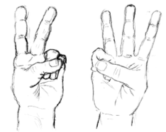
Algorithmique 1
Grands entiers

Hajer Akid & Sandrine Julia

Semestre 4

Compter

On peut supposer qu'après le développement du langage, les humains commencent à compter ...On peut également supposer que les doigts de la main constituent le boulier naturel



Remarque

- Le système décimal et duodécimal ne sont pas un hasard, 10 et 12 ont été les bases de la plupart des systèmes de comptage dans l'histoire
- Abaque du grec *Abacus* « table à poussière » est le nom générique donné à un instrument mécanique plan pour le calcul et le comptage.

Nombres modernes

Les nombres arabes (et le système de base à 10 chiffres) sont une représentation moderne (4^e siècle)

	0	1	2	3	4	5	6	7	8	9
Arabic	٠	١	٢	٣	٤	٥	٦	٧	٨	٩
Devanagari	०	१	२	३	४	५	६	७	८	९
Bengali	০	১	২	৩	৪	৫	৬	৭	৮	৯
Gurmukhi	੦	੧	੨	੩	੪	੫	੬	੭	੮	੯

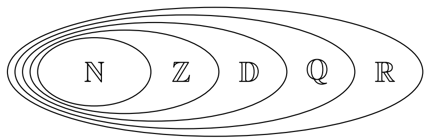
$$324 = \overset{10^2}{\downarrow}{3 \times 100} + \overset{10^1}{\downarrow}{2 \times 10} + \overset{10^0}{\downarrow}{4 \times 1}$$

unités
dizaines
centaines

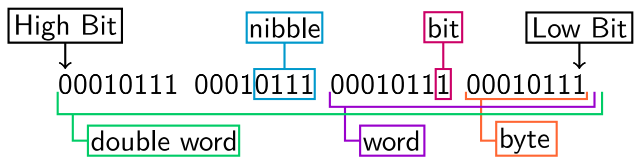
$$\lfloor \log_{10}(n) \rfloor + 1 = \lceil \log_{10}(n+1) \rceil$$

Mots de données

Les nombres machines sont des abstractions de nombres mathématiques :



Ils sont codés par des bits, un groupe/mots (finis) de bits dont la nature logique est compatible avec les traitements électroniques (on-off)



La représentation est finie, sur n bits, on ne peut représenter qu'un sous-ensemble contenant 2^n valeurs.

Encoder les nombres

La plupart des machines proposent 2 méthodes de codage bien connues.

Complément à deux pour les entiers signés

$$x = -b_{n-1} \times 2^n + \sum_{i=0}^{n-1} b_i \times 2^i$$

$$\begin{array}{ccc} \multicolumn{3}{c}{12_{10} - 5_{10} = 12_{10} + (-5_{10})} \\ & 0000\ 1100 & (12_{10}) \\ + & 1111\ 1011 & (5_{10}) \\ \hline = & 0000\ 0111 & (7_{10}) \end{array}$$

$$-2^{n-1} \rightarrow 2^{n-1} - 1$$

La norme IEEE 754 pour les flottants

$$x = (-1)^s \times 1.m \times 2^{(e-biais)}$$

- s : bit de signe (1 bit)
- m : mantisse (23 bits)
- e : exposant (8 bits)

$$1,175\ 494\ 35 \times 10^{-38} \rightarrow 3,402\ 823\ 46 \times 10^{38}$$

Représentation-machine limitée

La précision arithmétique fixée par la machine peut varier de 8 à 64 bits.

