abstraite suivante:

- Un arbre vide est représenté par la constante `N`.
- Un arbre non vide est représenté par le terme `A(n,fg,fd)` où `n` est un entier et `fg` (resp. `fd`) est la représentation du fils gauche (resp. droit) du nœud étiqueté avec `n`.

Ainsi par exemple



est représenté par $A(6, A(4, N, A(5, N, N)), A(8, A(7, N, N), N))$.

b) Les rotations dans les arbres binaires de recherche

On appelle *rotation gauche autour du noeud* `n` la transformation d'un arbre où la configuration $A(n, X, A(n', Y, Z))$ est remplacée par la configuration $A(n', A(n, X, Y), Z)$. On appelle *rotation droite autour du noeud* `n` la transformation d'un arbre où la configuration $A(n, A(n', X, Y), Z)$ est remplacée par la configuration $A(n', X, A(n, Y, Z))$.

Dans ce projet on travaillera uniquement avec des arbres binaires de recherche sans répétitions d'éléments. On remarquera dans tels arbres, le résultat d'une rotation gauche ou droite donne toujours un arbre binaire de recherche vérifiant la même condition.

Le programme suivant vous donne des fonctions de lecture, impression, destruction et rotation gauche d'arbres binaires.

```
#include <stdio.h>
```

```
typedef enum
{ false, true }
bool;

typedef struct tmp_abr
{
  int val;
  struct tmp_abr *fg, *fd;
} rec_arbre;

typedef rec_arbre *arbre;

arbre Lire_Arbre ()
{
int info;
char c;
arbre g,d,res;
  c=getchar();
  if (c=='N')
    /* cas feuille */
    return NULL;
  else
    {/* cas non feuille */
     /* skip ( */
     getchar();
     /* lire entier */
     scanf("%d",&info);
     /* skip , */
     getchar();
     /* lire sous-arbre gauche */
     g=Lire_Arbre();
    /* skip , */
     getchar();
     /* lire sous-arbre droit */
     d=Lire_Arbre();
    /* skip ) */
     getchar();
     /* creation du noeud interne */
     res=(arbre)malloc(sizeof(rec_arbre));
     res->fg=g;
     res->val=info;
     res->fd=d;
     return(res);
    }
}
```

```
void Imprimer_Arbre (arbre a)
{
  if (a==NULL)
     /* cas feuille */
      printf("N");
  else
    {/* cas non feuille */
     printf("A(%d,",a->val);
     /* imprime sous-arbre gauche */
     Imprimer_Arbre(a->fg);
     printf(",");
     /* imprime sous-arbre droit */
     Imprimer_Arbre(a->fd);
     printf(")");
    }
}

/* Rotation_Gauche(a) renvoie l'arbre qui resulte de faire
   une rotation gauche autour de la racine de l'arbre a */

arbre Rotation_Gauche(arbre a)
{
arbre tmp,res;

  if (a!=NULL)
     /* cas non feuille */
     {if (a->fd !=NULL)
         { res=a->fd;
           tmp = (a->fd)->fg;
           a->fd=tmp;
           res->fg=a;
           return(res);}
      else return(a);
      }
  else /* cas feuille */
       return(a);
}

main()
{
arbre a;

  a=Lire_Arbre();
  Imprimer_Arbre(a);
  printf("\n");
  Imprimer_Arbre(Rotation_Gauche(a));
  printf("\n");
}
```

En pratique, quand on fait la recherche d'un élément dans un arbre binaire de recherche, on peut constater que ce sont souvent les mêmes éléments qui sont recherchés, et il est donc intéressant de rapprocher les éléments recherchés de la racine. Ainsi, à chaque fois que l'on effectue la recherche d'un élément `n`, on veut effectuer une rotation (gauche ou droite selon le cas) autour du père de `n` pour monter `n` d'un cran.

c) But du Projet

Ce projet consiste à écrire une procédure

```
bool Recherche_et_Rotation(arbre a, arbre *res,int n)
```

qui prend en argument un arbre binaire de recherche `a`, et un entier `n` et renvoie `true` si et seulement si l'entier `n` appartient à l'arbre `a`. Dans le cas où `n` appartient à l'arbre `a` l'arbre `*res` sera le résultat d'une rotation (gauche ou droite) autour du père de `n`, quand cela a un sens.

Votre programme principal devra comporter les lignes suivantes:

```
main()
{
arbre a,res;
int n;
bool b;

  n=Lire_Element();
  a=Lire_Arbre();
  Imprimer_Entier(n); printf("\n");
  Imprimer_Arbre(a);  printf("\n");
  b=Recherche_et_Rotation(a, &res, n);
  Imprimer_Arbre(res);
  printf("\n");
}
```

d) Jeux de tests

A titre indicatif, voici quelque jeux de tests:

n	a	*res
1	N	N
1	$A(2, A(1, N, N), A(4, A(3, N, N), A(5, N, N)))$	$A(1, N, A(2, N, A(4, A(3, N, N), A(5, N, N))))$
2	$A(2, A(1, N, N), A(4, A(3, N, N), A(5, N, N)))$	$A(2, A(1, N, N), A(4, A(3, N, N), A(5, N, N)))$
3	$A(2, A(1, N, N), A(4, A(3, N, N), A(5, N, N)))$	$A(2, A(1, N, N), A(3, N, A(4, N, A(5, N, N))))$
4	$A(2, A(1, N, N), A(4, A(3, N, N), A(5, N, N)))$	$A(4, A(2, A(1, N, N), A(3, N, N)), A(5, N, N))$
5	$A(2, A(1, N, N), A(4, A(3, N, N), A(5, N, N)))$	$A(2, A(1, N, N), A(5, A(4, A(3, N, N), N), N))$
8	$A(2, A(1, N, N), A(4, A(3, N, N), A(5, N, N)))$	$A(2, A(1, N, N), A(4, A(3, N, N), A(5, N, N)))$

e) Comment rendre le programme?

Vous rendrez votre programme par mail avec la commande

```
mail waller@lri.fr  < fichier.c
```