Algorithmique III. L2 Informatique I41.

TD 5. Calcul multiprécision [1]

EXERCICE 1. Soit k un entier naturel non-nul. On s'intéresse au produit de deux entiers naturels codés comme des entiers non-signés en machine sur des registres de 2^k bits.

(1) Quelle est l'entier *naturel* le plus grand que l'on peut représenter avec un registre de 64 bits ?

(2) Quelle est la condition sur les tailles de deux entiers A et B pour lesquels on peut réaliser l'opération de multiplication $A \times B$ en machine ?

On se propose d'écrire un algorithme qui calcule le produit $R := A.B$ de deux entiers non-signés A et B codés sur $n := 2^k$ bits en rangeant dans deux registres R_H et R_L les n bits de poids fort et les n bits de poids faible du résultat. On dispose des algorithmes suivants dont les paramètres sont toujours codés sur n bits (les exemples donnés le sont pour les valeurs $k = 2$, i.e. pour $n = 4$) :

- `Mul(X,Y)` : renvoie le produit $X \times Y$ mais en opérant uniquement sur les $\frac{n}{2}$ bits de poids faible de X et Y. Ex : $\texttt{Mul}(7, 10) = 6$.
- `Add(X,Y)` : renvoie la somme $X + Y$ mais en opérant uniquement sur les $n - 1$ bits de poids faible de X et Y. Ex : $\texttt{Add}(7, 13) = 12$.
- `H(X)` : renvoie l'entier dont les $\frac{n}{2}$ bits de poids faible sont ceux de poids fort de X et dont les $\frac{n}{2}$ bits de poids fort sont nuls. Ex : $\texttt{H}(12) = 3$.
- `L(X)` : renvoie l'entier dont les $\frac{n}{2}$ bits de poids faible sont ceux de X et dont les $\frac{n}{2}$ bits de poids fort sont nuls. Ex : $\texttt{L}(13) = 1$.
- `REG(X,Y)` : renvoie l'entier dont les $\frac{n}{2}$ bits de poids fort sont ceux de poids faible de X et dont les $\frac{n}{2}$ bits de poids faible sont ceux de poids faible de Y. Ex : $\texttt{REG}(7, 9) = 13$.

Le principe est le suivant : on coupe chaque opérande X en deux moitiés X_H et X_L sur $\frac{n}{2}$ bits :

$$\begin{aligned} A \times B &= (A_H 2^{\frac{n}{2}} + A_L)(B_H 2^{\frac{n}{2}} + B_L) \\ &= A_H B_H \, 2^n + A_H B_L \, 2^{\frac{n}{2}} + A_L B_H \, 2^{\frac{n}{2}} + A_L B_L. \end{aligned} \tag{1}$$

1. Version du 12 avril 2023, 23 : 00

Ces 4 produits sont réalisables sans débordement par l'algorithme `Mul(X,Y)`, il reste à déterminer comment obtenir le contenu des deux registres R_H et R_L sur n bits chacun.

(3) Écrivez les algorithmes `L(X)`, `H(X)` et `REG(X,Y)` à l'aide des opérateurs logiques bit à bit.

(4) Pour $k = 2$, c'est-à-dire quand les opérandes A et B sont codés sur $n = 4$ bits, représentez les contenus des 4 produits (cf. (1)) pour $A = B = 2^n - 1$ ainsi que ceux des registres R_H et R_L en binaire.

(5) Dressez une table des valeurs binaires des 4 termes de la somme (1) pour mettre en évidence le calcul à réaliser pour obtenir le contenu des registres R_H et R_L contenant respectivement la moitié haute et la moitié basse du produit $A\,B$. Généralisez à des opérandes A et B codés sur n bits pour écrire un algorithme qui calcule R_H et R_L.

Solution. **(1)** Il s'agit évidemment de $2^{64} - 1$.

(2) Il faut que le produit soit inférieur ou égal à cette valeur, on fixe généralement la taille de chacun des entiers à une valeur inférieure ou égale à $2^{32} - 1$. Il faut noter cependant que les processeurs modernes savent multiplier deux nombres de 64 bits, en codant le résultat dans deux registres, celui de poids faible est celui renvoyé par le calcul mais celui de poids fort existe mais n'est pas accessible directement, on y accède via un code en assembleur.

