Complexité & graphes : TD1

Université de Tours

`Département informatique de Blois`

Complexité et machines de Turing

*
* *

Problème 1

1. Soient f et g deux fonctions positives. On note $f \in o(g)$ si est seulement si :

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

On dit que f est *négligeable* devant g.

Démontrer que $f \in o(g) \Rightarrow f \in O(g)$.

On rappelle la définition de $f \in O(g)$:

$$f \in O(g) \Leftrightarrow \exists C \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, \frac{f(n)}{g(n)} \leq C$$

On voit clairement ici que o est un cas particulier de O. En outre, la propriété est valide lorsque $n \to \infty$. Dès lors, on a: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \leq C$ quelconque.

2. Donner l'ordre de complexité des fonctions suivantes[1] :

- $f_1(n) = 3n^2 + 3n + 1$

  $f_1 \in O(n^2)$

- $f_2(n) = 2^{n+100}$

  $f_2 \in O(2^n)$

- $f_3(n) = \log(\sqrt{n} + n + n2^n)$

  On sait que $f_3 \underset{\infty}{\sim} \log(n2^n)$
  On sait que $\log(n2^n) = \log(2^n) + \log(n) = n + \log(n)$
  $f_3 \in O(n)$

- $f_4(n) = 2^{\log(2n)}$

  $f_4 \in O(n)$

- $f_5(n) = n + (\log n)^2$

  $f_5 \in O(n)$

- $f_6(n) = \log_n 2^n$

  On a $f_6(n) = \frac{\ln(2^n)}{\ln(n)} = \frac{\ln(2)\log_2(2^n)}{\ln(n)} = \frac{\ln(2)n}{\ln(n)}$
  $f_6 \in O\left(\frac{n}{\ln(n)}\right)$

3. Hiérarchie de classes de complexité.

   (a) Montrer que $2^{\sqrt{\log(n)}} \in O(n)$.

[1] Sauf précision, on considérera que log désigne $\log_2$, le logarithme de base 2.

On a pour $n > 2, 2^{\sqrt{\log(n)}} \leq 2^{\log(n)} \Leftrightarrow 2\sqrt{\log(n)} \leq n \Leftrightarrow \frac{2\sqrt{\log(n)}}{n} \leq 1$
Dès lors $2\sqrt{\log(n)} \in O(n)$

(b) Montrer que $\forall k \geq 1, \log(n)^k \in O\left(2^{\sqrt{\log(n)}}\right)$. On considèrera que $\forall a > 1, \forall k \geq 1 \lim\limits_{X \to +\infty} \frac{X^k}{a^X} = 0$

Soit $\frac{\log(n)^k}{2^{\sqrt{\log(n)}}}$, on pose $X = \sqrt{\log(n)}$. On a : $\frac{X^{2k}}{2^X}$
On sait $\lim\limits_{X \to +\infty} \frac{X^k}{a^X} = 0$, dès lors $\lim\limits_{n \to \infty} \frac{\log(n)^k}{2^{\sqrt{\log(n)}}} = \lim\limits_{X \to \infty} \frac{X^{2k}}{2^X} = 0$.
On a $\log(n)^k \in O\left(2^{\sqrt{\log(n)}}\right)$.

(c) Hiérarchiser les trois classes de complexité : $O\left(2^{\sqrt{\log(n)}}\right), O\left(\log(n)^k\right)$ et $O(n)$ selon la relation d'inclusion $\subseteq$.

D'après (a) et (b), on a $O\left(\log(n)^k\right) \subseteq O\left(2^{\sqrt{\log(n)}}\right) \subseteq O(n)$.

Problème 2 (partiel 2019)

Étant donné un réel $\gamma > 0$, le but de cet exercice est de prouver que $e^{\gamma n} \in O(n!)$. Pour cela, on pose les deux suites $(u_n)_{n \geq 1}$ et $(v_n)_{n \geq 1}$ définies telles que :

$$\begin{cases} u_n = e^{\gamma n} \\ v_n = n! \end{cases}$$

1. Montrer que : $\exists n_0 \geq 1, \forall n \geq n_0, \frac{u_{n+1}}{u_n} \leq \frac{1}{2}\frac{v_{n+1}}{v_n}$.

- $\frac{u_{n+1}}{u_n} = \frac{e^{\gamma(n+1)}}{e^{\gamma n}} = e^{\gamma}$
- $\frac{v_{n+1}}{v_n} = \frac{(n+1)!}{n!} = n + 1$

On résout :

$\frac{u_{n+1}}{u_n} \leq \frac{1}{2}\frac{v_{n+1}}{v_n}$
$\Leftrightarrow e^{\gamma} \leq \frac{1}{2}(n+1)$
$\Leftrightarrow 2e^{\gamma} - 1 \leq n$
On a $n_0 = \lceil 2e^{\gamma} - 1 \rceil$

2. En déduire que : $\exists C \in \mathbb{R}, \forall n \geq n_0, u_{n+1} \leq C\left(\frac{1}{2}\right)^{n-n_0} v_{n+1}$. On pourra démontrer à l'aide d'un télescopage.

