Université de Limoges **Année universitaire 2015-2016**
Faculté des Sciences et Techniques

Master 1 Informatique et Mathématiques
Complexité et calculabilité - Feuille d'exercices n°4

Notation de Landau

Exercice 1.

1. Soient f, g et h des fonctions positives d'une variable réelle. Montrer que si $f \in \mathcal{O}(h)$ et $g \in \mathcal{O}(h)$, alors $f + g \in \mathcal{O}(h)$. Montrer que si $f \in \mathcal{O}(h)$ et $g \in \mathcal{O}(h)$, alors $fg \in \mathcal{O}(h^2)$.
2. Soient f, g et h des fonctions positives d'une variable réelle. Montrer que si $f \in \mathcal{O}(g)$ et $g \in \mathcal{O}(h)$, alors $f \in \mathcal{O}(h)$.
3. En utilisant la notation de Landau, exprimer le comportement asymptotique des fonctions suivantes :

$$f_1(n) = 3n^2 + 5ne^n \quad f_2(n) = \frac{17}{4}n^2 \log^3 n + 5n^3 \log^2 n \quad f_3(n) = 4\sqrt{n} + \log n \quad f_4(n) = 17$$

$$f_5(n) = e^{n+\sqrt{n}} \quad f_6(n) = \log(n^3) + n \quad f_7(n) = f_1(n)f_2(n)f_4(n).$$

4. Soit f une fonction positive strictement croissante. Montrer que $\sum_{i=0}^{n} f(i) \in \mathcal{O}(nf(n))$. En déduire que que $\sum_{i=1}^{n} i^m \in \mathcal{O}(n^{m+1})$.

Exercice 2. Soit N et M deux entiers positifs de tailles (binaires) respectives n et m.

1. Montrer que l'addition de M et N peut être effectuée par une machine de Turing en $\mathcal{O}(\max(n, m)) \subset \mathcal{O}(n + m)$ transitions. Que peut-on dire de cette complexité asymptotique ? Peut-on faire mieux ?
2. Quelle est la complexité asymptotique du calcul de $M + 1$ par une machine de Turing ?
3. En vous basant sur l'algorithme de multiplication usuel, montrer que le produit de M par N peut être effectué par une machine de Turing à 3 bandes en $\mathcal{O}(n \max(n, m)) \subset \mathcal{O}((n+m)^2)$ transitions. Que peut-on dire de cette complexité ? *Indication : on pourra se contenter de décrire le fonctionnement du multiplieur, sans chercher à écrire sa table de transition.*

   Remarque: il existe un algorithme plus sophistiqué permettant d'effectuer le produit en $\mathcal{O}((n + m) \log(n + m) \log\log(n + m))$ opérations binaires.

Exercice 3. Etudier l'algorithme de tri de fusion (voir en TD), évaluer la complexité. Peut-on faire mieux ?

Un modèle pour le calcul de la complexité binaire

Pour un calcul complexe, il est extrêmement fastidieux de construire explicitement une machine de Turing et de compter les transitions qu'elle effectue. De plus, dans cette partie du cours, on ne s'intéresse qu'à la distinction entre les algorithmes de complexité polynomiale et les autres. On n'a donc pas besoin d'une mesure très fine du coût des algorithmes.
On va donc se contenter de décrire les algorithmes dans un pseudo-langage de plus haut niveau en faisant des hypothèses (vérifiables) sur la complexité des opérations et instructions autorisées dans ce langage,

mesurées en nombre d'opérations sur les bits, de sorte que, s'il existe un algorithme de complexité polynomiale pour résoudre le problème dans ce langage, alors il existe une machine de Turing qui effectue le même traitement en un nombre polynomial de transitions.
Le pseudo-langage proposé est analogue à utilisé dans les cours d'algorithmique. D'une manière générale, on notera $T(x)$ la taille, en nombre de bits, d'un objet x.
Les hypothèses sur lesquelles on s'appuiera sont les suivantes:

1. Le coût d'une opération arithmétique sur deux entiers m et n (addition, multiplication, division euclidienne) est polynomial en la taille des arguments, c'est-à-dire en $T(m) + T(n)$. Idem pour un test d'égalité ou une comparaison.

2. Le coût d'une opération d'incrémentation ou de décrémentation d'un entier m est polynomial en $T(m)$.

3. Le coût d'une affectation $v \leftarrow x$ est polynomial (en fait linéaire) en $T(x)$.

4. Le coût de l'écriture ou de la lecture d'un objet x est $T(x)$. Idem pour une instruction de type `Retourner(`x`)`.

5. Le coût d'un test d'égalité entre deux objets x et y est polynomial en $T(x) + T(y)$.

6. Le coût d'un appel de fonction $f(x)$ est polynomial en $T(x)$. Ce coût ne comptabilise évidemment pas le coût de l'exécution de f sur x, qu'il faudra ajouter. S'il y a plusieurs paramètres, on ajoutera les tailles.

7. Le coût de l'exécution d'une instruction:
   `Tant que <condition> faire <suite d'instructions>`
   est la somme des coûts d'exécution de la suite d'instructions plus la somme des coûts du test conditionnel.

8. Le coût d'une instruction:
   `Pour` i `de` i_0 `à` n `faire <suite d'instructions>`
   est la somme des coûts d'exécution de la suite d'instructions plus un polynôme en $(n - i_0)$, ce dernier comptabilisant le coût de l'incrémentation d'un compteur et celui du test d'arrêt.

9. Le coût d'une instruction:
   `Si <condition> alors <suite d'instruction> sinon <suite d'instruction>`
   est évidemment la somme du coût du test conditionnel de celui des instructions effectuées.

10. La taille d'un tableau (à une ou plusieurs dimensions) est la somme des tailles des objets qu'il contient.

11. etc.

Ainsi, avec ces hypothèses, si le coût d'un algorithme s'avère polynomial en la taille des entrées, on pourra conclure qu'il existe une machine de Turing qui effectue le même traitement en un nombre polynomial de transitions.

Exercice 4. Montrer que le calcul du produit de deux matrices carrées (n, n) et à coefficients entiers de taille k peut se faire en temps polynomial.

Exercice 5. On considère un graphe non orienté $G = (S, A)$, où $S = \{1, 2, \ldots, n\}$ est l'ensemble de sommets et A l'ensemble des arêtes (paires de sommets). Montrer que la question de déterminer si G est connexe est dans P. On pourra supposer que le graphe est représenté par sa matrice d'adjacence $M = (m_{ij})_{1 \leq i,j \leq n}$, définie par $m_{ij} = 1$ si $\{i, j\}$ est une arête de G et $m_{ij} = 0$ sinon.

Exercice 6. Soit A un alphabet. On considère un algorithme qui prend en entrée un entier n et qui écrit tous les mots de A^* de longueur n. Un tel algorithme peut-il avoir une complexité polynomiale ?

Exercice 7. On considère la fonction suivante :

`Fonction Square-and-Multiply`(a ,n)

Entrée: a et n, deux entiers positifs
Sortie: l'entier a^n

{
$r \leftarrow 1$
Tant que $n > 0$ faire
{
Si $n \mod 2 = 1$ alors $r \leftarrow r \times a$
$a \leftarrow a \times a$
$n \leftarrow n/2$ // Quotient de la division euclidienne par 2
}
Retourner(r)
}

1. Vérifier que le nombre d'opérations arithmétiques dans $\mathbb{Z}$ effectuées par cet algorithme est polynomial en $T(n)$.

2. Peut-on en conclure que cet algorithme est de complexité (binaire) polynomiale ? Indication: attention à la taille des entiers sur lesquels s'effectuent ces opérations arithmétiques.