

# Informatique

## Corrigé du concours 2008

### 1 Recherche trichotomique

a. Le principe de la recherche trichotomique est le suivant :

La recherche de l'élément  $x$  se fait sur une liste  $\lambda$  dont les  $n$  éléments sont compris entre deux bornes : une gauche  $g$  et une droite  $d$  initialisées respectivement aux valeurs 1 et  $n$ .

Le principe de recherche est récurrent :

Tant que  $g + 1 < d$  (au moins deux valeurs d'écart) on calcule deux valeurs de pivots  $p1 = (2g + d) \text{ div } 3$  et  $p2 = (g + 2d) \text{ div } 3$ . On regarde alors si  $x = \text{ieme}(\lambda, p1)$  ou si  $x = \text{ieme}(\lambda, p2)$ .

Si c'est le cas, la recherche est positive et l'on retourne  $p1$  ou  $p2$  selon le cas.

Sinon, on poursuit la recherche de  $x$  sur un intervalle réduit au tiers des éléments.

Dans ce cas, la question est : Quel est cet intervalle ?

Les réponses possibles sont les suivantes :

- (a) si  $x < \text{ieme}(\lambda, P1)$  alors on recommence sur l'intervalle  $[g, p1 - 1]$
- (b) si  $\text{ieme}(\lambda, P1) < x < \text{ieme}(\lambda, P2)$  alors on recommence sur l'intervalle  $[p1 + 1, p2 - 1]$
- (c) si  $\text{ieme}(\lambda, P2) < x$  alors on recommence sur l'intervalle  $[P2 + 1, d]$

Lorsque les bornes  $g$  et  $d$  se croisent, la recherche est négative (l'élément  $x$  n'existe pas dans la liste). Dans ce cas, on retourne 0.

*remarque : Nous pourrions continuer la récursion lorsque les bornes ont moins de deux valeurs d'écart, mais nous ferions des tours de plus et des calculs de pivots inutiles. L'idéal est, lorsque les bornes sont égales ou n'ont qu'une valeur d'écart, de tester la valeur de l'élément directement sur ces bornes. C'est cette solution qui est retenue pour l'algorithme et l'arbre d'exécution demandés.*

b. Algorithme :

Plutôt que de fournir des solutions en C, Caml ou Pascal, nous avons préféré fournir une solution algorithmique en pseudo-langage facilement implémentable dans un de ces langages.

Cette solution ne tient pas compte de l'aspect débranchant des **Retourne** et positionne à chaque fois les **sinon**. C'est bien évidemment optimisable.

```

algorithme fonction trichotomie : Entier
parametres locaux
  Element x
  Liste l
  Entier g,d
variables
  Entier p1,p2
debut
  sig+1<d alors
    p1 ←(2*g+d)div 3
    p2 ←(g+2*d)div 3
    six=ieme(l,p1) alors

```

```

retourne p1          /* Recherche positive sur p1 */
sinon
  six=ieme(l,p2) alors
    retourne p2      /* Recherche positive sur p2 */
  sinon
    six<ieme(l,p1) alors
      retourneTrichotomie(x,l,g,p1-1)
    sinon
      six<ieme(l,p2) alors
        retourneTrichotomie(x,l,p1+1,p2-1)
      sinon
        retourneTrichotomie(x,l,p2+1,d)
      fin si
    fin si
  fin si
fin si
sinon
  six=ieme(l,g)
  retourne g        /* Recherche positive sur g ou d */
sinon
  six=ieme(l,d)
  retourne d        /* Recherche positive sur d */
sinon
  retourne 0        /* Recherche négative */
fin si
fin si
fin si
fin algorithme fonction trichotomie
    
```

c. L'arbre d'évaluation de la recherche d'un élément  $x$  dans une liste  $\lambda$  de 20 éléments est celui de la figure 1.