On reprend l'inéquation ci-dessus et on l'applique pour tous les rangs entre n et n_0 :

$$\frac{u_{n+1}}{u_n} \times \frac{u_n}{u_{n-1}} \times ... \times \frac{u_{n_0+1}}{u_{n_0}} \leq \frac{1}{2}\frac{v_{n+1}}{v_n} \times \frac{1}{2}\frac{v_n}{v_{n-1}} \times ... \times \frac{1}{2}\frac{v_{n_0+1}}{v_{n_0}}$$

Par télescopage, on obtient :

$$\frac{u_{n+1}}{u_{n_0}} \leq \left(\frac{1}{2}\right)^{n-n_0} \frac{v_{n+1}}{v_{n_0}} \Leftrightarrow u_{n+1} \leq \frac{u_{n_0}}{v_{n_0}} \left(\frac{1}{2}\right)^{n-n_0} v_{n+1}$$

On voit que $C = \frac{u_{n_0}}{v_{n_0}}$. L'inéquation est vraie pour tout $n \geq n_0$.

3. En conclure que $e^{\gamma n} \in O(n!)$.

On sait que $\forall n \in \mathbb{N}, u_n > 0$.

On a montré question (b) que $\forall n \geq n_0, u_{n+1} \leq C\left(\frac{1}{2}\right)^{n-n_0} v_{n+1}$, alors :

$$\forall n \geq n_0, 0 < \frac{u_{n+1}}{v_{n+1}} \leq C\left(\frac{1}{2}\right)^{n-n_0}$$

On applique la limite, on a : $0 < \lim\limits_{n\to\infty} \frac{u_{n+1}}{v_{n+1}} \leq \lim\limits_{n\to\infty} C\left(\frac{1}{2}\right)^{n-n_0}$.

Or, on sait que $\forall \alpha \in]-1,1[, \lim\limits_{n\to\infty} \alpha^n = 0$. Par le théorème des gendarmes, on a : $\lim\limits_{n\to\infty} \frac{u_{n+1}}{v_{n+1}} = 0$

Dès lors $e^{\gamma n} \in o(n!)$ et donc $e^{\gamma n} \in O(n!)$.

Problème 3

1. Algorithme A :

Algorithme 1 : $A(n, m)$

```
Data : n ≥ 0, m ≥ 0
begin
    i ← 1
    j ← 1
    while j ≤ n do
        if i ≤ m then
            i ← i + 1
        else
            j ← j + 1
```

Regardons l'algorithme, on voit que celui-ci implémente deux conditions :

(a) La condition de la boucle **while** requiert au minimum n opérations.

(b) Seulement, l'incrémentation de j n'est possible que si $i > m$. Dès lors, il faut au préalable réaliser m incrémentations de i pour pouvoir incrémenter j.

Dès lors : $T_A(n, m) = \begin{cases} 2 & \text{si } n = 0 \\ 2 + 3(m + n) & \text{sinon} \end{cases}$.

Au final, on a une complexité globale telle que $T_A(n, m) \in O(m + n)$.

2. Algorithme B :

Algorithme 2 : $B(n)$

```
Data : n ≥ 0
begin
    i ← 1 while i ≤ n do
        j ← 1 while j ≤ i do
            j ← j + 1
        i ← i × 2
```

Cet algorithme est constitué de deux boucles imbriquées qui peuvent être modélisées computationnellement de la forme suivante.

Comme la boucle extérieure est en progression géométrique de raison 2, on cherche le rang n_0 tel

que $2^{n_0} = n \Leftrightarrow n_0 = \log_2(n)$.

Il vient que pour aller jusqu'à n, on exécute $\log(n)$ opérations.

