TD 2 – Nombres Binaires, Algorithme de Karatsuba

Structures de données (IF 122)

But de l'algorithme : multiplier 2 grands entiers de longueurs sensiblement égales[1]. Nous travaillerons en base 2, puisque c'est ainsi que les nombres sont représentés dans la mémoire d'un ordinateur. On note $|E|$ la longueur de l'entier E.

Pour les exercices marqués TP, vous trouverez une liste de commandes Java à la fin de l'énoncé.

Exercice 1 — *TD* Ecrire le nombre décimal 122 en base 2. Quelle est sa longueur ?

Exercice 2 — *TD* Quel valeur décimale représente le nombre binaire 100010 ?

Exercice 3 Faire le produit des deux nombres binaires précédents.

On considère deux nombres binaires E et F, de longueurs sensiblement égales. On note cette longueur n. Si $e_0, e_1, \dots e_{n-1}$ sont les chiffres de E (e_0 étant le chiffre des unités) et $f_0, f_1, \dots f_{n-1}$ ceux de F, on fait alors le découpage suivant : :

$$E = \underbrace{\boxed{e_{n-1}}\boxed{\dots}\boxed{e_{n/2}}}_{E_1}\ \underbrace{\boxed{e_{n/2-1}}\boxed{\dots}\boxed{e_0}}_{E_0}$$

$$F = \underbrace{\boxed{f_{n-1}}\boxed{\dots}\boxed{f_{n/2}}}_{F_1}\ \underbrace{\boxed{f_{n/2-1}}\boxed{\dots}\boxed{f_0}}_{F_0}$$

qui est tel que $E = E_1 \cdot 2^{n/2} + E_0$ et $F = F_1 \cdot 2^{n/2} + F_0$ (on considère la partie entière de $n/2$, pour que le cas où n est impair ne pose pas de problème).

Multiplication, la méthode naïve.

L'algorithme récursif naïf revient à écrire :

$$E \cdot F = E_1 \cdot F_1 \cdot 2^n + (E_0 \cdot F_1 + E_1 \cdot F_0) \cdot 2^{n/2} + E_0 \cdot F_0. \quad (1)$$

On voit qu'à chaque étape de la récursion, on effectue 4 multiplications d'entiers de longueur $n/2$.

Exercice 4 — *TD* On note $T(n)$ le temps utilisé pour multiplier 2 nombres binaires ayant n chiffres par l'algorithme naïf. On suppose qu'il existe une constante C telle que la somme de deux nombres binaires de taille n se fait en temps $C \cdot n^2$ (on ne tient pas compte du temps mis pour la multiplication d'un nombre par une puissance de 2). Calculer (par récurrence) $T(n)$ quand $n = 2^m$.

Exercice 5 — *TD+TP* Ecrire une fonction **récursive** qui prend en argument 2 entiers longs `E` et `F` et un entier `n` tel que `n` est supérieur ou égal à `|E|` et à `|F|` (on ne demande pas que la fonction vérifie cette propriété) et qui renvoie le produit de `E` et `F`, calculé par la méthode naïve.

L'algorithme de Karatsuba.

L'algorithme de Karatsuba repose sur le fait suivant :

$$E_0 \cdot F_1 + E_1 \cdot F_0 = (E_0 + E_1) \cdot (F_0 + F_1) - E_0 \cdot F_0 - E_1 \cdot F_1.$$

[1] La longueur d'un entier est son nombre de chiffres dans la base considérée.

[2] Cette approximation devient réaliste quand n est suffisemment grand, à partir de $n = 1000$ environ.

L'équation (1) peut donc s'écrire

$$E\cdot F = 2^n\cdot m_1 + 2^{n/2}\cdot(m_2 - m_0 - m_1) + m_0, \text{ où } \begin{cases} m_0 = E_0\cdot F_0, \\ m_1 = E_1\cdot F_1, \\ m_2 = (E_0+E_1)\cdot(F_0+F_1). \end{cases}$$

Ce qui amène à faire seulement 3 multiplications (à comparer avec les 4 multiplications de la méthode naïve).

Exercice 6 — *TD* On note $T(n)$ et C les mêmes valeurs que dans l'exercice précédent mais en utilisant l'algorithme de Karatsuba. Calculer (par récurrence) le temps mis pour effectuer le produit de 2 nombres binaires E et F ayant $n = 2^m$ chiffres.

Exercice 7 — *TD+TP* Ecrire une fonction **récursive** qui prend en argument 2 entiers longs `E` et `F` et un entier `n` tel que `n` est supérieur ou égal à |`E`| et à |`F`| (on ne demande pas que la fonction vérifie cette propriété) et qui renvoie le produit de `E` et `F`, calculé par l'algorithme de Karatsuba.

