



Les objectifs de cette séance de TD / TP sont :

- Utiliser la programmation récursive
- Conception d'algorithmes Qsort et recherche dichotomique
- LSystem (Grammaire, Récursivité)

TD

1.Tri d'un tableau par méthode dite « Quick Sort »

Analyse :

Soit une liste L dans le désordre.

Soit P le premier élément de la liste, on le nomme Pivot

- La première étape du calcul consiste à déterminer LA LISTE DE TOUTES LES VALEURS DE LA LISTE STRICTEMENT INFÉRIEURES au pivot.
- La seconde étape consiste à déterminer LA LISTE DE TOUTES LES VALEURS DE LA LISTE STRICTEMENT SUPÉRIEURES au pivot.
- La troisième étape consiste à construire (concaténer) une liste composée
 - a) de la liste TRIÉE des valeurs inférieures au pivot
 - b) du pivot
 - c) de la liste TRIÉE des valeurs supérieures au pivot

Votre travail consiste à :

écrire une méthode : `public abstract Liste inferieurs(int s);` (Etape 1)

écrire une méthode : `public abstract Liste superieurs(int s);` (Etape 2)

écrire une méthode : `public abstract Liste QS();`

rmq : Vous utiliserez la méthode Concat pour assembler deux listes.

En plus,

écrire une méthode : `public abstract void listeEnTab(ArrayList T)`

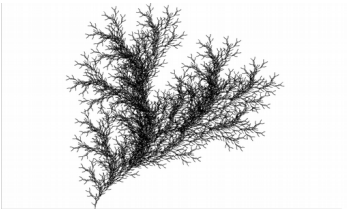
écrire une méthode : `public ListeCons(int nbElements)`

Construit une liste aléatoires de nbElements, utiliser Alea(n1 , n2)

2.Présentation des systèmes de Lindenmayer

Utilisation d'une grammaire, de la récursivité, d'une pile de contextes.

"Quelque part, caché dans les cellules des plantes, un brin d'ADN décrit leur développement au fil des semaines ou des années. Au lieu d'analyser le code génétique des plantes, Aristid Lindenmayer a tenté de deviner par lui-même leurs règles de croissance, puis de les vérifier. Bien qu'il ait été biologiste, il est principalement reconnu pour son travail mathématique : les L-Systèmes. Cette grammaire formelle permet de modéliser la croissance des végétaux."



Etape 1 : Grammaire formelle décrivant la croissance (code ADN)

- La grammaire formelle sera constituée :
- D'un alphabet $A=\{ a, g, d \}$
 - D'un axiome point de départ de l'expression $U_0 = a$
 - De règles $a \rightarrow agaddaga ; g \rightarrow g ; d \rightarrow d$

Etape 2 : Dérivation de l'expression

Il est possible de construire une suite $(U_n)_{n \in \mathbb{N}}$ de mots à partir de l'axiome et des règles de dérivation.

n	U_n
0	a
1	agaddaga
2	agaddaga g agaddaga dd agaddaga g agaddaga

Cette succession de dérivations décrit symboliquement l'évolution des formes ou des structures.

Etape 3 : interprétation graphique

Si l'on interprète maintenant les symboles d'un mot et que l'on trace avec un Logo, nous allons obtenir des dessins intéressants.

Interprétation des symboles

- a : avancer et tracer un trait ;
- g : tourner à gauche de 60° ;
- d : tourner à droite de 60°



FIGURE 2 – Interprétation du mot $u_0 = a$



FIGURE 3 – Interprétation du mot $u_1 = agaddaga$

L'angle de départ du dessin est 0° donc nous commencerons à tracer en allant vers la droite de l'écran.

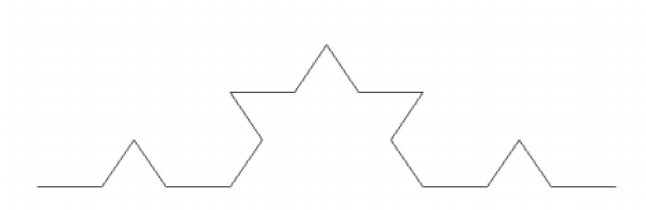


FIGURE 4 – Interprétation du mot $u_2 = agaddagagagaddagaddagaddagagagaddaga$



FIGURE 5 – Interprétation du mot u_3

a.Exercice

Soit la grammaire suivante :

- $A = \{ X, +, - \}$
- Axiome $U_0 = X$
- Règles $X \rightarrow X-X+X+X-X$
 $\quad \quad \quad + \rightarrow +$
 $\quad \quad \quad - \rightarrow -$

Interprétation : X : avance d'un segment ; $+$ tourne à gauche de 90° ; $-$ tourne à droite de 90°

Dérivez une fois l'expression et interprétez U_1 .