Dès lors : $T_B(n) = \sum_{i=1}^{\lfloor \log(n) \rfloor} \sum_{j=1}^{2^i} 1$

$$= \sum_{i=1}^{\lfloor \log(n) \rfloor} 2^i$$

$$\underset{+\infty}{\sim} 2^{\lfloor \log(n) \rfloor + 1} = 2n \in O(n)$$

3. Algorithme C :

Algorithme 3 : $C(n, m)$

Data : $n \geq 0, m \geq 0$

begin

```
i ← 1
j ← 1
while j ≤ n do
    if i ≤ m then
        i ← i + 1
    else
        j ← j + 1
        i ← 1
```

Cet algorithme est semblable au A, cependant, lorsque les m premières étapes sont passées, l'algorithme incrémente j et remet la variable i à 1 ce qui oblige à ré-effectuer m incrémentations.

Dès lors : $T_C(n, m) = \begin{cases} 2 & \text{si } n = 0 \\ 2 + n(2m + 3) & \text{sinon} \end{cases}$.

Au final, on a une complexité globale telle que $T_C(n, m) \in O(m \times n)$

4. Algorithme D :

Algorithme 4 : $D(n)$

Data : $n \geq 0$

begin

```
if n < 2 then
    return 1
else
    return D(n − 2) + D(n − 1)
```

L'algorithme est récursif et calcule la suite de Fibonacci. De façon générale, notons, $C \geq 1$ le nombre d'opérations élémentaires effectuées.

On note $T_D(n) = C + T_D(n-1) + T_F(n-2)$

Par croissante de $T_D(n)$, on peut majorer $T_D(n)$ telle que :

$T_D(n) \leq C + 2T_F(n-1)$

$T_D(n) \leq C + 2\left(C + 2T_D(n-2)\right)$

$\vdots$

$T_D(n) \leq C + 2C + 4C + ... + 2^n C$

$T_D(n) \leq C\left(\sum_{i=0}^{n} 2^i\right) = C(2^{n+1}-1)$

Dès lors $T_D(n) \in O(2^n)$.

En minorant $T_D(n)$ par $C + 2T_D(n-2)$, on montre de manière similaire que $T_F(n) \in \Omega\left(2^{n/2}\right)$.

On peut calculer plus exactement la complexité de la manière suivante :

Soit $T'_D(n) = T_D(n) - C = T_D(n-1) + T_D(n-2)$, alors $T'_D(n)$ est une récurrence linéaire d'ordre deux. Il vient que :

$T'_D(n) = K_1 r_1^n + K_2 r_2^n$ avec $(K_1, K_2) \in \mathbb{R}^2$ et r_1 et r_2 les racines du polynôme résultant de l'expression de T'_D, soit : $X^2 + X - 1$

Alors $r_1 = \varphi = \frac{1+\sqrt{5}}{2}$ et $r_2 = -\frac{1}{\varphi} = \frac{1-\sqrt{5}}{2}$.

Dès lors $T_D(n) \underset{+\infty}{\sim} T'_D(n) \underset{+\infty}{\sim} K_1\varphi^n \in O(\varphi^n)$

5. Calcul de la puissance d'un nombre.

   (a) Soit $a > 0$, écrire un algorithme naïf permettant de calculer a^n pour $n \geq 1$.

Algorithme 5 : Algorithme naïf du calcul de a^n

```
Data : a > 0, n ≥ 1
begin
    a_pow ← 1
    i ← 1
    while i ≤ n do
        a_pow ← a_pow × a
    return a_pow
```

   (b) Déterminer un algorithme de complexité $O(\log n)$ et démontrer sa complexité.

On propose l'algorithme récurisf suivant

Algorithme 6 : $pow(a, n)$: Algorithme efficace du calcul de a^n

```
Data : a > 0, n ≥ 1
begin
    if n = 1 then
        return a
    else
        if n mod 2 = 0 then
            return pow(a × a, n/2)
        else
            return a × pow(a × a, n/2)
```

Calculons sa complexité pour $n > 1$. On voit que la complexité de *pow* ne dépend que de n et peut-être calculée comme suit:

$T_{pow}(n) = C + T_{pow}(n/2)$ où $C \geq 1$ est une constante du nombre d'opérations maximum effectués lors d'un appel.

L'équation est récursive et on note alors que $T_{pow}(n) = C + T_{pow}(n/2) = C + C + T_{pow}(n/4) = ...$

On cherche à partir de quel rang n_0, on a $\frac{n}{2^{n_0}} = 1 \Leftrightarrow n = 2^{n_0} \Leftrightarrow \log(n) = n_0$.

Dès lors, $T_{pow}(n) = C\,\lfloor \log(n) \rfloor \in O(\log(n))$

Problème 4

1. Avec un ordinateur actuel, vous êtes capable de traiter en 1 heure un problème en $O(n)$ d'une taille maximale de N. Avec la même procédure de calcul, quelle est la taille maximale du problème que vous pourriez traiter en 1 heure avec une machine 100 fois plus rapide ou 1 million de fois plus rapide.

