Aix-Marseille Université
Préparation à l'agrégation: Option C
`http://iml.univ-mrs.fr/~kohel/tch/Agreg/`

Arithmétique des entiers

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 précisement, 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 $m \mathrel{+}= n$ 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 $m \mathrel{+}= n$ 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 $f \mathrel{+}= g$ dans la classe $O(\deg(g))$.

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

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

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

Indication. Les sommes internes correspond aux sommes des colonnes :

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

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 reduction $(n, m) \mapsto n \bmod m$ avec complexité

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

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

n_s	n_{s-1}	$\dots$	n_{s-r}	n_{s-r-1}	$\dots$	n_0
m_t	m_{r-1}	$\dots$	m_0	0	$\dots$	0
0	$*$	$\dots$	$*$	n_{s-r-1}	$\dots$	n_0

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

n_s	n_{s-1}	$\dots$	n_{s-r-1}	n_{s-r-2}	$\dots$	n_0
0	m_r	$\dots$	m_0	0	$\dots$	0
0	$*$	$\dots$	$*$	n_{s-r-2}		n_0

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 intérmediares dans les deux cas.

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

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

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

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