ACT - TP1 - Ordres de grandeur asymptotiques

Objectif : L'objectif de ce TP est d'illustrer les notions d'ordres de grandeur asymptotiques (O, Ω, Θ) en traçant les courbes des différentes fonctions de référence et en les comparant entre elles et avec des courbes expérimentales obtenues sur certains algorithmes.

Attention ! Ceci n'est pas un TP de programmation ! Ne vous attendez pas à produire des centaines de ligne de code pour construire des modèles très élaborés. Le code développé ici a pour seul but la production de graphiques d'analyse. Il est bien plus important de comprendre les concepts que de savoir refaire le code de ce TP.

1 - Les classiques

En cours et en TD nous avons utilisé des fonctions dites "de référence" pour classer nos algorithmes par ordre de grandeur. La première partie de ce TP a pour but de montrer graphiquement les différences entre ces diverses fonctions.

1.1 - Méthodologie

Téléchargez l'archive contenant le code exemple depuis le semainier. Dans le dossier décompressé vous devriez avoir deux dossiers ; un nommé *data* contenant cet énoncé et un second nommé *src* et contenant les fichiers sources java du programme. Pour cette première partie nous n'aurons besoin que des classes présentes dans les packages *comparaisons* et *io*.

1.1.1 - Générer des données

Nous allons illustrer la suite du TP par l'exemple déjà codé. Dans cet exemple nous souhaitons comparer l'allure des courbes des fonctions *f(n)=n* et *g(n)=log(n)*. Pour effectuer cela, compilez le programme de la manière que vous appréciez le plus (ligne de commande, via un IDE, en faisant appel au voisin, …). La classe contenant le main s'appelle *ComparaisonRefs*. Lisez le *main* pour comprendre comment les résultats sont générés. Exécutez le programme pour obtenir le fichier "comparaisons_refs.txt" dans le dossier *data*. Les données ont été générées sur 3 colonnes. La première donne la valeur de x, seconde f(n) et la troisième de g(n)

1.1.2 - Visualiser des résultats

Lire des chiffres c'est bien mais voir des courbes c'est mieux ! Nous allons utiliser gnuplot, un logiciel de traçage de courbes, pour visualiser nos résultats. Allez dans un terminal puis rendez vous dans le dossier data du projet. Tapez gnuplot pour vous retrouver dans l'invite de commande du logiciel. Puis tapez la commande suivante pour observer le tracé de la fonction *f(n)*

gnuplot> plot "comparaisons_refs.txt" using 1:2 title "f(n)=n" with lines

Nous venons de tracer les données enregistrées dans le fichier *comparaisons_refs.txt* en utilisant la colonne 1 en abscisse et la colonne 2 en ordonné. Nous avons également joint les points entre eux en utilisant les mots *with lines*.

Traçons maintenant les deux courbes de *f(n)* et *g(n)* pour les comparer

gnuplot> plot "comparaisons_refs.txt" using 1:2 title “f(n)=n” with lines, "comparaisons_refs.txt" using 1:3 title “g(n)=log(n)” with lines

Vous aurez compris, par analogie avec le premier test, que la commande trace deux courbes sur la même image en prenant respectivement les colonnes 1,2 et 1,3. Pour mieux voir vous pouvez vous déplacer dans l’image de manière dynamique.

Pour aller voir précisément un morceau des courbes vous pouvez définir les plages de valeurs pour x et pour y comme suit.

gnuplot> set xrange [0:30]
gnuplot> set yrange [0:30]
gnuplot> plot "comparaisons_refs.txt" using 1:2 title “f(n)=n” with lines, "comparaisons_refs.txt" using 1:3 title “g(n)=log(n)” with lines

Que peut-on déduire de ces tracés ?

1.1.3 - Automatisation et enregistrement des résultats

Pour ne pas avoir à réécrire les commandes a chaque fois que nous relançons gnuplot et pouvoir reproduire aisément nos tracés, nous allons créer des scripts de lancement. Créez un fichier “premier_script.plt” dans le répertoire data. Recopiez les lignes précédentes à l’intérieur et enregistrez le fichier. Dans le terminal tapez la commande :

$> gnuplot premier_script.plt

Vous devriez voir l’image s’afficher puis disparaître immédiatement. En effet, lorsque le script arrive à son terme tout ce qui a été affiché est fermé simultanément. Rajouter la commande *pause -1* à la fin du script puis recommencez. Si vous souhaitez obtenir votre tracé sur un fichier png plutôt qu’afficher dans une fenêtre gnuplot, vous pouvez vous renseigner sur la commande *set terminal png* de gnuplot.

1.2 - Comparaisons des fonctions références

Maintenant que nous avons vu comment tracer les courbes nous allons pouvoir comparer les fonctions de références classique.

