Examen de Complexité

Durée : 1h30

Documents et calculatrice autorisés

Connexion à Internet interdite (smartphones et ordinateurs interdits)

Justifiez précisément chacune de vos réponses

Le but de ce problème est d'étudier la complexité d'algorithmes, connus dès l'école primaire par tous les enfants du monde : l'addition et la multiplication de nombres écrits en base 10, en numération de position (la manière usuelle que nous utilisons pour écrire les nombres). On suppose disposer d'un ordinateur incapable de réaliser les opérations arithmétiques de base (additions, multiplications). On suppose en revanche qu'il a accès, en temps constant, aux tables d'additions et aux tables de multiplications, pour les nombres composés d'un seul chiffre décimal : on peut ainsi calculer $5 \times 7 = 35$ en temps constant, et on peut isoler en temps constant le chiffre des unités (5) et le chiffre des dizaines (3) de cette multiplication. Il en est de même pour l'addition $5 + 7 = 12$, avec le chiffre des unités (2) et le chiffre des dizaines (1) calculés en temps constant.

On représente un nombre de n chiffres par un tableau dont les indices vont de 0 pour le chiffre des unités, le plus à droite dans l'écriture usuelle du nombre, à $n-1$ pour le chiffre le plus à gauche.

Addition

1. Poser et effectuer l'addition suivante (à reproduire et à compléter dans votre copie) :

$$\begin{array}{r} 1 \quad 5^1 \quad 4 \quad 3^1 \quad 8 \\ + \quad 4 \quad 9 \quad 2 \quad 7 \\ \hline 1 \quad 0 \quad 3 \quad 6 \quad 5 \end{array}$$

Le résultat de l'addition est 10365.

2. Combien d'additions élémentaires à deux chiffres avez-vous utilisées pour effectuer cette addition de deux nombres à quatre chiffres ?

Pour chaque position (0, 1, 2, 3), il faut effectuer une addition à deux chiffres : $8+7$, $3+2$, $4+9$ et $5+4$. Il y en a donc déjà quatre. Mais deux de ces additions ($8+7$ et $4+9$) génèrent une retenue, qui doit, à son tour, être ajoutée à un des résultats précédents. Il y a donc en tout 6 additions. La dernière addition, $5+4+1$ génère aussi une retenue, mais pas de nouvelle addition, il suffit d'affecter cette retenue à la position 4 de V .

3. Etant donnés deux nombres T et U à n chiffres, représentés dans des tableaux de n éléments $T[0..n-1]$ et $U[0..n-1]$, décrire l'algorithme (en français, ou en pseudo-code) permettant de calculer V , le nombre à $n+1$ chiffres $V[0..n]$ représentant la somme des deux nombres T et U .

En français : si on suppose que le chiffre des unités est stocké en case 0 des tableaux, le chiffre des dizaines en case 1, et ainsi de suite, on peut calculer S égal à la somme de $T[0]$ et de $U[0]$. On place le chiffre des unités de ce résultat dans $V[0]$, et le chiffre des dizaines dans une variable retenue R (qui peut être égale à 0, s'il n'y a pas de retenue, ou à 1 s'il y a retenue). A l'itération suivante, il faut faire deux additions : $S = T[1] + U[1] + R$. L'une de ces deux additions (mais pas les deux) peut à son tour générer une retenue, que l'on stockera dans R pour la prochaine itération. Le chiffre des unités sera stocké dans $V[1]$. On continue comme cela jusqu'à calculer $S = T[n-1] + U[n-1] + R$. On place le chiffre des unités de ce résultat dans

$V[n-1]$ et le chiffre des dizaines dans $V[n]$ (ce n'est pas la peine de faire une addition de plus car ce serait le nombre 0 qui serait ajouté à cette éventuelle retenue).

4. Combien d'additions élémentaires effectue cet algorithme au maximum, en fonction de n ?

A la première itération, on fait une addition. A chacune des suivantes (pour i allant de 1 à $n-1$), on en fait deux (on pourrait parfois n'en faire qu'une si on testait qu'il n'y a eu pas de retenue à l'addition précédente). En tout, il y a donc au maximum $1 + 2 * (n-1)$, c'est à dire $2n-1$ additions.

5. Quelle est sa complexité en mémoire, sans tenir compte de la place occupée par les données d'entrée T et U ? Comment pouvez-vous réduire cette complexité à $O(1)$ en acceptant d'écraser une partie des données d'entrée ?

