IFT2015 hiver 2012 — Devoir 2

Miklós Csűrös

1er février 2012

Le devoir vaut 30 points : vous avez le choix de travailler sur des problèmes théoriques (2.1 et 2.2), ou faire un TP de programmation (2.3). Règles :

- ⋆ Vous devez travailler sur les exercices théoriques 2.1 et 2.2 seul.
- ⋆ Vous avez le droit de travailler sur Exercice 2.3 en une équipe de deux (ou seul).
- ⋆ Vous pouvez travailler sur les deux parties pour des points boni : je vais calculer votre note selon l'algorithme suivant pour t points sur la partie théorique et p points sur la partie pratique :

```
DEVOIR2(t, p)                          // {0 ≤ t, p ≤ 30}
D1  if t > p then m ← t; b ← p
D2  else m ← p; b ← t
D3  if m ≥ 20 then note ← m + b/3
D4  else note ← m
```

(Dans d'autres mots, vous pouvez avoir jusqu'à 10 points de boni, si vous avez au moins 2/3 des points pratiques ou théoriques.)

Remise

	date limite	format
exercices théoriques (2.1 et 2.2)	8 février, 20 :15	1 fichier PDF
projet de programmation (2.3)	15 février, 20 :15	1 fichier JAR (`*.java` et `*.class`)

2.1 Parenthèses (10 points)

Le langage $\mathsf{Dyck}(s)$ est celui des expressions avec s types de parenthèses : $\underset{\smile}{j}$, $\overset{\frown}{j}$ pour $j = 1, 2 \ldots, s$ (comme les balises en XML). Une expression valide est dérivée par les règles

$$
\begin{aligned}
&E \mapsto \varepsilon && \textit{ε dénote l'expression vide}\\
&E \mapsto E\,E && \textit{plusieurs termes}\\
&E \mapsto \underset{\smile}{1}\,E\,\overset{\frown}{1} && \textit{parenthèses de type } 1\\
&E \mapsto \underset{\smile}{2}\,E\,\overset{\frown}{2} && \textit{parenthèses de type } 2\\
&\ldots\\
&E \mapsto \underset{\smile}{s}\,E\,\overset{\frown}{s} && \textit{parenthèses de type } s
\end{aligned}
$$

Donc $\underset{\smile}{1}\,\underset{\smile}{2}\,\overset{\frown}{2}\,\underset{\smile}{3}\,\overset{\frown}{3}\,\overset{\frown}{1}\,\underset{\smile}{1}\,\overset{\frown}{1}$ est valide, mais $\underset{\smile}{1}\,\underset{\smile}{2}\,\overset{\frown}{2}\,\underset{\smile}{3}\,\overset{\frown}{1}\,\overset{\frown}{3}$ ne l'est pas. Une *expression encodée* est un tableau $x[0..n-1]$ avec éléments $x[i] \in \{\pm 1, \pm 2, \ldots, \pm s\}$, où $+j$ est le code de parenthèse ouvrant $\underset{\smile}{j}$, et $-j$ est le code de parenthèse fermant $\overset{\frown}{j}$ pour $j = 1, \ldots, s$. Exemple : on encode $\underset{\smile}{1}\,\underset{\smile}{2}\,\overset{\frown}{2}\,\underset{\smile}{3}\,\overset{\frown}{3}\,\overset{\frown}{1}\,\underset{\smile}{1}\,\overset{\frown}{1}$ par le tableau $(1, 2, -2, 3, -3, -1, 1, -1)$.

► Donnez un algorithme pour vérifier si un encodage $x[0..n-1]$ est valide, en temps $O(n)$. Justifiez bien que votre algorithme est correct.

2.2 Asymptotiques (20 points)

a. (10 points) Démontrez, en utilisant la définition de Θ, que

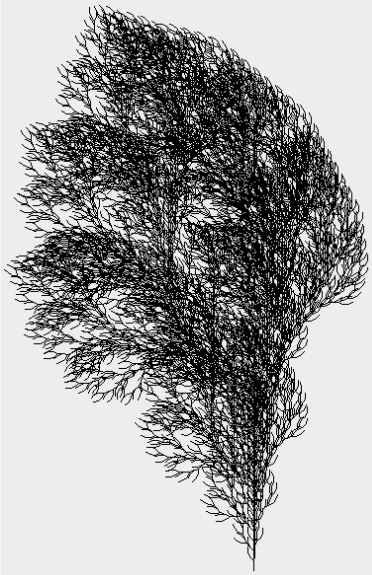
$$n^3 - 2n^2 + 2015 = \Theta(n^3) \quad \text{et que} \quad n^{2+1/\lg n} = \Theta(n^2).$$

b. (10 points) Démontrez que si la récurrence suivante vaut pour la fonction $T\colon \{1, 2 \ldots\} \mapsto [0, \infty)$, alors $T(n) = \Theta(n)$.

$$T(n) = \begin{cases} T(n-1) + O(1) & \text{si } n \bmod 2015 \neq 0\\ T(n-2015) + O(1) & \text{si } n = 2015k \text{ avec } k = 1, 2, 3, \ldots\\ O(1) & \text{si } n = 1. \end{cases}$$

Donnez votre réponse en notation $\Theta(\cdot)$ avec une preuve détaillée en utilisant la définition de $\Theta(\cdot)$.

