

Corrigé du Contrôle d'Informatique INF 311

Promotion 2010

Sujet proposé par François Morain, Philippe Chassignet, Étienne Duris

11 juillet 2011

Seuls les documents fournis dans le cadre du cours et les notes personnelles sont autorisés.

Durée : 2 heures. Les étudiants EV2 ont le droit à 30 minutes supplémentaires.

Les trois exercices sont indépendants. Un barème pour chaque exercice, pour un total de 20 points, est donné à titre indicatif. Nous donnons le nombre de lignes attendues pour la réponse à chaque question ; si la réponse que vous envisagez est beaucoup plus longue, il est probable qu'il s'agisse d'une solution trop compliquée.

Les exercices qui suivent sont indépendants et peuvent être traités dans n'importe quel ordre. On attachera une grande importance à la clarté, à la précision et à la concision de la rédaction.

Exercice 1

barème envisagé : 2 points

On considère la fonction suivante :

```
public static String f(int n) {
    if (n <= 1)
        return "" + n;
    else {
        int m = n/2;
        String s = "(" + f(m) + ") * 2";
        if ((n % 2) == 1)
            s += "+1";
        return s;
    }
}
```

a) Que retourne $f(13)$?

(réponse attendue : 1 ligne)

Corrigé.

"((1)*2+1)*2)*2+1"

b) Que fait la fonction f pour $n \geq 0$?

(réponse attendue : moins de 5 lignes)

Corrigé. Si n est un entier, on peut l'écrire sous la forme $n = n_t 2^t + n_{t-1} 2^{t-1} + \dots + n_0$ avec $n_i \in \{0, 1\}$ et $n_t = 1$ que l'on peut récrire sous la forme

$$n = (\dots(((1)2 + n_{t-1})2) \dots)2 + n_0.$$

La fonction retourne la chaîne de caractères correspondant à l'expression symbolique.

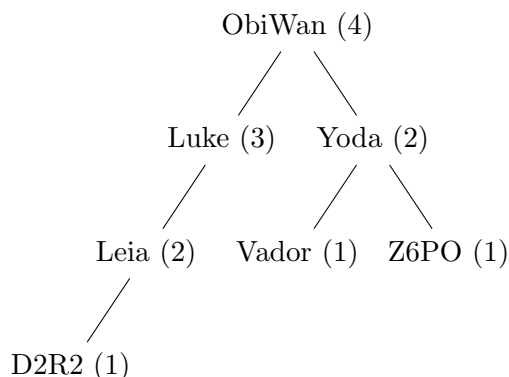
Exercice 2

barème envisagé : 6 points

On considère un arbre binaire de recherche dans lequel chaque nœud possède un champ supplémentaire, qui est la hauteur de l'arbre dont il est racine. On note $H(A)$ la hauteur d'un arbre A . La hauteur de l'arbre **null** est 0, celle d'une feuille est 1, et plus généralement celle de (r, A_g, A_d) est $1 + \max(H(A_g), H(A_d))$.

Ce type d'*enrichissement* des données est très fréquent. Dans le contexte qui nous intéresse, c'est le premier pas vers les algorithmes d'équilibrage d'arbres.

On a indiqué sur l'exemple qui suit la hauteur pour chaque nœud (donc la hauteur du sous-arbre de racine le nœud en question) :



Le but de l'exercice est de modifier (aussi peu que possible) la classe ABR vue en cours, de façon à créer une nouvelle classe ABRH :

```
public class ABRH{
    public String nom;
    public ABRH filsg, filsd;
    // à compléter
}
```

a) Ajoutez la déclaration d'un champs hauteur, dont on veut qu'il soit résultat d'opérations sur l'arbre et dont la valeur ne puisse être modifiée sans passer par les méthodes du nœud.

(réponse attendue : moins de 5 lignes)

Corrigé. Une bonne façon d'atteindre le but est de déclarer le champ privé :

```
public class ABRH{
    public String nom;
    public ABRH filsg, filsd;
    private int hauteur;
}
```

b) Écrire une méthode

```
public static int hauteur(ABRH a){...}
```

qui retourne la hauteur d'un ABRH.

(réponse attendue : moins de 5 lignes)

Corrigé.

```

public static int hauteur(ABRH a) {
    if(a == null)
        return 0;
    else
        return a.hauteur;
}

```

c) Définissez deux constructeurs pour la classe ABRH, l'un général, l'autre pour une feuille, qui mettent à jour la hauteur du champ. On pourra utiliser

`Math.max(int a, int b)`

qui retourne un entier, le maximum de a et b.