Exercice 8 — *TD+TP* Tester les fonctions écrites dans les exercices précédents en écrivant un programme complet. Vous pourrez notamment vérifier que les 2 méthodes donnent bien le même résultat.

Commandes Java – opérateurs de décalage

Les deux opérateurs de décalage sont le décalage à droite et le décalage à gauche. Il s'agit de décalages binaires. Ils sont quasi instantanés en Java. Regardons sur un exemple :

```
int x, y, z;

x = 37;       /* en binaire : 100101 */
y = x << 2;   /* en binaire : 10010100 */  /* décalage a gauche */
z = x >> 2;   /* en binaire : 1001 */      /* décalage a droite */

Deug.println("x = " + x);
Deug.println("y = " + y);
Deug.println("z = " + z);
```

Après compilation, son exécution donne :

```
x = 37    y = 148     z = 9
```

Il faut se méfier de certaines limitations, en particulier un entier (même long) a un nombre limité de chiffres. Si on dépasse ce nombre, on repart de "l'autre côté". Ainsi, quand on utilise des décalages à gauche, on risque d'avoir des résultats bizarres si l'on ne fait pas attention.

L'intérêt des décalages pour notre problème est assez évident, puisque multiplier un nombre par 2^n revient à le décaler à gauche de n.

Correction

Exercice 9 122 s'écrit 1111010 en binaire.

Exercice 10 100010 en binaire vaut 34

Exercice 11 Le produit des deux nombres en binaire vaut 1000000110100, soit 4148.

Exercice 12

$$T(2^m) = 4T(2^{m-1}) + 3C2^m$$
$$= 4(4T(2^{m-2}) + 3C2^{m-1})) + 3C2^m$$
$$= 16T(2^{m-2}) + 2 * 3C2^m + 3C2^m$$
$$= 3C2^m(1 + 2 + 4 + ... + 2^m)$$
$$= 3C2^m(2^{m+1} - 1)$$

Exercice 13

```
public static long multiplication(long E, long F, int n){
long e0,e1,f0,f1,m0,m1,m2,res;

if (n==1) return E*F;

else{
e1 = E >> (n/2);
e0= E - (e1 << (n/2));
f1 = F >> (n/2);
f0 = F - (f1 << (n/2));
m0 = multiplication(e0,f0,n/2);
m1 = multiplication(e1,f1,n/2);
m2 = multiplication (e0, f1, n/2) + multiplication(e1,f0,n/2);
res = m0 + (m1 << (2 *(n/2))) +(m2 << (n/2));
//Deug.println(res);
return res;
}
}
```

Exercice 14

$$T(2^m) = 3T(2^{m-1}) + 4C2^m + 2C2^{m-1}$$
$$= 3T(2^{m-1}) + 5C2^m$$
$$= 3(3T(2^{m-2})5C2^{m-1}) + 5C2^m$$
$$= 9T(2^{m-2}) + (3/2)5C2^m + 5C2^m$$
$$= 5C2^m(1 + 3/2 + 9/4 + ... + (3/2)^m))$$
$$= 5C2^m \frac{1 - (3/2)^{m+1}}{1 - 3/2}$$
$$= 5C2^{m+1}((3/2)^{m+1} - 1)$$

$$(3/2)^m = 2^{mlog_2(3/2)} = n^{log_2(3/2)} \sim n^{0.58}$$

Exercice 15

```
import fr.jussieu.script.*;

public class Karatsuba {

    public static long karatsuba(long E, long F, int n){
    long e0,e1,f0,f1,m0,m1,m2,res;

    if (n==1) return E*F;
    else{
    e1 = E >> (n/2);
    e0= E - (e1 << (n/2));
    f1 = F >> (n/2);
    f0 = F - (f1 << (n/2));
    m0 = karatsuba(e0,f0,n/2);
    m1 = karatsuba(e1,f1,n/2);
    m2 = karatsuba (e0+e1, f0+f1, n/2);
    res = m0 + (m1 <<(2*(n/2))) +((m2-m0-m1) << (n/2));

    return res;
    }
}

    /* Programme principal */
    public static void main (String[] args) {
        Deug.println("Entrez deux nombres sous forme décimale : ");
        long E = Deug.readLong();
        long F = Deug.readLong();

        Deug.println("majoration du nombre d'octets ");
    int l = Deug.readInt();
    long mul = karatsuba(E,F,l);
    long mul1 = multiplication(E,F,l);


        Deug.println("Le résultat de leur multiplication : " + mul1);
        Deug.println("Le résultat de leur multiplication Karatsuba: " + mul);
    }
}
```