Devoir Libre n° 10*

La copie est à rédiger en LaTeX.

Me transférer le source en `.tex` compilant sans erreur (il peut y avoir des warnings, ainsi qu'un pdf.

Utiliser les canaux `#Maths` et `#LaTeX` du `Mattermost` dont je rappelle l'adresse

`https://chat.mp2i.synology.me/`

pour toute demande d'aide.

Une preuve du Théorème Maître [1]

A. Rappels et notations

- La partie entière d'un réel x est l'unique entier, noté $\lfloor x \rfloor$, tel que

$$\lfloor x \rfloor \leqslant x < \lfloor x \rfloor + 1.$$

  Ainsi, c'est le plus grand des entiers inférieurs ou égaux à x.
- La partie entière par excès de x est l'unique entier, noté $\lceil x \rceil$, tel que

$$\lceil x \rceil - 1 < x \leqslant \lceil x \rceil.$$

  Ainsi, c'est le plus petit des entiers supérieurs ou égaux à x.
- Pour réel $b > 1$, on définit le logarithme en base b par

$$\forall x \in \mathbb{R}_+^*, \quad \log_b(x) = \frac{\ln(x)}{\ln(b)}.$$

- Soit (u_n) et (v_n) deux suites strictement positives. Alors (u_n) est dominée par (v_n), c'est-à-dire $u_n = \mathcal{O}(v_n)$, si et seulement si la suite $\left(\frac{u_n}{v_n}\right)$ est majorée, donc s'il existe un réel $M > 0$ tel que

$$\forall n \in \mathbb{N}, \quad u_n \leqslant M v_n.$$

  Si (u_n) n'est définie qu'à partir d'un certain rang n_0, l'inégalité précédente n'est bien entendu requise qu'à partir de ce rang.

B. Problématique

Un algorithme de type « diviser pour régner » est un algorithme récursif ramenant la résolution d'un problème de taille n à la résolution d'un ou plusieurs problèmes de taille $\frac{n}{b}$ (à ± 1 près pour avoir un entier). Ainsi, la complexité d'un tel algorithme vérifiera une relation du type

$$\forall n \geqslant 2, \quad T(n) = \alpha T\left(\left\lfloor \frac{n}{b} \right\rfloor\right) + \beta T\left(\left\lceil \frac{n}{b} \right\rceil\right) + u_n,$$

initialisée par la donnée de $T(0)$ et $T(1)$, vérifiant $T(0) > 0$ et $T(1) > 0$, et pour tout $n \geqslant 2$, $u_n > 0$.

Dans cette formule, b est un certain entier supérieur ou égal à 2 , et α et β sont des entiers naturels tels que $\alpha + \beta \geqslant 1$.

1. D'après un sujet de A. Troesch

Le problème initial a été ramené à $\alpha+\beta$ problèmes de taille peu près $\frac{n}{b}$ (plus précisément les deux entiers entourant cette valeur, ou la valeur précise de $\frac{n}{b}$ lorsque celle-ci est entière).

Ainsi, on doit compter le temps d'exécution de chacun de ces sous-problèmes, ainsi que le temps du traitement qui permet de reconstituer la solution du problème de taille n à partir des solutions des sous-problèmes (c'est le rôle de la suite (u_n)).

Dans tout le problème, on se donne une suite T vérifiant les conditions ci-dessus.

Le but du problème est d'étudier le comportement asymptotique de T, c'est-à-dire d'estimer l'ordre de grandeur de T (ou au moins d'en trouver un majorant). Cette étude permet notamment de prévoir l'effet sur T de l'augmentation de la taille données. Par exemple, si $T(n)$ a le même ordre de grandeur que n (on dit que T est linéaire), doubler n aura pour effet de doubler $T(n)$. Si T a le même ordre de grandeur que n^2 (on dit que T est quadratique), doubler n aura pour effet de multiplier T par 4.

Dans la situation décrite, le comportement dépend fortement de l'ordre de grandeur de u_n. On se placera dans tout le problème dans une situation particulière qu'on explicite ci-dessous. Le même type de techniques permet d'obtenir un résultat plus général que celui qu'on montrera, distinguant trois grands cas pour les conditions à poser sur (u_n). C'est un résultat classique d'informatique théorique.

On note dans tout le problème $a=\alpha+\beta$ et $\gamma=\log_b(a)$. Par ailleurs, on note, pour tout $n\in\mathbb{N}$, $\ell(n)=\lceil\log_b(n)\rceil$.

On suppose dans tout le problème que $u_n=\mathcal{O}(n^\gamma)$

C. Questions préliminaires

1. Montrer que pour tout $x\in\mathbb{R}_+^*$, $b^{\log_b(x)}=x$.
   Ainsi, on a en particulier $b^\gamma=a$.
2. Montrer que pour tout $n\in\mathbb{N}$, $T(n)>0$.

