Remise à niveau en informatique

Contents

CHAPTER 1

Organisation

Ce module sera composé d'un cours d'introduction aux concepts fondamentaux d'informatique (structures de données, complexité, calculabilité, ...) et de TP d'applications. Vous serez évalués sur les TPs et sur un examen théorique final.

CHAPTER 2

Jeudi 07/10/2003: TP1, structures de données

À rendre pour le 16/10/2003.
Sujet: `http://lapcs.univ-lyon1.fr/~nthiery/DESS/Notes/Exemples/RN-TP1-Nicolas.Thiery-1.0.tar.gz`
Correction: `http://lapcs.univ-lyon1.fr/~nthiery/DESS/Notes/Exemples/RN-TP1-Nicolas.Thiery-1.0-Correction.tar.gz`

CHAPTER 3

Jeudi 16/10/2003: TP2, Tris et complexité

À rendre pour le 23/10/2003.

EXERCICE 1. Dans le cadre de l'utilisation de l'outil informatique, vous avez déjà implanté le tri à bulle. Complétez la class `Sort` avec des fonctions pour le tri rapide, le tri fusion et le tri par arbre binaire de recherche. Pour ce dernier tri, on suggère de définir une nouvelle classe `BinarySearchTree` qui hérite de `BinaryTree`, avec une méthode `add` pour insérer un nouvel élément.

Ceux qui voudront aller plus loin pourront implanter une classe `BalancedBinarySearchTree`, héritant de `BinaryTree`, dans laquelle les arbres ont des branches dont les longueurs diffèrent d'au plus 1. On pourra par exemple utiliser l'astuce des noeuds rouge/noir (red-black tree).

EXERCICE. Pour chacun de ces tris, estimez le nombre d'opérations élémentaires à l'aide d'un compteur, et faites quelques statistiques sur des échantillons aléatoires. Déduisez-en une évaluation de leur complexité en moyenne, au mieux et au pire.

3.1. Quelques références

- Un support de cours sur les tris `http://dept-info.labri.u-bordeaux.fr/~lachaud/IUT/ASD-Prog-E1-2000/planning-prof.html`
- Un support de cours sur les tris séquentiels `http://deptinfo.unice.fr/~fedou/ENSEIGNEMENT/ASD/TRIS/index.html`
- Une fiche de TP sur les tris `http://www.lri.fr/~denise/M2Spec/97-98.1/TDSpec6.ps`
- Démonstration de bubble sort et quicksort `http://jade.lim.univ-mrs.fr/~vancan/mait/demo/SortDemo/example1.html`

CHAPTER 4

Mardi 23/10/2003: TP3, Polynômes et complexité

À rendre pour le 30/10/2003. Non noté.

Exercice. Compléter les méthodes `add`, `multiply` et `multiplyKaratsuba` dans la classe `Polynom` fournie `http://lapcs.univ-lyon1.fr/~nthiery/DESS/Notes/Exemples/RN-TP3-Nicolas.Thiery-1.0.tar.gz`. Pour ceux qui veulent aller plus loin: compléter aussi la méthode `multiplyFFT`. Tracer, pour des polynômes aléatoires de degré croissant, la courbe du temps nécessaire pour les différentes opérations de multiplication en fonction du degré des polynômes en entrée.

Correction: `http://lapcs.univ-lyon1.fr/~nthiery/DESS/Notes/Exemples/RN-TP3-Nicolas.Thiery-1.0-Correction.tar.gz`

CHAPTER 5

Jeudi 30/10/2003, TP 4: Automates et langages rationnels

À rendre pour le 6 novembre.

Exercice 2. Écrire des programmes (par exemple en perl ou en java) utilisant des expressions régulières appropriées pour:

- Extraire la liste des utilisateurs depuis le fichier `/etc/passwd` contenant la base de donnée des utilisateurs des machines.
- Prendre un fichier de données contenant dans chaque ligne 3 nombres séparés par des tabulations, et inverser les deux premières colones.
- Étant donné un mot sur la ligne de commande, renvoyer tous les mots anglais le contenant comme sous-mot (on considère que `car` est un sous-mot de `spectator`). On pourra utiliser le fichier `/usr/share/dict/words`.
- Étant donnés deux mots sur la ligne de commande, chercher le nombre d'occurences de ces mots proches l'un de l'autre dans un texte. On pourra se contenter, pour simplifier, de ne rechercher de tels mots proches que lorsqu'ils sont sur la même ligne.