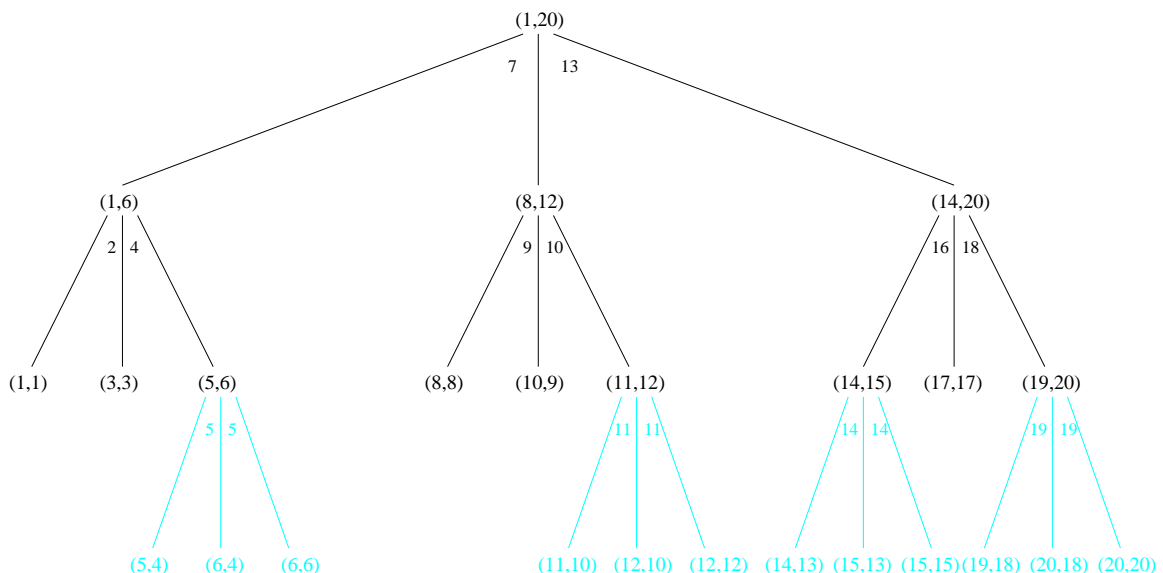


FIG. 1 – Exécution d'une trichotomie sur 20 éléments

*Remarque : Nous avons laissé la trichotomie visible lorsqu'il n'y a qu'une valeur d'écart entre  $g$  et  $d$ . Ceci afin de posséder toutes les valeurs de pivots. Mais notre algorithme s'arrête au niveau supérieur et teste les deux valeurs de borne.*

- d. Le nombre maximum de comparaisons dans ce cas pour une recherche négative est 3
- e. Le nombre de comparaisons de la recherche trichotomique positive d'un élément  $x$  pris parmi  $n$  est majoré par  $\log_3(n)$
- f. Dans la mesure où la trichotomie offre trois choix à chaque évaluation, Le nombre d'éléments  $n$  de la liste est majoré par  $3^x$  où  $x$  est le nombre d'évaluations nécessaires à atteindre n'importe quel élément. Donc  $x = \log_3(n)$ .

## 2 Arbres de Fibonacci

- a. L'arbre  $F_6$  de Fibonacci est celui de la figure 2 dont les noeuds contiennent leur propre valeur de déséquilibre.