- $2^8 - 1 = 255$
- $2^{16} - 1 = 65\,535$
- $2^{32} - 1 = 4\,294\,967\,295 \approx 4,294 \times 10^9$
- $2^{64} - 1 \approx 1,844 \times 10^{19}$
- $2^{128} - 1 \approx 3,402 \times 10^{38}$

- $5\,! = 120$
- $8\,! = 40\,320$
- $12\,! = 479\,001\,600 \approx 4,790 \times 10^8$
- $20\,! \approx 2,432 \times 10^{18}$
- $34\,! \approx 2,952 \times 10^{38}$

- $F_{13} = 233$
- $F_{24} = 46\,368$
- $F_{47} \approx 2,971 \times 10^9$
- $F_{93} \approx 1,220 \times 10^{19}$
- $F_{186} \approx 3,328 \times 10^{38}$

Le problème d'échelle

- On doit garantir que l'entier dans une application spécifique ne provoque pas de débordement (*overflow*)
- De nouvelles applications nécessitent de sortir du champ de la machine et/ou du langage
  - nombre estimé de connexions neuronales dans le cerveau humain $10^{14} (\leq 2^{47})$
  - la masse de la Terre est constituée d'environ $4 \times 10^{51} (\leq 2^{172})$ nucléons
  - nombre estimé d'atomes dans l'univers observable $10^{80} (\leq 2^{266})$
  - limite inférieure estimée de la complexité de l'arbre de jeu des échecs (numéro de Shannon) $10^{120} (\leq 2^{399})$

Grands nombres

- L'arithmétique peut donc passer en *précision arbitraire* : **bignum**.
- Des langages de programmation ont de telles options intégrées : Lisp, Python, Perl, Haskell et Ruby
- D'autres comme C, C++ ou Java ont des bibliothèques disponibles pour les mathématiques entières et à virgule flottante de précision arbitraire

Plutôt que de stocker les valeurs sous forme d'un nombre fixe de bits binaires liés à la taille du registre du processeur, ces implémentations utilisent généralement des tableaux de chiffres de longueur variable :

- ✓ élimine les débordements simples
- ✓ garantit les résultats sur toutes les machines
- ✗ réduit les performances

Réduction des performances

- Le processeur est conçu pour traiter des instructions sur des entrées de la taille d'un registre
- Les entrées plus grandes doivent être divisées en morceaux de la taille d'un registre

Entiers

Taille

- Nombre N : $\mathrm{Size}(N) = O(1)$
- Tableau $A = [x_1, \ldots, x_k]$: $\mathrm{Size}(A) = O(k)$

Complexité

$\mathrm{Size}(x) = \mathrm{Size}(y) = O(1)$

Instruction	T	S
$x + y$	$O(1)$	$O(1)$
$x - y$	$O(1)$	$O(1)$
$x \times y$	$O(1)$	$O(1)$
$\frac{x}{y}$	$O(1)$	$O(1)$
$x \leq y$	$O(1)$	$O(1)$

Bignums

Taille

- Nombre N : $\mathrm{Size}(N) = O(\log(N))$
- Tableau $A = [x_1, \ldots, x_k]$: $\mathrm{Size}(A) = O(k \times \log(N))$

Complexité

$N = \max(x, y)$
$\mathrm{Size}(x) \leq \mathrm{Size}(N) = O(\log(N))$
$\mathrm{Size}(y) \leq \mathrm{Size}(N) = O(\log(N))$

Instruction	T	S
$x + y$	$O(\log(N))$	$O(\log(N))$
$x - y$	$O(\log(N))$	$O(\log(N))$
$x \times y$	$O(\log^2(N))$	$O(\log(N))$
$\frac{x}{y}$	$O(\log^2(N))$	$O(\log(N))$
$x \leq y$	$O(\log(N))$	$O(1)$

Bignums

Complexité

$N = \max(x, y)$
$\mathrm{Size}(x) \leq \mathrm{Size}(N) = O(\log(N))$
$\mathrm{Size}(y) \leq \mathrm{Size}(N) = O(\log(N))$

Instruction	T	S
$x \leq 0$	$O(\log(N))$	$O(1)$
$MULT2(x)$	$O(\log(N))$	$O(\log(N))$
$DIV2(x)$	$O(\log(N))$	$O(\log(N))$
$ODD(x)$	$O(1)$	$O(1)$
$EVEN(x)$	$O(1)$	$O(1)$