(3) On se contente de décrire les opérations bit à bit à réaliser, l'algorithme se limitant à renvoyer la valeur ainsi calculée :

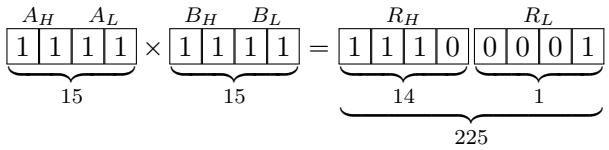
(1) `H(X)` : $\texttt{X} \gg (n \gg 1)$. On décale les bits de X de $\frac{n}{2}$ positions vers la droite.

(2) `L(X)` : $\texttt{X} \wedge ((\neg 0) \gg (n \gg 1))$. On masque les $\frac{n}{2}$ bits de poids fort de X.

(3) `R(X,Y)` : $(\texttt{L(X)} \ll (n \gg 1)) \vee \texttt{L(Y)}$. On décale les bits de X de $\frac{n}{2}$ positions vers la gauche et on rajoute les $\frac{n}{2}$ de poids faible de Y.

(4) On a $A = B = 15$, ce qui nous donne $A_H = A_L = B_H = B_L = 3$ et ainsi les 4 produits sont tous égaux à 9, soit 1001 en binaire. On résume

les différents calculs ci-dessous :



(5) Il va falloir réaliser quelques additions sur des opérandes de $\frac{n}{2}$ bits pour obtenir successivement les demi-registres de poids faibles et forts de chacun des registres R_L et R_H. L'équation (1) nous indique comment construire la table 1.

	retenues	0	1	0	1	0	0		
HH	$A_H B_H\, 2^n$	1	0	0	1				
HL	$A_H B_L\, 2^{\frac{n}{2}}$			1	0	0	1		
LH	$A_L B_H\, 2^{\frac{n}{2}}$			1	0	0	1		
LL	$A_L B_L$					1	0	0	1
	+	1	1	1	0	0	0	0	1
	$A \times B$	R_H				R_L			

Table 1. Contenus des différents registres.

Comme on peut l'observer dans cette table 1, la moitié faible de R_L est égale à la moitié faible du produit $A_L B_L$, les autres additions étant regroupées par couleur dans la table. L'algorithme s'en déduit directement. Les variables auxiliaires `G` et `D` serviront à construire respectivement la moitié gauche et la moitié droite du registre R_L puis celles du registre R_H. Notons qu'après la première addition la partie gauche de cette variable contient la retenue éventuelle.

```
ALGORITHME MMUL (A,B):couple d'entiers
DONNEES
 · A,B: entiers
VARIABLES
 · HH,HL,LH,LL,RH,RL,G,D: entiers
DEBUT
 · HH ← MUL(H(A),H(B))
 · HL ← MUL(H(A),L(B))
 · LH ← MUL(L(A),H(B))
 · LL ← MUL(L(A),L(B))
 · D ← L(LL)
 · G ← ADD(L(HL),ADD(L(LH),H(LL)))
 · RL ← REG(L(G),D)
 · D ← ADD(L(HH),ADD(H(HL),ADD(H(LH),H(G))))
 · G ← ADD(H(HH),H(D))
 · RH ← REG(L(G),D)
 · RENVOYER (RH,RL)
FIN
```

EXERCICE 2. Soit n un entier naturel. On veut estimer le nombre de chiffres nécessaires pour représenter n en base 2 (asymptotiquement).

(1) À l'aide de la formule de Stirling ci-dessous :

$$\lim_{n\to+\infty} \frac{n!}{\sqrt{2\pi n}(n/e)^n} = 1, \tag{2}$$

donnez une estimation asymptotique du nombre de chiffres nécessaires pour représenter la factorielle $n!$ d'un entier naturel n en base 2.

(2) Même question, mais en comparant avec une somme intégrale.