En plus des deux tableaux T et U , il faut stocker le résultat dans le tableau V qui contient $n+1$ cases. Les variables R et S comptent pour 1 case chacune, il y a donc $n+3$ cases mémoires consommées en plus des données d'entrée. La complexité mémoire est donc $O(n)$.

Si on accepte de perdre la valeur de l'un des deux tableaux (par exemple, U), on peut stocker le résultat dans U , en écrasant les chiffres de U , au fur et à mesure de l'addition : Au lieu de calculer $V[0] = T[0] + U[0]$, on peut calculer $U[0] = T[0] + U[0]$, en propageant les retenues de la même manière que précédemment. Au lieu de consommer $n+3$ cases mémoire supplémentaires, on consomme 3 cases mémoire supplémentaires, c'est à dire un nombre constant $O(1)$.

6. Etant donné un nombre N , exprimez en fonction de N , le nombre de chiffres n nécessaires à la représentation décimale de N . En déduire la complexité de l'addition des deux nombres M et N (avec $M \leq N$), en utilisant l'algorithme de la question 3, généralisé à deux nombres de taille m et n (avec $m \leq n$).

Le nombre de chiffres n nécessaires pour écrire le nombre N est égal à $n = \log_{10}(N)+1$, le logarithme en base 10 de N , plus 1. Par exemple, $\log_{10}(1000) = 3$, puisque $10^3 = 1000$. Et il faut bien $3+1 = 4$ chiffres pour écrire le nombre 1000. Autre exemple : $\log_{10}(1000000) = 6$, puisque $10^6 = 1000000$. Et il faut bien $6+1 = 7$ chiffres pour écrire 1000000. Si on prend les nombres M et N , il faut donc $m = \log_{10}(M)+1$ et $n = \log_{10}(N)+1$ chiffres pour les écrire. L'addition d'un nombre à m chiffres avec un nombre à n chiffres va s'écrire sur $n+1$ chiffres au plus, puisque n est le plus grand des deux. Si M et N n'ont pas la même taille (N étant le plus grand), il suffit de compléter les chiffres de M à gauche avec des 0, puis réutiliser l'algorithme de la question 3. La complexité de l'addition des deux nombres M et N est donc $O(n)$, c'est à dire $O(\log_{10}(N))$.

Multiplication

Une première approche pour effectuer la multiplication de deux nombres entiers M et N (avec $M \leq N$), consiste à considérer que l'on peut calculer cette opération, en additionnant M exemplaires du nombre N .

7. Quelle est la complexité d'un algorithme basé sur cette méthode de calcul ?

L'algorithme de base pour ajouter M exemplaires du nombre N consiste à faire $M-1$ additions : D'abord $N+N$ (une addition pour calculer deux fois le nombre N), puis le résultat + N (deux additions pour calculer trois fois le nombre N), et ainsi de suite, jusqu'à $M-1$ additions, pour calculer M fois le nombre N . Mais chacune de ces additions a elle-même une complexité proportionnelle à la taille du plus grand des deux nombres ajoutés (voir question 6). Au début, le plus grand des deux nombres est de taille $n = \log_{10}(N)$. A la fin, le dernier résultat est le nombre $M.N$, qui s'écrit avec $\log_{10}(M.N) + 1$ chiffres. Mais $\log_{10}(M.N) = \log_{10}(M) + \log_{10}(N) \leq 2 \log_{10}(N)$, puisque N est le plus grand des deux. Chacune des $M-1$ additions a donc une complexité de l'ordre de $O(\log_{10}(N))$. On a donc une complexité totale de l'ordre de $M \cdot \log_{10}(N)$.

qu'il faut également ajouter au résultat de la colonne qui suit. Il y a 5 retenues en bleu. En tout, cela fait donc $10 + 12 + 5 = 27$ additions.

11. Décrire l'algorithme (en français, ou en pseudo-code) permettant de calculer le nombre à $2n$ chiffres $V[0..2n-1]$, égal à la multiplication des deux nombres à n chiffres $T[0..n-1]$ et $U[0..n-1]$, en utilisant l'algorithme de la question 9.

