Algorithmique Avancée **Devoir de Programmation**

---

*Devoir à faire en **binôme**. Soutenance en séance de TD/TME 10 (entre le 18/12 et 20/12).*
Rapport et Code Source à rendre (par mail à vos enseignants) au plus tard le 21/12 à 23h59.
Langage de programmation libre.

1 Présentation

Le but du problème consiste à visualiser graphiquement les temps d'exécution sur des données réelles des algorithmes et structures de données que nous avons introduits dans les chapitres 1, 2 et 3 du module.
Il est attendu un soin particulier concernant la réflexion et la mise en place concernant les expérimentations dans ce devoir.

1.1 Échauffement

Dans tout le devoir, les clés sont des entiers codés sur 128 bits : on peut, par exemple, utiliser *4 entiers non signés de 32 bits* pour les représenter.

Question 1.1 Étant donné 2 clés 128 bits, écrire un prédicat `inf` (une fonction renvoyant un booléen) permettant de déterminer si `cle1` est strictement inférieure à `cle2`.

Question 1.2 Étant donné 2 clés 128 bits, écrire un prédicat `eg` permettant de déterminer si `cle1` et `cle2` sont égales.

Par la suite, un *jeu de données aléatoires* est disponible à l'adresse : `http://www-master.ufr-info-p6.jussieu.fr/site-annuel-courant/ALGAV`. Il y a 40 listes de clés de 128 bits encodées en hexadécimal (une clé par ligne dans chaque fichier) contenant chacune 100, 200, 500, 1000, 5000, 10000, 20000 ou 50000 clés.

2 Structure 1 : Tas priorité *min*

Sur le transparent 11 du cours du chapitre 1, nous avons introduit la structure de données de tas *min*. La structure sera représentée en mémoire avec deux structures distinctes : via un arbre binaire et via un tableau.

Question 2.3 Implanter les 4 fonctions fondamentales d'un tas *min* : `SupprMin, Ajout, ConsIter` et `Union`, pour chacune des deux structures (tableau et arbre). Ces dernières permettent respectivement de supprimer l'élément de clé minimale dans la structure, d'ajouter un élément à un tas, de construire itérativement un tas à partir d'une liste d'éléments (par ajout successifs) et de faire l'union de 2 tas pour en constituer 1 seul.

Question 2.4 Prouver que les complexités (au pire cas) présentées dans le transparent 12 du cours sont vérifiées dans vos implémentations (on sera particulièrement méticuleux pour l'étude de la complexité de `ConsIter`).

Pour chacune des deux structures effectuer les expérimentations suivantes.

Question 2.5 Utiliser les *jeux de données aléatoires* afin de vérifier graphiquement la complexité temporelle de la fonction ConsIter. Pour chaque taille de listes, calculer le temps de construction en mémoire de la structure de données et en faire la moyenne. Puis représenter sur un graphique les moyennes pour chacune des tailles.

Question 2.6 Utiliser les *jeux de données aléatoires* afin de vérifier graphiquement la complexité temporelle de la fonction Union. Expliquer la stratégie mise en place pour l'expérimentation, c'est-à-dire à partir de quelles données sont construites les expérimentations.

Question 2.7 Quelle est l'implémentation la plus efficace ? Argumenter votre choix en fonctions des expérimentations.

3 Structure 2 : Files binomiales

Question 3.8 Définir et encoder les primitives de base concernant les files binomiales.

Question 3.9 Implanter les 4 fonctions fondamentales d'une files binomiale : `SupprMin, Ajout, ConsIter, Union`.

Question 3.10 Utiliser les *jeux de données aléatoires* afin de vérifier graphiquement la complexité temporelle de la fonction ConsIter.

Question 3.11 Utiliser les *jeux de données aléatoires* afin de vérifier graphiquement la complexité temporelle de la fonction Union. Reprendre la même stratégie que celle de la Question 2.8.

4 Fonction de hachage

Dans cette section, nous allons définir et implémenter une fonction de hachage classique nommée *Message Digest 5*, ou `MD5`. Implémenter l'algorithme dont le pseudo-code est donné sur la page wikipedia de l'algorithme : `https://fr.wikipedia.org/wiki/MD5`.

5 Arbre de Recherche

Implémenter une structure arborescente de recherche permettant, en moyenne de savoir si un élément est contenu dans la structure de donnée en $O(\log n)$ où n est le nombre de clés stockées. On rappelle que les clés seront codées sur 128 bits.

6 Étude expérimentale

La dernière partie du devoir consiste à comparer expérimentalement les structures, sur des données réelles. On utilisera les mots de l'œuvre de Shakespeare [1]. Les mots que nous considérons sont construits sur l'alphabet du code ASCII. Celui-ci est composé de 128 caractères, chacun étant encodé sur 8 bits.

Question 6.12 Stocker dans une structure arborescente de recherche, le haché `MD5` de chaque mot, c'est-à-dire le résultat de l'application de la fonction `MD5` sur chaque mot. En parallèle, construire une liste des mots de l'œuvre de Shakespeare où chaque mot n'apparaît qu'une seule fois.
L'ordre d'apparition des mots dans le résultat doit correspondre à l'ordre induit par la première occurrence de chaque mot des fichiers pris en entrée.

Question 6.13 (facultatif) Quels sont les ensembles des mots différents de l'œuvre de Shakespeare qui sont en collision pour *MD5* ?

Question 6.14 Comparer graphiquement les temps d'exécution des algorithmes `SupprMin, Ajout, ConsIter, Union` pour les deux types de structure de données : tas *min* et files binomiales. Pour le tas *min*, on prendra l'implémentation la plus efficace d'après la Question 2.9), sur les données extraites de la question 6.14.
Pour l'algorithme `Union` expliquer quelles données sont utilisées.

1. Une archive de chaque œuvre est disponible sur le site de l'UE : `http://www-master.ufr-info-p6.jussieu.fr/site-annuel-courant/ALGAV`.