1.2.1 - En respectant l’architecture du programme déjà écrit, ajouter dans la classe *FonctionsReference* des méthodes pour obtenir les valeurs de *n*log(n)*, *exp(n)*, *k^n*, *n!* et *n^n*.

1.2.2 - Modifiez la classe *ComparaisonRefs* pour générer un fichier de données contenant toutes les données pour les 5 fonctions de références.

1.2.3 - Créez un script gnuplot permettant de tracer toutes les courbes pour les fonctions de référence sur un même graphique.

1.2.4 - Que pouvez-vous dire des tracés ? Attendait-on ce résultat ?

1.3 - Limites graphiques

Un tracé de courbes avec des très grandes valeurs pour n ne remplacera **jamais** une preuve de limite en +∞ mais il peut donner quelques indices sur les ordres de grandeur. Nous allons ici étudier les limites du rapport de deux fonctions pour déduire les majorations des unes par les autres.

1.3.1 - n^{ε} vs *log(n)*

Dans la classe *ComparaisonRefs*, écrivez une fonction permettant d'obtenir $log(n) \div n^{\varepsilon}$ puis tracez les courbes (en créant un script pour cela) sur un même graphique pour les valeurs de ε 1, 0.5, 0.2, 0.1. Que peut-on dire du rapport entre les deux fonctions ?

1.3.2 - 2^n vs n^k

Mêmes questions pour le rapport $n^k \div 2^n$ pour des valeurs de k□1 que vous choisirez.

2 - Mesures sur structures de données

L'objectif de cette partie est d'illustrer la mesure des complexités algorithmiques par la comparaison de méthodes sur des structures de données déjà implémentées dans Java. Dans le package *datastructures* vous trouverez deux classes nommées *Tableau* et *Liste* qui implémentent toutes deux l'interface *DataStructure* contenant les fonctions que nous allons comparer.

2.0 - Ayant une fonction f implémentée, comment pourrions-nous nous y prendre pour mesurer expérimentalement sa complexité ?

2.1 - L'aléa() des extrêmes

Pour ne pas avoir d'interférences avec la JVM dans la mesure des temps d'exécution il faut lui passer l'argument -Xint afin de désactiver la compilation à la volée. Pour mesurer le temps vous pourrez utiliser la fonction *nanoTime()* de la classe *System*.

2.1.1 - En lisant la fonction *aléa()* implémentée dans la classe *Tableau*, que pouvez-vous dire sur la classe algorithmique de la fonction ?

2.1.2 - Complétez la fonction *mesureAlea* de la classe *Mesures*, exécutez le main présent dans la classe *ComparaisonStructures* puis tracez la courbe sur les données engendrées. La courbe a-t-elle la forme attendue ?

2.1.3 - Ré-exécutez le main puis tracez à nouveau la courbe. La courbe est-elle toujours strictement la même ? Pourquoi ?

2.1.4 - Comment remédier au problème soulevé en 2.1.3 ? Modifiez le main en conséquence.

2.1.5 - Pour bien voir les différences entre les meilleurs et pires exécutions tracez les courbes du minimum, maximum et de la moyenne obtenus sur un très grand nombre de lancements.

2.1.6 - Ces courbes paraissent-elles correspondre à votre prédiction de complexité de la question 2.1.1 ?

2.2 - Un minimum() de performances

Dans la classe *DataStructure* sont présentes deux fonctions permettant d'obtenir l'élément minimal stoqué dans la structure de données. Les deux fonctions sont appelées *minimumSimple()* et *minimumTri()*. L'objectif de cette partie est d'essayer de voir expérimentalement laquelle des deux méthodes doit être employée. Comme dans la partie 1 vous effectuerez les tests sur la classe Tableau.

2.2.1 - Sur le même schéma que la fonction de mesure de *alea()*, écrivez une fonction de mesure pour chacune des fonctions minimum.

2.2.2 - Tracez sur un même graphique les extremums et la moyenne pour les temps d'exécution des deux fonctions. Vous prendrez soin de choisir une échelle pertinente pour les observations.

2.2.3 - Que pouvez-vous en conclure ? Au regard de cette conclusion, vous vous engagez à ne plus jamais utiliser la moins bonne des deux méthodes !

2.3 - Les structures de base pour le meilleur et surtout pour le pire

L'objectif de cette partie sera de comparer les performances en temps de calcul sur les fonctions implémentées pour les deux structures de données étudiée (*Tableau* et *Liste*)

2.3.1 - Pour chacune des fonctions présentes dans *DataStructure* créez un fichier de données pour comparer les performances entre les Tableaux et les Listes (extremums et moyenne pour les marges d'erreur).

2.3.2 - Les courbes correspondent-elles toutes aux ordres de grandeur attendus pour ces fonctions ?

2.3.3 - Peut-on en déduire qu'une des structures de données est plus efficace que l'autre ? Pourquoi ?