2.3 Comment dessiner un arbre (30 points)



Vous avez à implanter un programme qui dessine des graphiques aléatoires à l'aide d'un *système de Lindenmayer* ou système L. En particulier, on veut produire des dessins qui ressemblent à des plantes. Le système L était inventé pour ce but : il permet de modéliser le développement de structures végétales.

Système de Lindenmayer

Un système L est une grammaire formelle qui définit la génération de chaînes de caractères sur un alphabet. Dans ce travail, on utilise l'alphabet `Ff+-[]X`. Le système est spécifié par la chaîne de départ ω et un ensemble de règles de réécriture dans la forme «caractère $\rightarrow$ chaîne». Exemple :

$$\omega = \texttt{F}$$
$$\texttt{F} \rightarrow \texttt{FF-F}$$

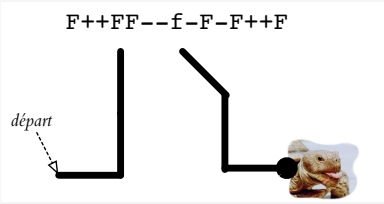
Dans un tel système, on génère des chaînes $S_0, S_1, S_2, \ldots$ en appliquant les règles de remplacement à tous les caractères de S_i en parallèle pour arriver à S_{i+1}. On commence par $S_0 = \omega$: `F` $\Rightarrow$ `FF-F` $\Rightarrow$ `FF-FFF-F-FF-F` $\Rightarrow \cdots$ S'il existe plusieurs règles avec le même côté gauche, on choisit une des règles de remplacement applicables au hasard (avec probabilité uniforme).

$$\omega = \texttt{F}$$
$$\texttt{F} \rightarrow \texttt{ff}$$
$$\texttt{F} \rightarrow \texttt{F-F}$$
$$\texttt{f} \rightarrow \texttt{F++}$$

$$\texttt{F} \underbrace{\Rightarrow}_{\text{avec proba } 1/2} \texttt{F-F} \underbrace{\Rightarrow}_{\text{avec proba } 1/4} \texttt{ff-F-F} \Rightarrow \cdots$$

Graphisme tortue

On interprète les chaînes générées par un système L comme des instructions pour une tortue graphique. La tortue possède un crayon et peut bouger en avant (par une distance D) en traçeant une ligne ou sans. La tortue peut aussi tourner (par une angle δ) dans ou contre le sens d'aiguille. L'état de la tortue comprend sa position (x, y) ainsi que l'angle θ de son nez par rapport à la ligne horizontale.



Un dessin est spécifié par une chaîne de caractères du système L, en interprétant les caractères un-à-un selon le tableau ci-dessous (l'exemple à la gauche utilise $\delta = 45^\circ$ pour tourner)

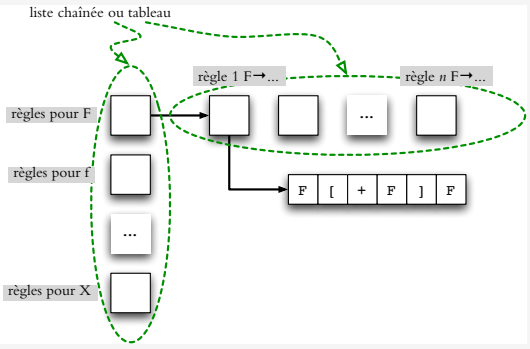
- `F` Avance la tortue par D, en dessinant une ligne entre la position de départ et celle de l'arrivée. L'état de la tortue change de (x, y, θ) à $(x + D\cos\theta, y + D\sin\theta, \theta)$ où D est un paramètre global du dessin spécifiant l'échelle.
- `f` Avance la tortue par D, mais sans dessin. L'état de la tortue change de (x, y, θ) à $(x + D\cos\theta, y + D\sin\theta, \theta)$.
- `+` Tourne la tête de la tortue. L'état de la tortue change de (x, y, θ) à $(x, y, \theta + \delta)$ où δ est un paramètre global du dessin.
- `-` Tourne la tête de la tortue. L'état de la tortue change de (x, y, θ) à $(x, y, \theta - \delta)$ où δ est un paramètre global du dessin.
- `[` Empile l'état courant de la tortue sur la pile d'états sauvegardés. L'état de la tortue ne change pas.
- `]` Dépile l'état de la tortue et fait l'affectation de l'état courant. L'état de la tortue donc change de (x, y, θ) à (x', y', θ') où (x', y', θ') est l'état le plus récemment sauvegardé par `[`.
- `X` Aucun effet sur la tortue, ignoré dans le dessin.