(réponse attendue : 5 à 10 lignes)

Corrigé.

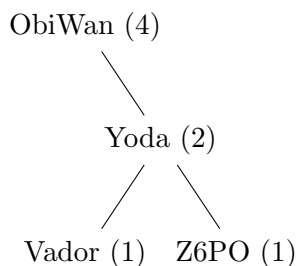
```

public ABRH(String r, ABRH g, ABRH d) {
    this.racine = r;
    this.filsg = g;
    this.filsd = d;
    this.hauteur = 1 + Math.max(hauteur(g), hauteur(d));
}

public ABRH(String r) {
    this.racine = r;
    this.filsg = null;
    this.filsd = null;
    this.hauteur = 1;
}

```

d) Faites un schéma de la représentation en mémoire de l'arbre (extrait de l'arbre exemple) :



en faisant apparaître les références des cases mémoires comme on l'a fait dans le cours pour les listes.

(réponse attendue : un dessin)

Corrigé. Voir la figure 1, qui contient l'occupation mémoire de l'arbre exemple complet.

e) Écrire une fonction

```

public static ABRH ajouter(ABRH a, String nom) {

```

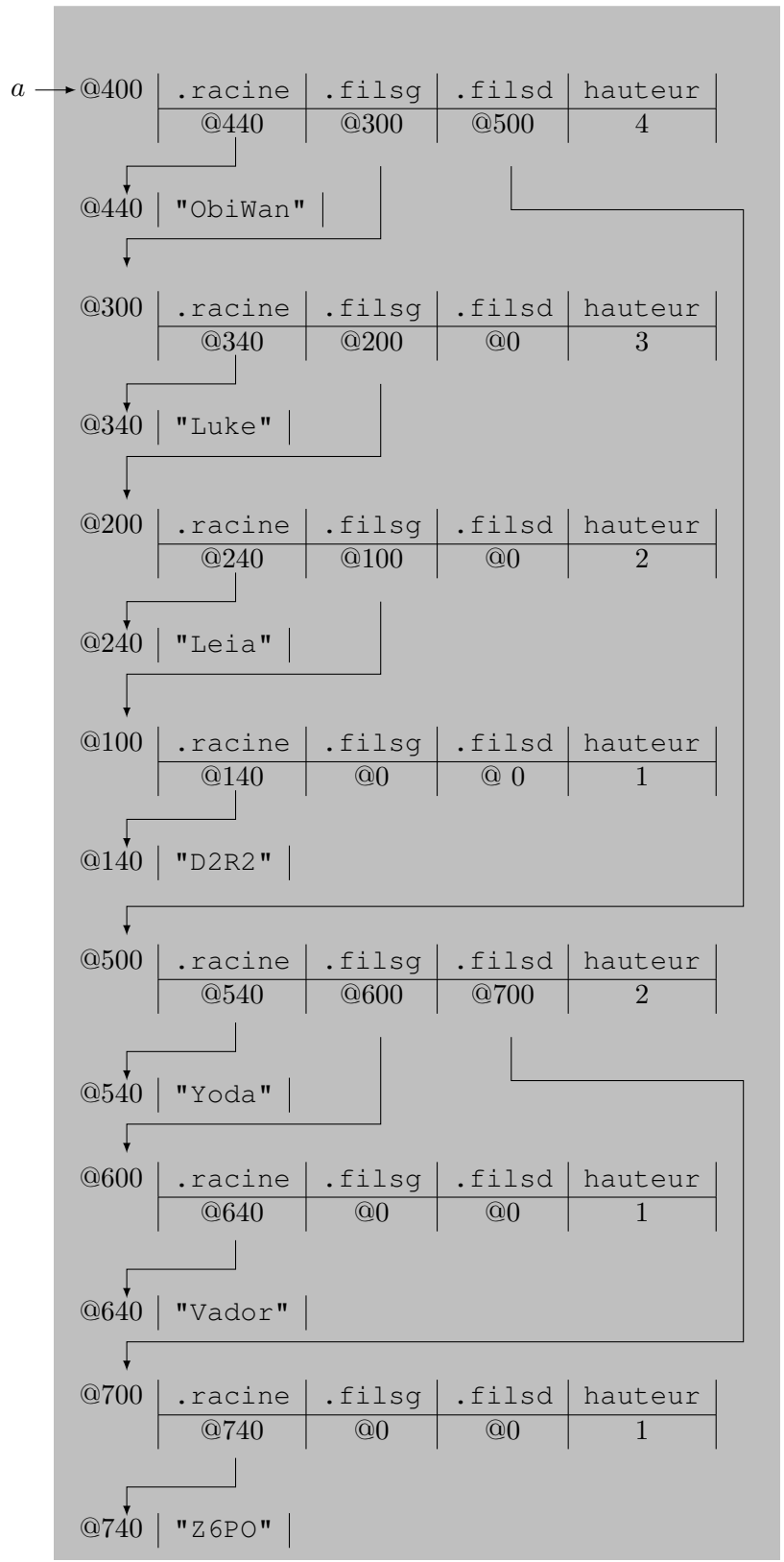


Figure 1: Transcription mémoire de l'arbre.

