

# Project Calculabilité, Complexité et évaluation de performances

## 1 Partie commune : multiplication d'entiers multiprécision positifs

L'objectif de cette partie est de proposer un algorithme de multiplication des entiers de tailles arbitraires. On considère deux entiers  $N = \sum_{i=0}^n a_i 2^i$  et  $M = \sum_{i=0}^m b_i 2^i$  représentés en base 2 (ici  $n = \lceil \log_2(N) \rceil$  et  $m = \lceil \log_2(M) \rceil$ ).

Ici, par abus de notation les entiers  $N$  et  $M$  sont confondus avec leurs représentations sous forme de liste de bits (leur représentation en base 2), i.e.  $N = [a_0, \dots, a_n]$  et  $M = [b_0, \dots, b_m]$ .

**Attention ! Ici les développements binaires des entiers sont écrits de gauche à droite (le bit de poids faible est à gauche) contrairement à la convention dans les machines où les bits de poids faible sont à droite. Donc pour ceux qui auront à programmer, il faudra faire attention.**

### 1.1 Addition binaire

Pour additionner deux entiers représentés sous forme de liste de bits d'entiers on considère l'algorithme suivant (où  $\vee$  est le ou exclusif et  $\wedge$  le et logique) :

Algorithme [Addition]

- **Entrées** :  $N = [a_0, \dots, a_n]$  et  $M = [b_0, \dots, b_m]$
- retenue  $\leftarrow 0$ ;
- **Si**  $n \geq m$  **alors** (1)
  - resultat  $\leftarrow [a_0, \dots, a_n, 0]$ ;
  - **Pour**  $i$  allant de 0 jusqu'à  $m$  faire
    - **Si** retenue = 0 **alors** (2)
      - $r_i \leftarrow r_i \vee b_i$ ;
      - retenue  $\leftarrow a_i \wedge b_i$ ;
      - **Sinon**
        - **Si**  $a_i \vee b_i = 1$  **alors** (3)
          - $r_i \leftarrow 0$ ;
          - retenue  $\leftarrow 1$ ;
        - **Sinon**
          - $r_i \leftarrow 1$ ;
          - retenue  $\leftarrow a_i \wedge b_i$ ;
      - fin du Si** (3)
    - fin du Si** (2)
  - fin du Pour**
  - **Pour**  $i$  allant de  $m+1$  jusqu'à  $n$  faire
    - $r_i \leftarrow r_i \vee \text{retenue}$ ;
    - retenue  $\leftarrow a_i \wedge \text{retenue}$ ;
  - fin du Pour**
  - $r_{n+1} \leftarrow \text{retenue}$ ;
  - **Retourner** resultat;
- **Sinon**
  - Retourner** Addition( $M, N$ );
- fin du Si** (1)



En C++ en utilisant la `std` on peut faire un objet contenant un champs `list<unsigned int>` par exemple et implanter les les méthodes de récupération d'un bit ...

Implanter l'addition, la soustraction et la multiplication par les deux méthodes précédentes. Comparer les deux produits en regardant les courbes expérimentales de temps de calcul des deux algorithmes. Expliquer la démarche expérimentale que vous choisissez et en quoi vos expériences infirment ou confirment la théorie. Vous m'envoyez le code par email, pas de listing dans le rapport s'il-vous-plait.

### 3 Partie pour les mathématiciens : produit de matrices d'entiers positifs

On suppose que la multiplication de deux entiers de taille  $\tau$  se fait en  $M(\tau)$  opérations. On considère deux matrices  $M$  et  $N$  carrées de taille  $n$  et dont toutes les entrées sont de taille  $\tau$ .

1. En utilisant la formule du produit naïf de matrices (celui que vous connaissez) donner une borne sur le nombre produit nécessaire pour le calcul d'une entrée de  $M * N$  et sur la taille des entrées du produit  $M * N$  (on estime ici que les addition en coûte rien comparées aux multiplications).
2. On suppose que  $n = 2 * l$  et on écrit :

$$N = \begin{pmatrix} N_{1,1} & N_{1,2} \\ N_{2,1} & N_{2,2} \end{pmatrix} \text{ et } M = \begin{pmatrix} M_{11} & M_{1,2} \\ M_{2,1} & M_{2,2} \end{pmatrix}.$$

Ici  $N_{i,j}$  et  $M_{i,j}$  sont carrées de taille  $l$ . Combien d'opérations effectuerait un algorithme qui calculerait  $N * M$  en utilisant la formule suivante :

$$\begin{pmatrix} M_{1,1} * N_{1,1} + M_{1,2} * N_{2,1} & M_{1,1} * N_{1,2} + M_{1,2} * N_{2,2} \\ M_{2,1} * N_{1,1} + M_{2,2} * N_{1,2} & M_{2,1} * M_{1,2} + M_{2,2} * N_{2,2} \end{pmatrix}.$$

3. Avec les mêmes notations que la question précédente, on définit les matrices suivantes :

$$\begin{aligned} S_1 &= M_{2,1} + M_{2,2}; & S_2 &= S_1 - M_{1,1}; \\ S_3 &= M_{1,1} - M_{2,1}; & S_4 &= M_{1,2} - S_2; \\ S_5 &= N_{1,2} - N_{1,1}; & S_6 &= N_{2,2} - S_5; \\ S_7 &= N_{2,2} - N_{1,2}; & S_8 &= S_6 - N_{2,1}; \end{aligned}$$

Puis :

$$\begin{aligned} P_1 &= S_2 * S_6; & P_2 &= M_{1,1} * N_{1,1}; \\ P_3 &= M_{1,2} * N_{2,1}; & P_4 &= S_3 * S_7; \\ P_5 &= S_1 * S_5; & P_6 &= S_4 * N_{2,2}; \\ & & P_7 &= M_{2,2} * S_8; \end{aligned}$$

et enfin :

$$\begin{aligned} S_9 &= P_3 + P_2; & S_{10} &= P_1 + P_2; \\ S_{11} &= S_{10} + P_4; & S_{12} &= S_{10} + P_5; \\ S_{13} &= S_{12} + P_6; & S_{14} &= S_{11} - P_7; \\ & & S_{15} &= S_{11} + P_5. \end{aligned}$$

Montrer que  $M * N = \begin{pmatrix} S_9 & S_{13} \\ S_{14} & S_{15} \end{pmatrix}$ .

4. Pour les très courageux : cet algorithme améliore-t-il les précédents ?