Dérivez une nouvelle fois l'expression. Tracez U_2 ...

3. Algorithmes récursifs : QS et Dichotomie

- Prenez Liste.java et implantez inferieurs(), superieurs(), QS().

Dans un main(), concevez une liste aléatoire avec le constructeur ListeCons(n) qui génère une liste aléatoire. Testez QS()

- Ajoutez la méthode `public void listeEnTab(ArrayList T)`. Le tableau T sera rempli avec les valeurs.
- Testez la transformation d'une liste triée en tableau.

- Ouvrez à présent le fichier Qstore.java qui va nous aider à tester la dichotomie sur la base d'un tableau trié.

Cette classe va nous permettre de compter le nombre d'appels récursifs de Dicho afin d'évaluer le coup de la complexité.

Rappel : la méthode itérative classique trouve un élément en $O(n/2)$ en moyenne, au pire $O(n)$ si $n = 2048...$

Combien pour la dichotomie ?

- Concevez la méthode dico (voir principe en cours)
public boolean dico(int n , int iDebut, int iFin)

Elle recherche n dans le tableau T trié entre iDebut et iFin

Utilisez la variable d'instance compteurDicho pour compter les appels, affichez dans dico() les bornes de recherche et compteurDicho.

- Testez la dichotomie

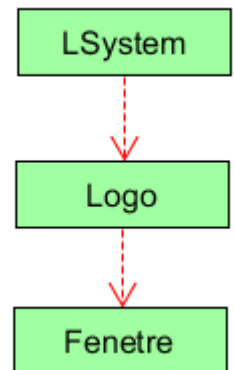
-Dans exec() utilisez le test précédent sur les listes pour générer une liste aléatoire de 2200 éléments.

-Triez la liste

-Lancez la dichotomie

4. Mise en œuvre des LSystem en TP

- Classe LSystem : Modélise la grammaire, permet de dériver les expressions, permet d'interpréter l'expression et faire appel à Logo.
- Classe Logo : permet grâce à des commandes simples de tracer des segments dans un repère.
- Classe Fenetre : trace dans une fenêtre graphique les segments de droites.



a. La Classe Logo

Constructors

Constructor and Description

Logo(int initX, int initY, int initAngle, Color initColor, boolean posStylo, int pas)
Constructeur principal

Method Summary

Methods

Modifier and Type	Method and Description
void	av() Avance de "pas" pixels puis trace le segment dans la fenetre
int	getAngle() getter
Color	getCouleur() getter
boolean	getEtatStylo() donne l'etat du stylo
int	getPas() getter
int	getX() getter
int	getY() getter
void	memo() Mémorise le contexte actuel du Logo : position, angle, couleur, etatStylo, pas
void	recupMemo() Recupère un contexte mémorisé de Logo puis re-initialise les variables du Logo avec les données
void	reduction() Réduit le pas d'un COEFDIMINUTION
void	rotD(int a) Rotation à droite sens horaire de a degrés
void	rotG(int a) Rotation à gauche antihoraire de a degrés
void	setAngle(int a) Change l'angle de rotation
void	setPas(int pas) Change la longueur des segments
void	setX(int x) Change la position x
void	setY(int y) Change la position y
void	StyloBas() stylo en bas
void	StyloHaut() stylo en haut
String	toString() getter

Changement de contextes

Comme vous le voyez dans la documentation de la classe Logo, deux méthodes permettent de mémoriser et restituer des contextes de travail. Regardez la déclaration de la Pile de mémorisation.