> On note $T_P : \mathbb{N} \to \mathbb{R}^+$, la fonction qui associe à la taille de l'instance d'un problème P, le temps de calcul de celle-ci.
>
> Aussi, on note $T_P(N) = 1\text{h} \Leftrightarrow N = T_P^{-1}(1\text{h})$. Pour une machine X fois plus rapide on aboutit à l'équation :
>
> $$T_P(N') = X\text{h}$$
>
> Autrement dit, on va réaliser en 1h l'équivalent de Xh de calculs pour N'. Pour une machine 100 fois plus rapide, on a alors :
>
> $T_P(N') = 100\text{h} \Leftrightarrow N' = T_P^{-1}(100\text{h})$. Or, T_P est la fonction linéaire, donc $T_P^{-1} = n$. On a alors : $N' = 100\text{h} \Leftrightarrow N' = 100 \times \underbrace{1\text{h}}_{N} = 100N$.
>
> Le raisonnement est analogue pour $X = 10^6$, on trouve $10^6 N$.

2. Répondez à la même question pour des problèmes en $O(n^2)$, $O(n^5)$, $O(2^n)$ et $O(3^n)$.

> De façon similaire à précédemment, on cherche à résoudre $N' = T_P^{-1}(100\text{h})$.
>
> Or, on suppose ici dans un premier temps que le problème est de complexité $O(n^2)$, dès lors, $T_P^{-1}(n) = \sqrt{n}$. On a alors:
>
> $N' = \sqrt{100\text{h}} \Leftrightarrow N' = 10 \times \sqrt{1\text{h}} = 10N$.
>
> On peut donc uniquement résoudre une instance 10 fois plus grosse pour un problème de complexité $O(n^2)$ avec une machine 100 fois plus puissante.
>
> Avec un problème de complexité $O(3^n)$ et une machine un million de fois plus puissante on a :
>
> $T_P(N') = 10^6 h \Leftrightarrow N' = \log_3\left(10^6 h\right) \Leftrightarrow N' = \underbrace{\log_3(10^6)}_{=12.58} + \log_3(1h) = 12 + N$.
>
> Soit environ une instance $N + 12$.

Problème 5

Soit la machine de Turing $M = (Q, \Gamma, \Sigma, \delta, q_0, \#, \emptyset)$ où :

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Gamma = \{a, b, A, A', B, B', \#\}$
- $\Sigma = \{a, b\}$
- $\delta : Q \times \Sigma \to Q \times \Gamma \times \{\mathtt{L}, \mathtt{R}\}$

$\delta(q_0, a) = (q_1, A', \mathtt{R})$ $\quad$ $\delta(q_1, b) = (q_1, b, \mathtt{R})$

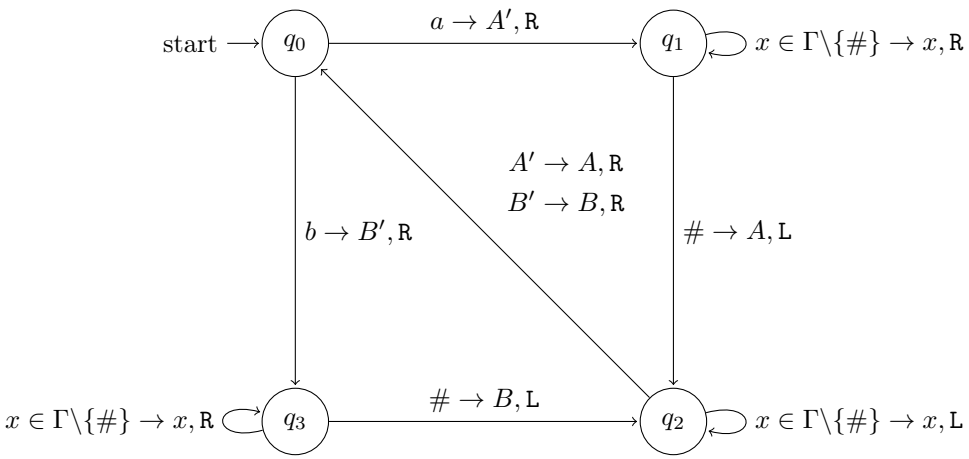
$\delta(q_0, b) = (q_3, B', \mathtt{R})$ $\quad$ $\delta(q_1, A) = (q_1, A, \mathtt{R})$

$\delta(q_1, a) = (q_1, a, \mathtt{R})$ $\quad$ $\delta(q_1, B) = (q_1, B, \mathtt{R})$