Solution. **(1)** On rappelle que le nombre de chiffres nécessaires pour représenter un nombre n en base b est égal à $\lfloor \log_b n \rfloor + 1$. La fonction logarithme est continue sur $\mathbb{R}^+$ on peut donc commuter avec la limite :

$$\lim_{n\to+\infty} \log_2\left(\frac{n!}{\sqrt{2\pi n}(n/e)^n}\right) = 0$$

$$\text{donc} \quad \lim_{n\to+\infty} \log_2(n!) = \lim_{n\to+\infty} \underbrace{\log_2\left(\sqrt{2\pi n}(n/e)^n\right)}_{L}$$

Calculons l'expression L à droite de la dernière égalité :

$$\begin{aligned} L &= \log_2(\sqrt{2\pi n}) + n\,\log_2\left(\frac{n}{e}\right) \\ &= \frac{1}{2}\left(\log_2(2) + \log_2(\pi) + \log_2(n)\right) + n\,\left(\log_2(n) - \log_2(e)\right) \\ &= \underbrace{n\log_2(n)}_{f(n)} - \underbrace{\left(\log_2(e)n - \frac{\log_2(n)}{2} - \frac{1+\log_2(\pi))}{2}\right)}_{g(n)} \end{aligned}$$

On vérifie aisément que

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

On en conclut que L a le même comportement asymptotique que la fonction définie par $n \mapsto n\log_2(n)$. Le nombre de chiffres de $n!$ exprimé en base 2 est donc asymptotiquement égal à $n\log_2(n)$.

(2) Cette fois on écrit directement :

$$\log_2(n!) = \sum_{k=2}^{n} \log_2(k) = \frac{1}{\ln(2)} \sum_{i=2}^{n} \ln(i).$$

Or on peut encadrer la somme par une intégrale :

$$\int_1^n \ln(x)dx \leqslant \sum_{i=2}^{n} \ln(i) \leqslant \int_2^{n+1} \ln(x)dx.$$

On rappelle qu'une primitive de $\ln(x)$ est la fonction $x\ln(x) - x$ (sinon faire une intégration par partie avec $u'(x) = 1$ et $v(x) = \ln(x)$) :

$$[x\ln(x) - x]_1^n \leqslant \sum_{i=2}^{n} \ln(i) \leqslant [x\ln(x) - x]_2^{n+1}$$

$$n\ln(n) + \underbrace{(1-n)}_{a(n)} \leqslant \sum_{i=2}^{n} \ln(i) \leqslant n\ln(n+1) + \underbrace{\big(\ln(n+1) - (n+1) - 2\ln(2)\big)}_{b(n)}$$

Là encore, les fonctions $a(n)$ et $b(n)$ sont négligeables asymptotiquement par rapport aux fonctions $n\ln(n)$ et $n\ln(n+1)$ et bien sûr $n\ln(n) \sim n\ln(n+1)$. On en conclut que le nombre de chiffres de $n!$ exprimé en base 2 est asymptotiquement égal à $n\log_2(n)$.

EXERCICE 3. Écrivez un algorithme qui réalise la soustraction $A - B$ de deux entiers naturels en multiprécision et en supposant que $A \geqslant B$. Calculez sa complexité dans le meilleur des cas et dans le pire des cas.

Solution. On considère que A et B sont deux listes contenant les chiffres de l'écriture dans une base donnée des entiers correspondant, l'indexation commençant à 0 et correspondant au chiffre le *moins* significatif. Il faut initialiser une liste résultat R à la taille de la liste A.

```
ALGORITHME SOUSTRACTION(A,B,base):liste
DONNÉES
 · A,B: listes
 · base: entier
VARIABLES
 · i,retenue: entiers
 · R: liste
DEBUT
 · R ← Allouer(#A,0) // cellules initialisees a 0
 · retenue ← 0
 · i ← 0
 · TQ (i < #B) FAIRE
 ·  · SI ((B[i] + retenue) <= A[i]) ALORS
 ·  ·  · R[i] ← A[i] - (B[i] + retenue)
 ·  ·  · retenue ← 0
 ·  · SINON
 ·  ·  · R[i] ← (A[i] + base) - (B[i] + retenue)
 ·  ·  · retenue ← 1
 ·  · FSI
 ·  · i ← i + 1
 · FTQ
 · TQ (i < #A) FAIRE
 ·  · SI (retenue <= A[i]) ALORS
 ·  ·  · R[i] ← A[i] - retenue
 ·  ·  · retenue ← 0
 ·  · SINON
 ·  ·  · R[i] ← (A[i] + base) - retenue
 ·  ·  · retenue ← 1
 ·  · FSI
 ·  · i ← i + 1
 · FTQ
 · RENVOYER R
FIN
```