Implantation

Vous devez implanter un programme Java qui prend un système L, fait la dérivation en n itérations, et dessine le résultat sur l'écran. Pour le dessin, créez une sous-classe de `javax.swing.JPanel` (ou un autre composant graphique que vous préférez) ; D et δ sont spécifiés lors de l'instanciation. Le dessin est spécifié par l'état initiel de la tortue (x_0, y_0, θ_0) et la séquence de commandes s (dérivé par le système L). Dans l'exemple ici, on utilise les variables privées `x0`, `y0`, ... pour stocker le dessin — `paintComponent` en aura besoin.

```
private double x0; // départ de la tortue
private double y0; // départ de la tortue
private double angle0; // départ de la tortue
private String drawing; // commandes pour le dessin
/** Spécifie le dessin dans l'alphabet 'Ff+-[]X' */
public void dessin(double x0, double y0, double a0, String s)
{
    this.x0 = x0;
    this.y0 = y0;
    this.angle0 = a0;
    this.drawing = s;
    repaint();
}
```

Les commandes (stockées par la variable `drawing`) sont executées dans la méthode `paintComponent`, où vous devez utiliser une pile (classe `java.util.Stack` ou votre propre code) pour sauvegarder et retrouver les états de la tortue ('`[`' et '`]`'). Dans le système graphique de Java, la tortue devrait partir en bas du panneau, au milieu, avec un angle de 270°, vers le haut.



Pour stocker les règles du système, je recommande une structure de listes. Une itération de remplacement peut se faire (1) en parcourant la chaîne S et construisant la chaîne de remplacements S', ou bien (2) en considérant S comme file FIFO.

Voici le remplacement par file FIFO : on utilise le caractère spécial `$` pour dénoter la fin-de-la-chaîne.

```
// (remplacement de caractères dans la queue S)
1  S.enqueue($)
2  boucler
3      c ← S.dequeue()
4      si c = $ alors sortir de la boucle
5      si aucune règle n'applique à c alors S.enqueue(c)
6      sinon
7          choisir une règle c → x_1 x_2 ··· x_k au hasard
8          pour j ← 1, ..., k faire S.enqueue(x_j)
```

L'exécutable est lancé comme

`java plante.Dessin` ⟨arguments optionnels⟩ n ω **règle**$_1$ **règle**$_2$ ...

Exemple :

```
% java -cp build/classes plante.Dessin \
   -D 10 -y 600 -delta 24 5 F \
   'F:F[+F]F[-F]F' 'F:F[+F]FF' 'F:F[-F]F'
```

Arguments obligatoires

n entier non-négatif, c'est le nombre d'itérations dans la dérviation

ω chaîne de départ dans le système

règle$_i$ est une chaîne sans espace où le côté gauche et droit sont séparés par ' `:`' (dénotant donc le symbole $\rightarrow$)

Arguments optionnels

`-D double` spécifie l'échelle du dessin

`-delta double` spécifie l'angle d'unité pour la tournée de la tortue en degrés. Attention : `Math.sin` et `Math.cos` prennent leurs arguments en radians — faites la conversion par `Math.toRadians` si nécessaire.

`-x double` coordonnée X de la position initiale de la tortue

`-y double` coordonnée Y de la position initiale de la tortue

`-a double` angle initial de la tortue en degrés

Pour les structures de données (pile de la tortue, listes pour le système), vous pouvez utiliser les classes standardes de `Java` (comme `java.util.LinkedList`), ou fournir votre propre code. Mettez vos classes dans un package appellé `plante`, et soumettez le code source avec les fichiers de classe dans un archive JAR. (Votre soumission sera testée par `java -cp Plante.jar plante.Dessin ...` et le code source sera examiné aussi.)

Exemples. Vous povez crér vos propres sytèmes de règles pour trouver des dessins intéressants. Quelques exemples inspirants :

Nom	Paramètres	ω	Règles
Flocon	$n = 4, \delta = 90^\circ$	`-F`	`F` $\rightarrow$ `F+F-F-F+F`
Ilots	$n = 2, \delta = 90^\circ$	`F+F+F+F`	`F` $\rightarrow$ `F+f-FF+F+FF+Ff+FF-f+FF-F-FF-Ff-FFF`
			`f` $\rightarrow$ `ffffff`
Plante	$n = 5, \delta = 25.7^\circ$	`F`	`F` $\rightarrow$ `F[+F]F[-F]F`
Buisson	$n = 5, \delta = 22.5^\circ$	`F`	`F` $\rightarrow$ `FF-[-F+F+F]+[+F-F-F]`
Plante	$n = 5, \delta = 22.5^\circ$	`F`	`F` $\rightarrow$ `F[+F]F[-F]F`
			`F` $\rightarrow$ `F[+F]F`
			`F` $\rightarrow$ `F[-F]F`