$\delta(q_1, \#) = (q_2, A, \mathtt{L})$
$\delta(q_2, a) = (q_2, a, \mathtt{L})$
$\delta(q_2, b) = (q_2, b, \mathtt{L})$
$\delta(q_2, A) = (q_2, A, \mathtt{L})$
$\delta(q_2, B) = (q_2, B, \mathtt{L})$
$\delta(q_2, A') = (q_0, A, \mathtt{R})$
$\delta(q_2, B') = (q_0, B, \mathtt{R})$

$\delta(q_3, \#) = (q_2, B, \mathtt{L})$
$\delta(q_3, a) = (q_1, a, \mathtt{R})$
$\delta(q_3, b) = (q_1, b, \mathtt{R})$
$\delta(q_3, A) = (q_1, A, \mathtt{R})$
$\delta(q_3, B) = (q_1, B, \mathtt{R})$
$\delta(q_3, \#) = (q_2, B, \mathtt{L})$

1. Représenter cette machine sous forme de diagramme d'états.



2. Quel est le contenu du ruban après l'exécution de M sur le mot d'entrée $abab$?

	#	A	B	A	B	A	B	A	B	#	

3. Quel est le comportement général de cette machine pour un mot d'entrée de la forme $(a|b)^*$?

On peut montrer que pour un mot $x \in (a|b)^*$, le contenu final du ruban est de la forme $X \cdot X$ où X est la version capitalisée de x.

En substance, on remarque que la machine est symétrique au niveau des états q_1 et q_3 respectivement responsable du traitement des symboles a et b. Dès lors, on peut représenter plus simplement le comportement de la machine M par la fonction $\mathrm{maj}(x) = \begin{cases} A \cdot \mathrm{maj}(x'A) & \text{si } x = a \cdot x' \\ B \cdot \mathrm{maj}(x'B) & \text{si } x = b \cdot x' \\ \varepsilon & \text{sinon} \end{cases}$

Problème 6

1. Comment peut-on reconnaître simplement qu'un nombre écrit en binaire est divisible par 4 ?

On rappelle que l'écriture binaire d'un nombre entier n est une combinaison unique de $x_i \in \{0, 1\}$ telle que $n = \sum_{i=0}^{N} x_i 2^i$.

Dès lors, pour $i \geq 2$, on voit que toutes les puissances supérieures sont factorisables par 4.

Ainsi, un nombre est divisible par 4 si ses deux premiers bits x_0 et x_1 sont égaux à 0 ou que $x = 0$.

2. Proposer une machine de Turing permettant de décider si oui ou non un nombre binaire est divisible

par 4 ?

$M = (\{q_0, q_1, q_2, q_f\}, \{0, 1, \#\}, \{0, 1\}, \delta, q_0, \#, \{q_f\})$, avec δ telle que :

$\delta(q_0, 0) = (q_0, 0, \mathtt{R})$

$\delta(q_0, 1) = (q_0, 1, \mathtt{R})$

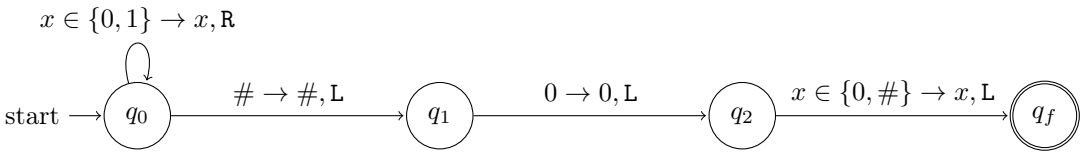
$\delta(q_0, \#) = (q_0, \#, \mathtt{L})$

$\delta(q_1, 0) = (q_2, 0, \mathtt{L})$

$\delta(q_2, 0) = (q_f, 0, \mathtt{L})$

$\delta(q_2, \#) = (q_f, \#, \mathtt{L})$

On obtient le diagramme d'états suivant :



3. Vérifier que votre machine reconnaît que 20 est divisible par 4.

On a $20 = \langle 10100 \rangle_2$. Dès lors, on a la dérivation :

$(q_0, \varepsilon, 10100) \vdash^* (q_0, 10100, \varepsilon)$

$(q_0, 10100, \varepsilon) \vdash (q_1, 1010, 0)$

$(q_1, 1010, 0) \vdash (q_2, 101, 00)$

$(q_2, 101, 00) \vdash (q_f, 1010, 0)$

Problème 7

1. Proposer une machine de Turing permettant de tester si un mot d'entrée en binaire comporte le même nombre de 0 que de 1.