Université Paris 7 - Licence d'informatique - L2

EA4 – Corrigé de l'examen du 30 Mai 2012

Les réponses sont en [illegible].

Exercice 1 (4 points).

Soit la fonction :

```
int f(int n) {
  int m = abs(n); int res = 0;
  tantque  (m>1) faire
     si (m est pair) alors m:=m/2 sinon m:=m-8;
     res=res+1;
  ftq;
  retourne(res);
}
```

1. Quel est le temps de calcul de f relativement à n dans le meilleur cas ? (Donner la quantité exacte, pas le Grand-O).
   [illegible] $\lceil \log_2 |n| \rceil$ [illegible].
2. Quel est le temps de calcul de f relativement à n dans le pire des cas ? (Donner la quantité exacte, pas le Grand-O).
   [illegible] $\lceil \frac{|n|}{8} \rceil$ [illegible].
3. Donner une valeur de n qui réalise le meilleur cas et une qui réalise le pire des cas.
   [illegible] : 128.
   [illegible] : 371.
4. Quelle est la probabilité d'avoir le meilleur cas ?
   [illegible] $\lceil \log_2(\texttt{MaxInt}) \rceil$ [illegible] $\frac{\lceil \log_2(\texttt{MaxInt}) \rceil}{\texttt{MaxInt}}$.
5. Et celle d'avoir le pire des cas ?
   [illegible] $\frac{1}{2}$.

Exercice 2 (10 points).

Les fonctions demandées dans cet exercice prennent en paramètre un arbre binaire donné par l'adresse de sa racine, qui est un objet de type :

```
struct noeud {int val ; struct noeud * g ; struct noeud * d} ;
```

Elles doivent toutes avoir une complexité en temps en $O(N)$ où N est le nombre de noeuds de l'arbre. Vous expliquerez les raisonnements derrière vos algorithmes et vous justifierez leur complexité. Veuillez noter que des solutions élégantes et succinctes existent, qui comprennent environ dix lignes de pseudo-code pour les deux premières questions, et moins de vingt pour la troisième.

1. Une fonction booléenne qui retourne VRAI si et seulement dans l'arbre binaire donné tout noeud interne a exactement deux fils. (3 points)

```
Algo EstBinaireStrict (A : Noeud *) : booléen {
    Si (A = NULL) alors retourner (VRAI);
    Si ((A -> fg = NULL) ET (A -> fd = NULL)) alors retourner (VRAI):
    Si ((A -> fg = NULL) OU (A -> fd = NULL)) alors retourner (FALSE);
    Retourner (EstBinaireStrict (A -> fg) ET EstBinaireStrict (A -> fd));
)
```

2. Une fonction qui, pour un entier h, retourne h si dans l'arbre binaire donné tout noeud interne a exactement deux fils et toutes les feuilles se trouvent à la même profondeur h. Retourne -2 sinon. (Pourquoi on ne fait pas retourner -1 ?) (3 points)

```
Algo EstBinaireComplet (A : Noeud *) : int {
    Si (A = NULL) alors retourner (-1);
    Si ((A -> fg = NULL) ET (A -> fd = NULL)) alors retourner(0):
    Si ((A -> fg = NULL) OU (A -> fd = NULL)) alors retourner (-2);
    int hg := EstBinaireComplet (A -> fg);
    int hd := EstBinaireComplet (A -> fd);
    Si ((hg = hd) ET (hg != -2)) alors retourner (hg+1)
    Sinon retourner (-2)
}
```

```
Algo EstBinaireCompletAux (A : Noeud *, h: int) : int {
    int k := EstBinaireComplet (A);
    Si (k = h) alors retourner (k)
    Sinon retourner (-2)
}
```

3. Une fonction qui, pour un entier h, retourne h si l'arbre binaire donné a la forme d'un tas de hauteur h (tous les niveaux sont "complets", sauf éventuellement le dernier de profondeur h, dans lequel les noeuds "occupés" sont ceux les plus à gauche). Retourne -2 sinon. (4 points)

couple, o` n couple po typeforme co pon n a` {Complet, Tas, Autre} n con co po n int co pon n

n on c o p nc p n :

n b co , co c , n) h − 1.

n b non n h n , n g e n :

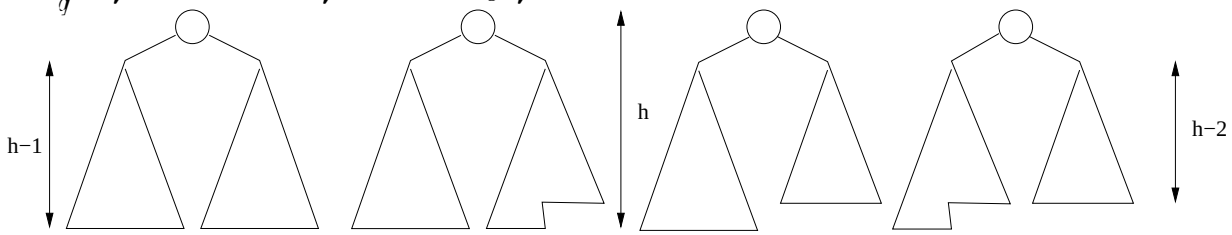
o b g b complet h − 1 o b o complet h − 1 (on n n b co , n c) ;

o b g b complet h−1 o b o n tas h − 1 (co pon n e o` n n p po p h − 1 (co pon n e o` n n p c n` o) ;

o b g b complet h − 1 o b o complet h − 2 (co pon n e o` n n p c n` o) ;

o b g b n tas h − 1 o b o complet h − 2 (co pon n e o` n n p po p o n h o) ;