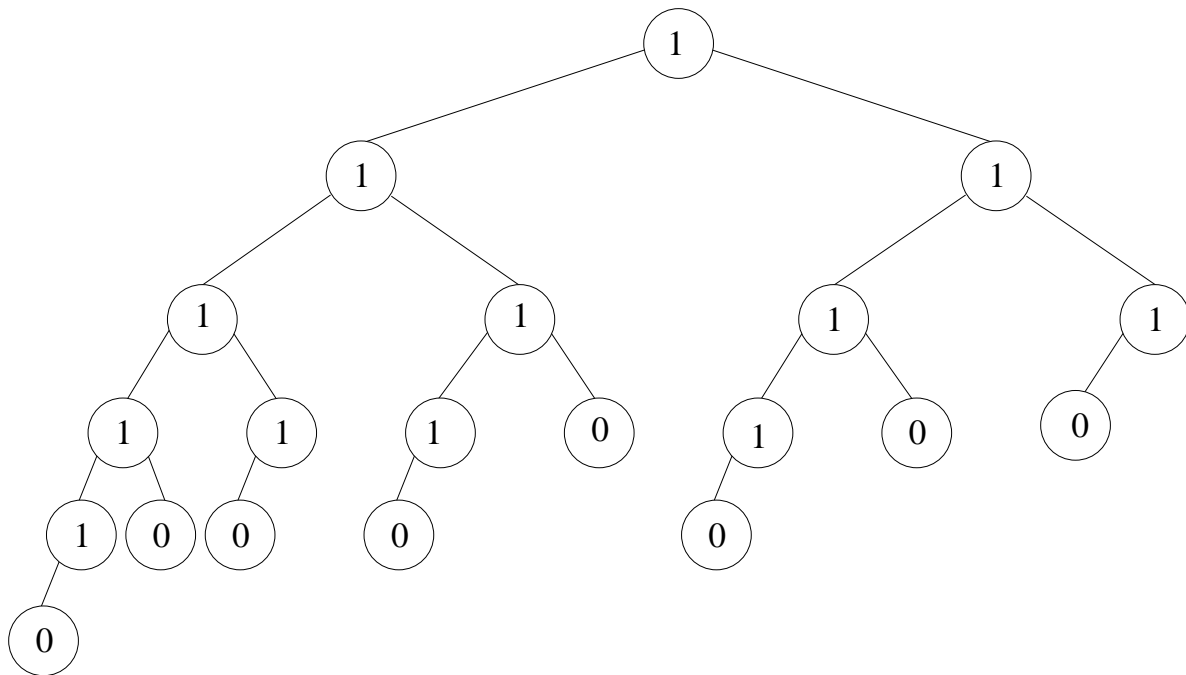


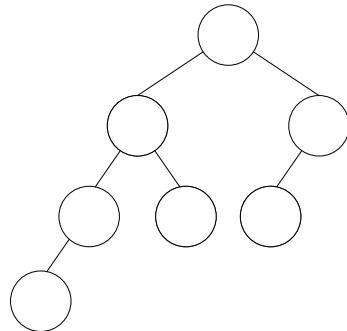
FIG. 2 – Arbre de Fibonacci  $F_6$

- b. Tableau des valeurs :

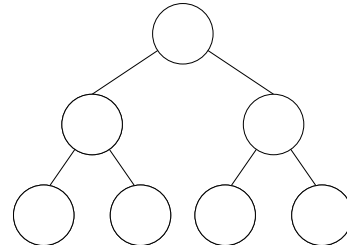
$n$	$H_n$	$T_n$	$F_n$	$Fib_n$
0	–	0	0	0
1	0	1	1	1
2	1	2	1	1
3	2	4	2	2
4	3	7	3	3
5	4	12	5	5
6	5	20	8	8

- c.  $H_n = n - 1$  évident par récurrence  
 $T_n = Fib_{n+2} - 1$  par récurrence aussi en notant que  $T_n = T_{n-1} + T_{n-2} + 1$  et en sachant que  $Fib_n = Fib_{n-1} + Fib_{n-2}$   
 $F_n = Fib_n$  Vérifient la même récurrence, donc  $F_n = Fib_n = Fib_{n-1} + Fib_{n-2}$

- d. Oui, il existe des arbres (1-Equilibrés) qui pour le même nombre de noeuds qu'un arbre de Fibonacci de hauteur  $h$  ont une longueur de cheminement externe supérieur à celui-ci.
- e. Comme le montre les figures suivantes, il existe au moins l'arbre parfait (1-Equilibrés) de 7 sommets dont la longueur de cheminement externe est supérieure à celle de l'arbre de Fibonacci de 7 sommets.



Arbre de Fibonacci à 7 Noeuds (LCE=7)



Arbre parfait à 7 Noeuds (LCE=8)

- f. Il y a deux façons de déterminer la longueur de cheminement externe d'un arbre de Fibonacci :
- (a) la première (simple) consiste à faire le parcours en profondeur de l'arbre (arbre binaire) en maintenant la profondeur courante et la longueur de cheminement externe dans des paramètres globaux. A chaque fois que l'on rencontre une feuille, on accumule sa profondeur à la longueur de cheminement externe.
- (b) la deuxième est de faire la aussi un parcours en profondeur de l'arbre, mais en appliquant, avec  $g(x)$ ,  $d(x)$  et  $nbf(x)$  respectivement les fils gauche, fils droit et nombre de feuille de  $x$ , la formule de calcul de  $LCE(x)$  suivante :

$$\begin{cases} LCE(x)=0 \text{ si } x \text{ est une feuille,} \\ LCE(x)=LCE(g(x))+nbf(g(x))+LCE(d(x))+nbf(d(x)) \text{ sinon.} \end{cases}$$