Si l'on note $n = |A|$ et $m = |B|$, la complexité dans le meilleur des cas est en $\Theta(m)$ puisque le nombre d'opérations est proportionnel à la longueur de B et dans le pire des cas en $\Theta(n)$ puisque le nombre d'opérations est alors proportionnel à la longueur de A.

EXERCICE 4. On considère deux entiers A et B de n chiffres en base b et on note $\ell := \frac{n}{2}$. On scinde A (resp. B) à l'aide de deux nombres de ℓ chiffres A_H et A_L (resp. B_H et B_L) :

$$A = A_H\, b^\ell + A_L \quad \text{et} \quad B = B_H\, b^\ell + B_L.$$

(1) Calculez les produits $A\,B$ et $(A_H + A_L)(B_H + B_L)$.

(2) À partir des résultats de la question (1), vérifiez que

$$A\,B = A_H B_H b^n + \big((A_H+A_L)(B_H+B_L) - (A_H B_H + A_L B_L)\big)b^\ell + A_L B_L \quad (3)$$

(3) En supposant que le nombre de multiplications pour calculer le produit de deux nombres à n chiffres est $P(n)$ et que le produit de deux nombres

à un chiffre a un coût constant de $\Theta(1)$ (c'est une recherche en table), exprimez la fonction P avec une relation de récurrence. Pour simplifier les calculs, on suppose que n est une puissance de 2. Indication : utilisez l'égalité (3).

(4) Montrez que $P(n) = \Theta(n^{\log_2(3)})$.

Solution. **(1)** On calcule les produits :

$$\begin{aligned} A\,B &= (A_H\,b^\ell + A_L)(B_H\,b^\ell + B_L) \\ &= A_H B_H\,b^n + (A_H B_L + A_L B_H) b^l + A_L B_L. \end{aligned}$$

Et $(A_H + A_L)(B_H + B_L) = A_H B_H + (A_H B_L + A_L B_H) + A_L B_L$

donc $(A_H B_L + A_L B_H) = (A_H + A_L)(B_H + B_L) - (A_H B_H + A_L B_L)$.

(2) Par conséquent

$$A\,B = A_H B_H b^n + \big((A_H + A_L)(B_H + B_L) - (A_H B_H + A_L B_L)\big) b^\ell + A_L B_L$$

ce qui permet de calculer le produit de deux nombres à n chiffres en effectuant uniquement 3 produits de nombres à $\frac{n}{2}$ chiffres (et 5 additions que l'on néglige).

(3) On déduit directement de l'égalité (3) :

$$P(n) = \begin{cases} 3P(\frac{n}{2}) & \text{si } n > 0 \\ \Theta(1) & \text{sinon} \end{cases}$$

(4) On obtient facilement $P(n) = 3^k\Theta(1) = \Theta(3^k)$ si $n = 2^k$. Mais $k = \log_2(n)$, donc

$$\begin{aligned} P(n) &= 3^{\log_2(n)} \\ &= \left(2^{\log_2(3)}\right)^{\log_2(n)} \\ &= \left(2^{\log_2(n)}\right)^{\log_2(3)} \\ &= n^{\log_2(3)} \end{aligned}$$

Comme $\log_2(3) \simeq 1,585$, cet algorithme a une complexité proche de $n\sqrt{n}$ ce qui améliore notablement le temps de calcul d'une multiplication par rapport à l'algorithme naïf. Il a été conçu par Anatolli Karatsuba en 1960 et porte son nom.