D. Majoration de $T(n)$

D'après l'hypothèse faire sur (u_n), on peut se donner, dans toute la suite du problème, un réel strictement positif M tel que

$$\forall n\geqslant 2,\quad u_n\leqslant Mn^\gamma.$$

1. Montrer $\dfrac{\ell(n)}{\log_b(n)}\xrightarrow[n\to+\infty]{}1$.
2. Montrer que si n est un entier strictement positif,

$$\left\lceil\frac{n}{b}\right\rceil\leqslant 1+\frac{n-1}{b}.$$

3. Montrer que pour tout $n\geqslant b$, $\ell\left(\left\lceil\dfrac{n}{b}\right\rceil\right)=\ell(n)-1$
4. On note, pour tout $k\in\mathbb{N}$,

$$P(k)=\prod_{i=1}^{k}\left(1+\frac{b-1}{b^i}\right).$$

   Montrer que pour tout $k\in\mathbb{N}$, $\ln(P(k))\leqslant\displaystyle\sum_{i=1}^{k}\frac{b-1}{b^i}$, puis en déduire que $(P(k))$ est majorée.
5. On note, pour $n\in\mathbb{N}^*$, $Q_n=P(\ell(n)-1)$.
   Montrer que pour tout $n\geqslant b$, $\left(1+\dfrac{b-1}{n}\right)Q_{\lceil\frac{n}{b}\rceil}\leqslant Q_n$.

6. On définit $K = \max_{n\in[\![1,b-1]\!]}\left(\frac{T(n)}{n^\gamma Q_n^\gamma} - M(\ell(n)-1)\right)$. En raisonnant par récurrence forte, montrer que pour tout $n \in \mathbb{N}^*$,

$$\frac{T(n)}{n^\gamma} \leqslant Q_n^\gamma (K + M(\ell(n)-1))$$

On fera attention à bien comprendre quelles valeurs de n il est important de faire entrer dans l'initialisation.

7. En déduire que $T(n) = \mathcal{O}\left(n^\gamma \log_b(n)\right)$ puis $T(n) = \mathcal{O}(n^\gamma \ln(n))$.

Avec des hypothèses un peu plus fortes sur u_n, on peut avoir des résultats plus précis (obtenir également une minoration). Les techniques sont sensiblement les mêmes, et il ne serait pas d'un grand intérêt de les développer ici. Nous nous contenterons donc de cette majoration.

E. Exemples

1. **L'algorithme d'exponentiation rapide**

Cet algorithme a pour but de calculer de façon efficace x^n, pour un réel x fixé, et un entier $n \in \mathbb{N}$. Il se déroule de la façon suivante :

- si $n = 0$, l'algorithme retourne 1
- Si $n > 0$, on ramène le calcul de x^n au calcul de $p = x^{\lfloor \frac{n}{2} \rfloor}$ (i.e. on relance l'algorithme avec cette nouvelle valeur de l'exposant). On retrouve alors le résultat attendu en remarquant que $x^n = p^2$ si n est pair, et $x^n = p^2 \times x$ si n est impair.

(a) Montrer que cet algorithme permet de calculer x^n pour toute valeur de $n \in \mathbb{N}$.

(b) En supposant que $T(n)$ compte le nombre d'opérations nécessaires pour calculer x^n (on compte une affectation et un test de comparaison comme une opération), montrer que pour tout $n \in \mathbb{N}$

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + u_n,$$

où (u_n) est une suite majorée (suivant la façon d'écrire les instructions, on peut aboutir à des expressions différentes de u_n, donc à un majorant différent).

(c) En déduire que $T(n) = \mathcal{O}(\ln(n))$. Bien sûr, on ne se contentera ici d'arguments arboricoles.

(d) Comparer, pour des grandes valeurs de n, cet algorithme à l'algorithme naïf consistant à multiplier n fois successivement par x une variable initialisée à 1 .

2. **Le tri fusion**

Le tri fusion est un algorithme permettant de trier un tableau `tab` de réels (ou tous autres objets qu'on sait comparer entre eux). Il se déroule de la façon suivante :

- Si `tab` est de taille 1, il est trié, et on ne fait rien.
- Si `tab` est de taille $n \geqslant 2$, on coupe `tab` en deux sous-tableaux de longueurs $\left\lfloor \frac{n}{2} \right\rfloor$ et $\left\lceil \frac{n}{2} \right\rceil$. On relance l'algorithme sur ces sous-tableaux. En supposant qu'il aboutisse (c'est le cas car la taille des tableaux traités diminue strictement au fil des appels récursifs), on dispose donc de deux sous-tableaux triés.
- On fusionne ces deux tableaux triés de sorte à obtenir le tableau initial trié.

Montrer qu'en effectuant la fusion des deux sous-tableaux de façon convenable (qu'on expliquera), le nombre $T(n)$ d'opérations nécessaires pour trier un tableaux de longueur n par le tri fusion vérifie

$$T(n) = \mathcal{O}(n \ln n).$$

FIN DE L'ÉNONCÉ