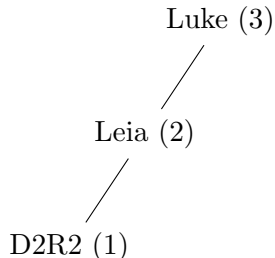
qui insère un nouveau nœud pour la chaîne `nom` dans l'arbre `a`. Cette fonction devra bien sûr respecter l'ordre ABR, mais également mettre à jour les nœuds sur le parcours, de sorte que la hauteur de tous les nœuds de l'arbre soient à jour après l'appel. On rappelle que si `s` et `t` sont deux `String`, l'instruction `s.compareTo(t)` retourne 0 si les deux chaînes sont égales, un nombre strictement négatif si `s` est avant `t` dans l'ordre lexicographique et un nombre strictement positif sinon.

(réponse attendue : 5 à 10 lignes)

Corrigé.

```
public static ABRH ajouter(ABRH a, String nom){
    if(a == null)
        return new ABRH(nom);
    else if(a.racine.compareTo(nom) > 0)
        a.filsg = ajouter(a.filsg, nom);
    else
        a.filsd = ajouter(a.filsd, nom);
    a.hauteur = 1 + Math.max(hauteur(a.filsg), hauteur(a.filsd));
    return a;
}
```

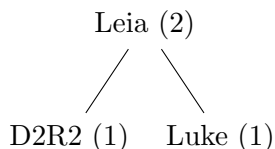
f) Nous donnons ici un exemple de rééquilibrage dans un cas particulier. On reprend l'exemple donné au début de l'exercice et on donne le code conduisant au sous-arbre :



c'est-à-dire

```
ABRH c = new ABRH("D2R2");
ABRH b = new ABRH("Leia", c, null);
ABRH a = new ABRH("Luke", b, null);
```

Donnez les instructions qui permettent d'obtenir l'arbre suivant sans créer de nouveau nœud.



(réponse attendue : moins de 5 lignes)

Corrigé.

```
a.filsg = null;
b.filsd = a;
a.hauteur = 1;
```

Exercice 3

barème envisagé : 12 points

La célèbre École de sorciers de PourDeLart organise sa fête des élèves lors du solstice d'hiver chaque année, fête appelée Point Glagla. Les élèves d'avant-dernière année sont chargés de différentes activités proposées aux participants de la fête (les élèves des autres années, leurs frères et sœurs, leurs parents, etc.).

Comme la magie ne peut pas résoudre tous les problèmes, le comité organisateur envisage de développer un programme de gestion de la fête. Votre mission (si vous l'acceptez) est de les aider dans cette tâche.

a) Écrire la classe `Sorcier` qui décrit un sorcier : nom, prénom, matricule (trois chaînes de caractères) ; on ajoutera un constructeur explicite prenant trois champs ainsi qu'une méthode d'objet

```
public String enChaine(){...}
```

qui est chargée de fabriquer et retourner une chaîne de caractères décrivant un sorcier au format
nom prénom matricule

(réponse attendue : 5 à 10 lignes)

Corrigé.

```
public class Sorcier{  
    public String nom, prenom, matricule;  
  
    public Sorcier(String n, String p, String m){  
        this.nom = n;  
        this.prenom = p;  
        this.matricule = m;  
    }  
  
    public String enChaine(){  
        return this.nom+" "+this.prenom+" "+this.matricule;  
    }  
}
```

b) La fête est organisée en différentes sortes d'activités : bar, restaurant, spectacle, jeu. Chaque activité a un nom, un responsable (un sorcier) assisté d'une équipe de plusieurs sorciers dont le nombre est fixé à la création de l'activité.

b-i) Écrire la classe `Activite` et choisir les champs de la classe en justifiant vos choix. On ajoutera un constructeur explicite

(réponse attendue : 15 à 20 lignes)

Corrigé. Comme on connaît le nombre de sorciers, on peut construire un tableau de taille correspondante. Cela donne

```
public class Activite{  
    public String nom;  
    public Sorcier responsable;  
    public Sorcier[] equipe;
```

```

public Activite(String n, Sorcier r, Sorcier[] e){
    this.nom = n;
    this.responsable = r;
    this.equipe = e;
}

```

b-ii) Écrire la classe PointGlagla avec 40 activités au maximum stockées dans une variable de classe. Écrire une fonction

```

public static void initialiser(){ ... }

```

qui initialisera cette variable avec une seule activité laissée à votre imagination.