Le code ci-dessous est une implémentation en PHP de l'algorithme de multiplication que vous utilisez dans vos cahiers depuis l'école primaire... On lui transmet les deux tableaux \$a et \$b, et il stocke le résultat dans \$c. Il effectue également l'optimisation proposée en question 13, en ne stockant pas explicitement tous les chiffres des tableau intermédiaires, mais en accumulant les unités et la retenue dans la case courante du résultat, et dans la case suivante.

```
function multiplie (&$c, &$a, &$b) {
    for($i=0; $i<count($a)+count($b); $i++)
        $c[$i] = 0; //initialisation à 0 du tableau résultat
    for ($i=0; $i<count($a); $i++) { // parcours de tous les chiffres du multiplicateur
        $aa = $a[$i]; // $aa est le chiffre courant
        for ($j=0; $j<count($b); $j++) { // parcours de tous les chiffres du multiplicande
            $bb = $b[$j]; // $bb est le chiffre courant
            $cc = $aa * $bb + $c[$i+$j]; // multiplication élémentaire des deux chiffres, et accumulation dans le chiffre du résultat
            $uc = $cc % 10; // isolement du chiffre des unités
            $dc = (int) ($cc/10); // calcul de la retenue
            $c[$i+$j] = $uc; // stockage des unités
            $c[$i+$j+1] += $dc; // propagation de la retenue dans la case suivante
        }
    }
}
```

12. Quelle est la complexité en temps et en mémoire, de cet algorithme ?

Cet algorithme effectue deux boucles imbriquées (pour i allant de 0 à $n-1$... pour j allant de 0 à $n-1$). Il effectue donc exactement $n * n = n^2$ itérations dans le corps de la boucle centrale, qui effectue un nombre constant d'opérations. La complexité en temps est donc de $O(n^2)$, c'est à dire quadratique en la longueur des nombres multipliés.

La complexité en mémoire de l'algorithme de la question 9 est $n^2+n = O(n^2)$ pour les lignes intermédiaires (n lignes de $n+1$ chiffres), et $2n$ pour le résultat, soit $n^2+3n = O(n^2)$. Dans l'optimisation qui était demandée en question 13, et qui est implémentée ci-dessus, les lignes intermédiaires ne sont pas utilisées, et le résultat se construit directement dans le tableau \$c de taille $2n$.

13. Voyez-vous un moyen pour diminuer l'ordre de grandeur de la complexité mémoire de cet algorithme à $2n$ cases mémoires supplémentaires seulement ?

Voir réponses précédentes.

Algorithme de Karatsuba

Considérons deux nombres de $2k$ chiffres : $M = \widehat{ab}$ et $N = \widehat{cd}$, où a, b, c et d sont des nombres de k chiffres. On désire calculer le produit $P=M \times N$. En 1960, Karatsuba remarque que le calcul de $(a \times 10^k + b)(c \times 10^k + d)$ qui, sous forme développée $ac \times 10^{2k} + (ad + bc) \times 10^k + bd$, semble nécessiter les quatre produits ac, ad, bc et bd , peut en fait être effectué seulement avec les trois produits ac, bd et $(a - b)(c - d)$ en regroupant les calculs sous la forme suivante :

$$(a \times 10^k + b)(c \times 10^k + d) = ac \times 10^{2k} + (ac + bd - (a - b)(c - d)) \times 10^k + bd$$

14. Pouvez-vous justifier cette nouvelle écriture ?

En identifiant les deux polynômes $ac \times 10^{2k} + (ad + bc) \times 10^k + bd$ et $ac \times 10^{2k} + (ac + bd - (a - b)(c - d)) \times 10^k + bd$, on se rend compte qu'ils coïncident déjà dans le terme en 10^{2k} et dans le terme constant. Il suffit de montrer l'égalité des termes en 10^k . En développant le deuxième, on obtient : $(ac + bd - (a - b)(c - d)) = ac + bd - (ac - ad - bc + bd) = \cancel{ac} + \cancel{bd} - \cancel{ac} + ad + bc - \cancel{bd} = ad + bc$, qui est bien le coefficient en 10^k du premier polynôme.

Pour de grands nombres, la méthode peut être appliquée de manière récursive pour les calculs de ac, bd et $(a - b)(c - d)$ en scindant à nouveau a, b, c et d en deux et ainsi de suite. C'est un algorithme de type « diviser-pour-régner ».

15. Soit $n = 2k$. Montrez que le temps d'exécution $T(n)$ de cet algorithme peut s'exprimer par la récurrence : $T(n) = 3 T(n/2) + O(n)$, que l'on pourra simplifier en $T(n) = 3 T(n/2)$.