Multiplication d'entiers

Soient x et y deux entiers non négatifs tels que $x \leq 2^{16}$ et $y \leq 2^{16}$, on considère que les opérations se font en temps constant $O(1)$.

```
multiplication (entier x, entier y) {
  resultat ← 0
  tant que (x > 0) {
    si (x mod 2 = 1) {
      resultat ← resultat + y
    }
    y ← 2 * y
    x ← x div 2
  }
  retourner resultat
}
```

Complexité : $T(x) = O(log(x))$

Multiplication d'entiers

Les opérations sont en $O(1)$.

Complexité

$T(x) = T_1 + T_2 + T_7$
$T_1 = O(1)$
$T_7 = O(1)$
$T_2 = \Sigma_{i=1}^{\lfloor \log(x) \rfloor}(T_{cond2} + T_3 + T_5 + T_6)$
$T_{cond2} = O(1)$
$T_3 = T_{cond3} + T_4 = O(1) + O(1) = O(1)$
$T_5 = O(1)$
$T_6 = O(1)$
$T_2 = \Sigma_{i=1}^{\lfloor \log(x) \rfloor}(O(1) + O(1) + O(1) + O(1)) = \Sigma_{i=1}^{\lfloor \log(x) \rfloor} O(1) = O(\log(x))$
$T(x) = O(\log(x))$

Multiplication de grands entiers

Soient x et y deux **grands entiers** non négatifs de n bits.

```
multiplication (entier x, entier y) {
  resultat ← 0
  tant que (x > 0) {
    si (ODD(x)) {
      resultat ← resultat + y
    }
    MULT2 (y)
    DIV2 (x)
  }
  retourner resultat
}
```

Complexité : $T(n) = O(n^2)$

Multiplication de grands entiers

Soient x et y deux **grands entiers** non négatifs de n bits : les opérations ne sont plus en temps constant.

Complexité

$T(n) = T_1 + T_2 + T_7$
$T_1 = O(1)$
$T_7 = O(\log(xy)) = O(log(x)) + O(log(y)) = O(n+n) = O(n)$
$T_2 = \Sigma_{i=1}^{\lfloor \log(x) \rfloor}(T_{cond2} + T_3 + T_5 + T_6)$
$T_{cond2} = O(\log(x)) = O(n)$
$T_3 = T_{cond3} + T_4$
$T_{cond3} = O(1)$
$T_4 = O(n) + O(\log(r) + \log(y')) = O(\log(y'))$ car $r \leq y'$ avec $y' \approx 2^i y$
$T_4 = O(\log(2^i y')) = O(i) + O(\log(y') = O(i) + O(n)$
$T_3 = O(i) + O(n)$
$T_5 = O(\log(y)) = O(n)$
$T_6 = O(\log(x)) = O(n)$
$T_2 = \Sigma_{i=1}^{\lfloor \log(x) \rfloor}(O(n) + O(i) + O(n) + O(n) + O(n)) = \Sigma_{i=1}^{\lfloor \log(x) \rfloor} O(n)$ car $i \leq n$
$= \log(x)O(n) = O(n^2)$
$T(n) = O(n^2)$

Complexité en espace

Algorithme non récursif

Soient $X = \{x_1, x_2, \ldots, x_k\}$ les (nouvelles) variables de l'algorithme :

$$S = \sum_{i=1}^{k} \mathrm{Space}(x_i)$$

Avec $\mathrm{Space}(x) = max(\mathrm{Size}(v))$ où v est n'importe quelle valeur stockée dans la variable x

Complexité en espace

Algorithme récursif

Soit h la hauteur de l'arbre de récursion, f_i les appels récursifs

$$S = \sum_{i=1}^{h} \mathrm{Space}(f_i)$$

Avec $\mathrm{Space}(f) = max(\mathrm{Size}(v))$ où v est n'importe quelle valeur stockée pour l'appel f