(réponse attendue : 5 à 10 lignes)

Corrigé.

```

public class PointGlagla{

    public static final int NB_ACTIVITES = 40;
    public static Activite[] act = new Activite[NB_ACTIVITES];

    public static void initialiser(){
        Sorcier hp = new Sorcier("Potter", "Harry", "WZ0011");
        Sorcier hg = new Sorcier("Granger", "Hermione", "WZ0012");
        Sorcier rw = new Sorcier("Weasley", "Ron", "WZ0013");

        Sorcier[] tab = new Sorcier[2];
        tab[0] = hp;
        tab[1] = rw;
        act[0] = new Activite("fire's cup", hg, tab);
    }
}

```

c) Chaque activité est payante. Le comité d'organisation veut disposer de la liste des clients (participants) et des factures payées.

Chaque transaction dans une activité est représentée par un objet de la classe Commande :

```

public class Commande {
    public String client;
    public int montant;
    public String detail;

    public Commande(String c, int m, String d) {
        this.client = c;
        this.montant = m;
        this.detail = d;
    }
}

```

Par exemple, si Hagrid commande une bière-au-beurre, sa commande sera représentée par

```

Commande c = new Commande("Hagrid", 1, "bière-au-beurre");

```

(rappelons que l'unité monétaire en cours est le galion).

c-i) Écrire une classe `ListeCommandes` représentant un maillon d'une liste chaînée dont les champs comprendront nécessairement un objet `Commande`. On écrira également un constructeur

```
public ListeCommandes(String c, int m, String d, ListeCommandes suiv){  
}
```

qui rajoute un client de nom `c`, un montant `m` et un détail `d`, en tête d'une liste `suiv`.

On écrira également un constructeur

```
public ListeCommandes(Commande cmd, ListeCommandes suiv){ ... }
```

qui prend un objet `Commande` et une liste en paramètres. Ce constructeur devra réaliser une copie de `cmd`.

(réponse attendue : 5 à 10 lignes)

Corrigé.

```
public class ListeCommandes{  
    public Commande commande;  
    public ListeCommandes suivant;  
  
    public ListeCommandes(String c, int m, String d, ListeCommandes suiv){  
        this.commande = new Commande(c, m, d);  
        this.suivant = suiv;  
    }  
  
    public ListeCommandes(Commande cmd, ListeCommandes suiv){  
        this.commande = new Commande(cmd.c, cmd.m, cmd.d);  
        this.suivant = suiv;  
    }  
}
```

c-ii) On souhaite garder l'ordre chronologique dans lequel les clients ont commandé. La tête de liste sera le premier client. Écrire une méthode de classe

```
public static ListeCommandes ajouter(ListeCommandes l, Commande cmd){  
    ...}
```

qui ajoute la nouvelle commande à la liste `l`, en respectant cet ordre et qui retourne la liste résultat.

(réponse attendue : moins de 5 lignes)

Corrigé. Il s'agit ici d'un ajout en queue de la liste :

```
public static ListeCommandes ajouter(ListeCommandes l, Commande cmd){  
    if(l == null)  
        return new ListeCommandes(cmd, l);  
    else{  
        l.suivant = ajouter(l.suivant, cmd);  
        return l;  
    }  
}
```

c-iii) Écrire une méthode


```
public static ListeCommandes enlever(String c, ListeCommandes l){}
```

qui retire de la liste l la première commande au nom du client c et qui retourne la liste résultat.

(réponse attendue : 5 à 10 lignes)

Corrigé.

```
public static ListeCommandes enlever(String c, ListeCommandes l){
    if(l == null)
        return null;
    if(l.commande.client.equals(c))
        return l.suivant;
    else{
        l.suivant = enlever(c, l.suivant);
        return l;
    }
}
```

c-iv) Écrire une méthode

```
public static Commande rechercher(String c, ListeCommandes l){}
```

qui retourne une commande (la première trouvée) au nom du client c, ou **null** s'il n'y en a pas.