➡ Testez les instructions suivantes :

```
Logo l = new Logo( 500, 100, 0, Color.red, Logo.PEN_DOWN , 10);
```

```
l.rotG(90); /* tronc*/  
l.av();  
l.memo(); /* branche1*/  
  l.rotG(45);  
  l.av();  
l.recupMemo(); /* branche2*/  
l.rotD(50);  
l.av();
```

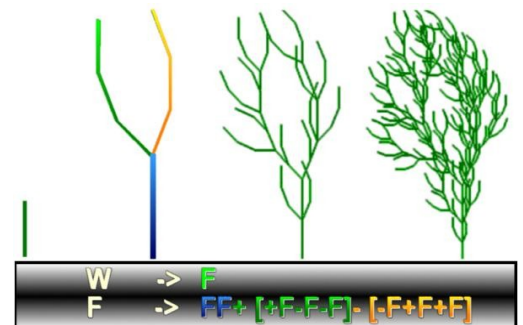
Compris le changement de contexte ?

b.La classe LSystem

Maintenant que nous savons tracer à l'écran des formes basiques, nous allons nous intéresser au LSystem.

La classe LSystem va nous permettre :

- d'implanter la grammaire,
- de procéder aux dérivations utilisant les règles données,
- interpréter l'expression résultante.



➡ Implantation de la grammaire

- Ouvrez le fichier LSystem.java mettez en place une grammaire à l'aide d'un HashMap.

Soit la grammaire suivante :

- $A = \{ X, Y, +, - \}$
- Axiome $U0 = X$
- Règles
 $X \rightarrow X+Y++Y-X - -XX-Y+$
 $Y \rightarrow -X+YY++Y+X--X-Y$
 $+ \rightarrow +$
 $- \rightarrow -$

Interprétation : X : avancer Y : avancer + : tourner à gauche de 60° - : tourner à droite de 60°

L'alphabet peut changer d'une description a une autre. Ainsi, il faut pouvoir associer des éléments entre eux. Ainsi, les règles pourront être stockées dans un tableau associatif ou dictionnaire de type *HashMap*

L'axiome sera stocké dans une variable simple.

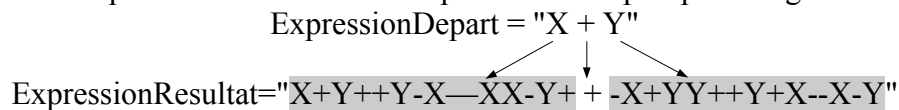
Le résultat final de même sera dans une variable simple dont vous définirez le type.

➡ Dérivation d'expression

- Faites `derivation()` qui prend l'expression courante et fabrique la nouvelle.
- Testez soigneusement la dérivation d'expression.

La dérivation consiste à remplacer un caractère de l'expression de départ par la règle associée.

ExpressionDepart = "X + Y"
ExpressionResultat = "X+Y++Y-X—XX-Y+ + -X+YY++Y+X--X-Y"



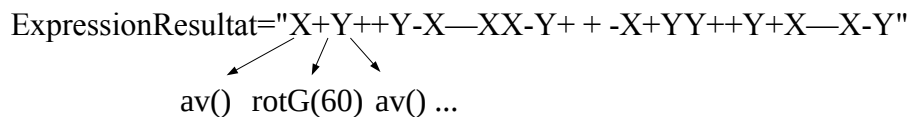
On lit l'expression de départ caractère par caractères, on cumule dans l'expressionResultat, les résultats de l'application des règles définies.

➡ Interprétation

- Mettez en place l'interprétation de l'expression.

Pour interpréter les éléments de l'alphabet $V = \{ X, Y, +, - \}$ que vous aller trouver dans l'expression, il suffit d'exécuter la commande du Logo associée selon le caractère que vous pointez.

ExpressionResultat = "X+Y++Y-X—XX-Y+ + -X+YY++Y+X—X-Y"
av() rotG(60) av() ...



Changez de grammaire

Exemple1 :

$U0 = 0$
 $0 \rightarrow 1[-0] + 0$
 $1 \rightarrow 11$

0 ou 1 : avancer

[: mémorisation

] : récupération Mémoire

+ : rotation gauche de 45°

- : rotation droite de 45°

Exemple2 :

$U0 = X$
 $X \rightarrow F-[[X]+X]+F[+FX]-X$
 $F \rightarrow FF$

F :avancer

X : rien

[: mémorisation

] : recuperation Mémoire

+ : rotation gauche en 20 et 30 °

- : rotation droite en 20 et 30 °

Trouvez sur le Web d'autres exemples.