Si vous optez pour java, vous pouvez utiliser la bibliothèque `gnu.regexp`. Un mini d'exemple d'utilisation est fourni:
`http://lapcs.univ-lyon1.fr/~nthiery/DESS/Notes/Exemples/RN-TP4-Nicolas.Thiery-1.0.tar.gz`
Pour la documentation complète de cette bibliothèque, voir:
`file:/usr/java/docs/gnu.regexp/`

Le reste des exercices est facultatif.

Exercice. Soit $\Sigma = \{a, b, c\}$ un alphabet. Les langages suivants sur Σ sont-ils rationnels?

(1) $L_1 = \{aba, bba, acab\}$
(2) $L_2 = a^*$
(3) $L_3 = a^*b^*$
(4) $L_4 = \{a^n b^n, n \in \mathbb{N}\}$
(5) $L_5 = \{(ab)^n, n \in \mathbb{N}\}$.

Exercice. Soit $\Sigma = \{a, b\}$ un alphabet. Construire les automates déterministe finis qui reconnaissent respectivement:

(1) Le langage $L_1 := \{aa, ab, bbaab\}$;
(2) Le langage $L_2 := a^*$;
(3) Le langage $L_3 := a^+$;
(4) Le langage $L_4 := a^*b^*$;
(5) L'ensemble des mots se terminant par b;
(6) L'ensemble des mots contenant au moins un b;

(7) L'ensemble des mots contenant une sequence de 4 b consécutifs;
(8) L'ensemble des mots dont le nombre d'occurrence de b est divisible par 3;
(9) $L_9 := \{a, b\}^*\{a\}\{a, b\}^2$;
(10) $L_{10} := (aab + aa + aabb)^*$.

EXERCICE. Implantation des automates sur machine.

(1) Concevoir une structure de donnée appropriée pour stocker un automate déterministe fini.
(2) Implanter, dans le langage de votre choix, une fonction prenant en entrée un automate et un mot, et vérifiant que le mot est bien reconnu par l'automate.
(3) Utiliser cette fonction pour écrire des programmes reconnaissant les langages de l'exercice précédent.

EXERCICE. Implanter une fonction qui étant donné un entier n construit un automate reconnaissant le langage $L := ab, aabb, aaabbb, \dots, a^n b^n$.

EXERCICE. Pour ceux qui n'ont peur de rien!

(1) Implantez une fonction prenant un automate non déterministe, et renvoyant un automate déterministe reconnaissant le même langage.
(2) Implantez une fonction prenant en entrée un automate fini déterministe pour L_1, et retournant un automate fini déterministe pour L_1^*.
(3) Implantez une fonction prenant en entrée des automates finis déterministes pour L_1 et L_2, et retournant un automate fini déterministe pour $L_1.L_2$.
(4) Implantez une fonction prenant en entrée des automates finis déterministes pour L_1 et L_2, et retournant un automate fini déterministe pour $L_1 \cup L_2$.

5.0.1. Quelques références.

- TD: Dragons, trolls et automates finis `http://pauillac.inria.fr/ups/tp/tp6.ps`
- Le Dragon Book `http://www.science.uva.nl/~mes/jargon/d/dragonbook.html`
- Langages et compilation, resume de cours `http://pauillac.inria.fr/algo/banderier/P13/html/`

CHAPTER 6

Jeudi 30/10/2003, TP 4: Grammaires

À rendre pour le 13 novembre. Non noté.

L'objectif de ce TP est d'apprendre à utiliser les grammaires en pratique, par exemple avec javacc. Un mini exemple d'utilisation de javacc est fourni:

`http://lapcs.univ-lyon1.fr/~nthiery/DESS/Notes/Exemples/RN-TP5-Nicolas.Thiery-1.0.tar.gz`

Pour la doc et des exemples plus complets d'utilisation de javacc, voir:

`file:/usr/java/javacc/`

Exercice 3. Modifier l'exemple fourni pour qu'il reconnaisse n'importe quel jeu de parenthèses, accolades, comme par exemple: [[{}[([])]]{()}]).

Écrire une application Calc qui calcule la valeur d'une expression comme '(3 + 4) * (3 + 4*(1+2))'.

En réutilisant les classes ExpressionTree & autres du premier TP, écrire une application InfixToPolish qui transforme en notation infixe une expression comme '(3 + 4) * (3 + 4*(1+2))'.

Exercice 4. Écrire une classe qui permet de lire des fichiers au format Hexier.