(réponse attendue : 5 à 10 lignes)

Corrigé.

```
public static Commande rechercher(String c, ListeCommandes l){
    if(l == null)
        return null;
    if(l.commande.client.equals(c))
        return l.commande;
    else
        return rechercher(c, l.suivant);
}
```

c-v) Écrire dans la classe ListeCommandes une méthode

```
public static int chiffre(ListeCommandes lc){ ... }
```

qui calcule le montant total des commandes de la liste lc.

(réponse attendue : 5 à 10 lignes)

Corrigé.

```
public static int chiffre(ListeCommandes lc){
    int tot = 0;
    while(lc != null){
        tot += lc.commande.montant;
        lc = lc.suivant;
    }
    return tot;
}
```

c-vi) On rajoute un champ de type `ListeCommandes` dans la classe `Activite`. Écrire une méthode (dans `PointGlagla`) qui calcule le chiffre d'affaire final de la fête (la somme totale qu'elle a rapportée).

(réponse attendue : 5 à 10 lignes)

Corrigé. On rajoute un champ `lc` dans la classe `Activite`.

```
public static int chiffre() {
    int tot = 0;

    for(int i = 0; i < act.length; i++)
        if(act[i] != null)
            tot += ListeCommandes.chiffre(act[i].lc);
    return tot;
}
```

d) Dans un restaurant (ou un bar), un serveur (ou une serveuse) doit s'occuper de ses clients suivant des priorités. Par exemple, un client qui arrive doit être placé rapidement (priorité la plus élevée 0), etc., comme donné ci-dessous :

```
public int priorite() {
    if(this.detail.equals("nouveau"))
        return 0;
    else if(this.detail.equals("..."))
        ...
}
```

de la classe `Commande`, qui retourne une priorité (un entier entre 0 et $pmax - 1$, qui est une constante de la classe).

On donne enfin l'interface

```
public interface GestionClients {
    // arrivée d'un nouveau client nommé c : on lui associe une commande
    // fictive de montant 0 et dont le détail est "nouveau".
    public void nouveauClient(String c);

    // retrouve la (première) commande de priorité la plus élevée,
    // la retire de la structure de gestion et la retourne.
    public Commande unClientAServir();

    // insère une commande cmd dans la structure de gestion,
    // sa priorité est donnée par la fonction priorite().
    public void prendreCommande(Commande cmd);

    // on retire la commande en cours pour le client c
    public void departClient(String c);
}
```

d-i) Réalisez une implantation de cette interface, nommée `GestionClientsAvecListes`, qui utilisera un tableau `tlc` de listes de type `ListeCommandes`. Ce tableau aura une taille $pmax$ et chaque case du tableau sera une liste de sorte que `tlc[p]` désignera la liste des clients ayant priorité p .

(réponse attendue : 10 à 20 lignes)

Corrigé.

```
import tc.TC;

public class GestionClientsAvecListes implements GestionClients{

    public ListeCommandes[] tlc;
    public int pmax;

    public GestionClientsAvecListes(int pmax){
        this.pmax = pmax;
        this.tlc = new ListeCommandes[pmax];
    }

    public void nouveauClient(String c){
        Commande cmd = new Commande(c, 0, "nouveau");
        int p = cmd.priorite();
        this.tlc[p] = ListeCommandes.ajouter(this.tlc[p], cmd);
    }

    public Commande unClientAServir(){
        for(int p = 0; p < this.pmax; p++){
            if(this.tlc[p] != null){
                Commande cmd = this.tlc[p].commande;
                this.tlc[p] = this.tlc[p].suivant;
                return cmd;
            }
        }
        return null;
    }

    public void prendreCommande(Commande cmd){
        int pr = cmd.priorite();
        this.tlc[pr] = ListeCommandes.ajouter(this.tlc[pr], cmd);
    }

    public void departClient(String c){
        for(int p = 0; p < this.pmax; p++){
            Commande cmd = ListeCommandes.rechercher(c, this.tlc[p]);
            if(cmd != null){
                this.tlc[p] = ListeCommandes.enlever(c, this.tlc[p]);
                return;
            }
        }
    }
}
```

d-ii) Quel est le coût des différentes opérations, en fonction d'un majorant L sur la taille des listes et $P = pmax$ une borne sur le nombre de priorités.

Corrigé. Le coût de unClientAServir est $O(P)$; prendreCommande en $O(L)$; nouveauClient en $O(L)$; departClient en $O(P \times L)$.

d-iii) Existe-t-il une implantation plus rapide ?

Corrigé. On peut gérer une table (client, commande) qui permet de faire `departClient` en $O(L)$. Si l'on utilise du hachage pour les `tlc[p]`, on gagne des accès en $O(1)$, mais on perd l'ordre sur les clients, à priorité égale.