L g on n h g n :



' o` go h :

```
Algo Test (A : Noeud *): (typeforme, int) {
    Si (A = NULL) alors retourner  (Complet, -1);
    (formeG, hG)=Test(A -> fg);
    (formeD, hD)=Test(A -> fd);
    Si (hG = hD) alors {
        si ((formeG = formeD) ET (formeG = Complet) alors retourne (Complet, hG+1);
        si ((formeG = Complet) ET (formeD = Tas) alors retourne (Tas, hG+1);
    }
    Si (hG = hD+1) alors {
          Si  (((formeG = Tas) OU (formeG = Complet)) & (formeD = Complet))
          alors  retourne (Tas,hG+1)
    };
    retourne (Autre, max(hG, hD)+1)
}

Algo EstTas(A : Noeud *) : int {
    (forme, hauteur) := Test(A);
    Si ((forme = Complet) OU (forme = Tas)) retourne (hauteur))
    Sinon retourne (-2);
}
```

n o on , p co p , c , b ' r ' a n :
On n p co p n x (n g p g) a b r ' a a , n file ( ).
n 'on ncon n no a Xa g n a x ' , ona o ' a a x ' a ' on o
X g ' . Q n a ona p no a X a o n a x ' , on X
a ' n n ' a o (n c e , ona c c g on o n g −2) o ' a ' n n

```
Algo EstTasIter (A : Noeud *) : int {
    Si (A == NULL) alors retourner (-1);
    Si ( (A -> fg == NULL) ET (A -> fd == NULL) ) alors retourner(0):
    F : File de Noeud*;
    X : Noeud *;
    compt : int ;
    X := A;
    compt := 1;
    Tantque ((X -> fg != NULL) ET (X -> fd != NULL) Faire
        Enfiler (F, X -> fg);
        Enfiler (F, X -> fd);
        compt := comp +2;
        X := defiler (F);
    ftq;
    Si (X -> fd) != NULL) alors retourner (-2);
    Si (X -> fg) != NULL) alors {Enfiler (F, X -> fg); compt++}
    tantque ( ! EstVide(F) ) faire
        X := defiler (F);
        Si (! EstFeuille (X)) alors retourner (-2);
    ftq;
    retourner (log_2(compt))
}
```

Exercice 3 (6 points).

Proposer un algorithme pour trier des valeurs entières stockées dans un tableau qui utilise comme structure auxiliaire un ABR. Les fonctions de gestion des ABR ne devront pas être réécrites.

```
Algo TriParABR (T : int [ ]) : int [ ]{
    A : ABR;
```

```
    A:= NouveauABR();
    Pour i de 1 à taille(T) faire InsertionABR(A, T[i]);
    Pour i de 1 à taille(T) faire T[i] := SuppriMinABR(A);
    retourner (T);
}
```

On suppose que l'on a :

```
Algo TriParABR (T : int [ ]) : int [ ] {
    A : ABR;
    A:= NouveauABR();
    Pour i de 1 à taille(T) faire InsertionABR (A, T[i]);
    ParcoursPrefixe (A, T, 1);
    retourner (T);
}

Algo ParcoursPrefixe(A : ABR, T : int [ ], i int) {
    Si A != Null alors {
         i := ParcoursPrefixe(A -> fg, T, i);
         T[i] := A -> val;
         i++;
         i := ParcoursPrefixe(A -> fd, T, i);
     }
}
```

1. Quel est le pire des cas de votre algorithme et quelle est sa complexité en temps ?
   On a pu voir en cours que l'algorithme InsertionABR / SuppriMinABR s'effectue en un temps proportionnel à la hauteur h de l'arbre. Le pire des cas correspond à un arbre filiforme et est en $O(n)$ où n est le nombre de noeuds de l'arbre.
   On a également vu en cours que la hauteur d'un arbre binaire de recherche à n noeuds est au plus n (arbre filiforme) et au moins $\log_2(n)$ (arbre complet). A chaque noeud on effectue une opération en temps constant, donc le pire des cas correspond à un arbre filiforme. Pour cet arbre :

$$1 + 2 + 3 + \ldots + (n-1) + n = \frac{n(n+1)}{2} \in O(n^2)$$

   Le meilleur des cas correspond à un arbre complet.
   *Cet algorithme est **en particulier** T **soit en ordre croissant soit en ordre décroissant.***
   Si on a un arbre filiforme, alors on a une complexité en $O(n)$, si on a un arbre complet, on a une complexité en $O(n^2)$ (pour chaque noeud, on parcourt tous les noeuds restants).
   Si on a un arbre filiforme, alors on a une complexité en $O(n)$, car chaque noeud est visité une fois et une seule, et on a n noeuds. Si on a un arbre complet, on a une complexité en $O(n^2)$ (pour chaque noeud, on parcourt tous les noeuds restants).
2. Quelle est la complexité en temps du meilleur des cas de votre algorithme ?
   Si on a un arbre complet, on a une complexité en $O(\log n)$. Le pire des cas correspond à un arbre filiforme et est en $O(n \log n)$ (pour chaque noeud, on parcourt tous les noeuds restants).
3. Quelle complexité en moyenne vous attendez pour votre algorithme ? Pourquoi ?
   Le cas précédent est rare, on a donc une complexité en moyenne en $O(n \log n)$ et non pas en $O(\log n)$.
4. Que peut-on dire des complexités si on utilise un AVL à la place d'un ABR ?
   Les AVL sont des ABR équilibrés. Le pire des cas est donc en $O(\log n)$. Le cas précédent est rare, on a donc une complexité en $O(n \log n)$ (pour chaque noeud, on parcourt tous les noeuds restants).