On commence par calculer les trois produits ac , bd et $(a - b)(c - d)$. Les deux soustractions peuvent se faire en temps $O(n)$. Une fois les trois produits effectués, on calcule le nombre $x = ac$, le nombre $y = ac + bd - (a - b)(c - d)$ et le nombre $z = bd$. On peut obtenir le résultat final de la multiplication en effectuant l'addition suivante (dans cette figure, on suppose que $n=8$, que a, b, c et d ont donc 4 chiffres, que les produits x, y et z ont 8 chiffres et que le résultat final est sur 16 chiffres).

x	x	x	x	x	x	x	x									
				y	y	y	y	y	y	y	y					
										z	z	z	z	z	z	z

Cette addition peut également se faire en complexité $O(n)$. La complexité du produit P de deux nombre à $n=2k$ chiffres peut donc s'exprimer comme trois fois la complexité du produit de deux nombres à k chiffres, plus un temps linéaire en n , soit $T(n) = 3T(n/2)+O(n)$. On peut faire l'hypothèse que le terme en $O(n)$ sera négligeable devant l'expression de $T(n)$, ce qui sera vérifié en question 16. On obtient donc $T(n) = 3T(n/2)$.

16. Montrez que cette récurrence peut se ramener à $T(n) = 3^{\log_2 n} = n^{\log_2 3} \approx n^{1,585}$

C'est une application directe de la méthode générale de résolution des équations récurrentes (voir transparent 36 du cours, avec $a=3, b=2, f(n) = O(1)$).

On peut aussi développer la récurrence :

$$T(n) = 3 T(n/2) = 9 T(n/4) = 27 T(n/8) = \dots \text{ et ainsi de suite...}$$

A chaque fois que l'on divise n par 2, on multiplie le facteur qui est devant par 3. Combien de fois peut-on diviser n par 2 jusqu'à obtenir 1 ? la réponse est $\log_2 n$. Que vaudra alors le facteur devant $T(1)$? la réponse est : 3 à la puissance $\log_2 n$, c'est à dire environ $n^{1,585}$.

17. Application numérique : on veut multiplier deux nombres de $n=1000$ chiffres. Combien de multiplications élémentaires effectue-t-on avec les algorithmes des questions 7, 8 et 9, et combien de multiplications en effectue-t-on avec l'algorithme de Karatsuba ?

Avec l'algorithme de la question 7, la complexité est en $M \cdot \log_{10}(N)$. M est un nombre à 1000 chiffres (soit plus grand que 10^{1000}) et $\log_{10} N$ vaut 999. Le nombre d'opérations est donc un nombre à 1002 chiffres. L'âge de l'univers, exprimé en nombre de nano-secondes, est un nombre à 27 chiffres seulement. Il faut donc attendre très très très longtemps pour avoir le résultat de ce calcul en utilisant cet algorithme !

Avec l'algorithme de la question 8, on est en $\log_2(M) \cdot \log_{10}(N)$, soit environ $3,32 \cdot 999 \cdot 999 \approx 3.313.000$ opérations. Avec une machine qui effectue 10^9 multiplications élémentaires par seconde, cela mettrait 0,003 secondes.

Avec l'algorithme de la question 9, on est en $O(n^2)$ opérations avec $n = \log_{10} N$, donc 998000 opérations élémentaires. En théorie, on mettrait donc 0,001 secondes, mais on n'a pas tenu compte de la constante devant le $O(\)$, et on ne peut sans doute pas être aussi précis.

Avec l'algorithme de Karatsuba, la complexité est de $1000^{1,585} \approx 56885$. En supposant que la constante est la même, le temps passe à 0,00006 secondes (60 millièmes de seconde).

Les temps n'étaient pas demandés... c'est juste pour illustrer le fait que la complexité, ça peut être utile !

Les hypothèses de ce problème peuvent sembler farfelues : pourquoi supposer qu'un ordinateur ne sait pas faire les opérations élémentaires ? Elles sont en effet câblées dans les processeurs, et les multiplications et les additions peuvent être faites en temps constant. Oui mais : seulement avec une précision limitée ! (par exemple : 64 bits). Dès que l'on veut multiplier ou additionner des nombres de longueur arbitraire, il faudra utiliser des algorithmes similaires à ceux étudiés dans ce problème.