Arithmétique des entiers

On résume les complexités des opérations élémentaires sur les entiers, qui vont être établies dans les exercices qui suivent.

Algorithme	complexité
$m \pm n$	$O(\log(m) + \log(n))$
mn	$O(\log(m) \log(n))$
$n \bmod m$	$O(\log(m) \log(n/m))$

On pose $x = \lfloor \log_2(m) \rfloor + \lfloor \log_2(n) \rfloor + 2$, la taille d'entrée pour ces algorithmes.

Remarque. La complexité de la multiplication est donnée pour un algorithme de multiplication naïve; on peut faire mieux. Dans la suite, on va détailler un algorithme (de Karatsuba) qui permet de multiplier des entiers en temps $O(x^\omega)$ avec

$$\omega = \log_2(3) \simeq 1.5849625 \dots < 2.$$

La complexité $O(\log(n) \log(n/m))$ pour la réduction modulaire $n \bmod m$ est donnée dans Demazure [1, p.30], mais comme on peut supposer $\log(m) < \log(n)$ (sinon le temps de calcul est nul), la complexité $O(\log(m) \log(n/m))$ est meilleur; voir les exercices **7** et **8** ci-dessus. En particulier, on note que la complexité en termes de la taille d'entrée x est linéaire pour addition, et quadratique pour multiplication (naïve) et réduction modulaire (dans le pire cas où $\log(n) = c \log(m)$ avec $c > 1$).

Exercices

1. Soit $m = 100110111_2$ et $n = 10111_2$ (en représentation binaire). Trouver $m + n$ en précisant les étapes pour leur addition.
2. Répéter l'exercice précédent avec $m = 2^{16} - 1$ et $n = 1$ (cf. **3.b.** ci-dessous).
3. Exprimer la complexité de l'addition de m et n en fonction de la taille x des données (avec la notation $O(f(x))$), en particulier, démontrer que cette complexité est $O(x)$.
  - a. Plus précisément, décrire un algorithme et compter le nombre d'opérations binaires en termes de $r = \lfloor \log_2(m) \rfloor$ et $s = \lfloor \log_2(n) \rfloor$, pour le problème suivant :
Entrées : m et n entiers (donc taille d'entrée $x = r + s + 2$)
Sortie : $m + n$
  - b. La syntaxe $m += n$ s'utilise pour un algorithme qui remplace m avec la valeur $m + n$. Supposer que m , n , et $m + n$ s'écrivent en représentation binaire. Est-il possible de décrire un algorithme pour $m += n$ avec complexité $O(\log(n))$?
4. Pour des polynômes f et g dans $\mathbb{F}_2[x]$, montrer qu'il existe un algorithme pour le calcul $f += g$ dans la classe $O(\deg(g))$.

5. Compter le nombre d'opérations pour la multiplication en utilisant la formule

$$\left(m = \sum_{i=0}^r m_i 2^i, n = \sum_{j=0}^s n_j 2^j \right) \mapsto mn = \sum_{i=0}^{r+s} \left(\sum_{j=0}^i m_{i-j} n_j \right) 2^i$$

en termes de $r = \lfloor \log_2(m) \rfloor$ et $s = \lfloor \log_2(n) \rfloor$.

Indication. Les sommes internes correspondent aux sommes des colonnes :

$$\begin{array}{cccccccccccc} n_0 \times & 0 & \cdots & & \cdots & 0 & 0 & m_r & \cdots & m_1 & m_0 \\ n_1 \times & 0 & \cdots & & \cdots & 0 & m_r & \cdots & m_1 & m_0 & 0 \\ \vdots & \vdots & & & \ddots & \ddots & & & \ddots & \ddots & \vdots \\ n_j \times & 0 & & 0 & m_r & \cdots & m_1 & m_0 & 0 & & 0 \\ & & & & & & & & & & \\ \vdots & \vdots & \ddots & \ddots & & \ddots & \ddots & & & & \vdots \\ n_{s-1} \times & 0 & m_r & \cdots & m_1 & m_0 & 0 & \cdots & & \cdots & 0 \\ n_s \times & m_r & \cdots & m_1 & m_0 & 0 & 0 & \cdots & & \cdots & 0 \end{array}$$

Quel est le nombre maximal de termes non nuls dans une colonne ?

6. Démontrer que la complexité de la multiplication de deux nombres n, m de tailles dans $O(x)$, est dans $O(x^2)$, en utilisant la multiplication naïve avec retenues.

Indication. Montrer qu'on obtient la complexité $O(\log(m) \log(n)) \leq O(x^2)$ en sommant les lignes de la matrices ci-dessus.

7. Décrire un algorithme pour la réduction $(n, m) \mapsto n \bmod m$ avec complexité

$$O(\log(m) \log(n/m)).$$

Indication. Étant donné $n = n_s \dots n_1 n_0$ et $m = m_r \dots m_1 m_0$, le premier étape est la soustraction $n \leftarrow 2^{s-r} m$:

$$\begin{array}{cccccccc} n_s & n_{s-1} & \cdots & n_{s-r} & n_{s-r-1} & \cdots & n_0 \\ m_r & m_{r-1} & \cdots & m_0 & 0 & \cdots & 0 \\ \hline 0 & * & \cdots & * & n_{s-r-1} & \cdots & n_0 \end{array}$$

ou $n \leftarrow 2^{s-r-1} m$:

$$\begin{array}{cccccccc} n_s & n_{s-1} & \cdots & n_{s-r-1} & n_{s-r-2} & \cdots & n_0 \\ 0 & m_r & \cdots & m_0 & 0 & \cdots & 0 \\ \hline 0 & * & \cdots & * & n_{s-r-2} & & n_0 \end{array}$$

Le nombre de coordonnées changées est au maximum $r+2$, et le résultat a au moins un bit de moins. En itérant, montrer qu'on obtient, après $O(s-r) = O(\log(n/m))$ étapes, la complexité $O(r(s-r)) = O(\log(m) \log(n/m))$.

8. Décrire un algorithme pour la fonction $n \mapsto n \bmod 2$ et comparer sa complexité avec l'algorithme de l'exercice précédent pour n grand.

9. Montrer que les deux algorithmes suivants calculent le même résultat avec la même complexité. Décrire les résultats intermédiaires dans les deux cas.

```
def exp_mod_L(a,k,n):          def exp_mod_R(a,k,n):
    # Données d'entree:        # Données d'entree:
    # entiers a, k, n > 0      # entiers a, k, n > 0
    # Sortie: a^k mod n        # Sortie: a^k mod n
    b = 1                      b = 1
    k_seq = k.bits()           k_seq = k.bits(); k_seq.reverse()
    for t in k_seq:            for t in k_seq:
        if t == 1:              b = (b**2) % n
            b = (b * a) % n      if t == 1:
            a = (a**2) % n        b = (b * a) % n
    return b                    return b

exp_mod_L(3,101,101) == 3      exp_mod_R(3,101,101) == 3
```

Références

- [1] M. Demazure, *Cours d'algèbre, primalité, divisibilité, codes*, Cassini, 1997.
- [2] M. Hindry, *Arithmétique*, Calvage & Mounet, 2008.
- [3] V. Shoup, *A Computational Introduction to Number Theory and Algebra*, Cambridge University Press, 2005.
- [4] R. P. Brent, P. Zimmermann, *Modern Computer Arithmetic*, Cambridge University Press, 2010.