## Algorithmique III. L2 Informatique I41.

TD 4. Square & Multiply et factorisation de Hörner [1]

**EXERCICE 1.** Écrivez l'algorithme d'exponentiation naïf sur la machine RAM, en supposant qu'un registre peut stocker des valeurs arbitrairement grandes. Calculez le nombre $C(n)$ d'instructions décodées par cet algorithme en fonction de $n$. On suppose que la première valeur sur la bande d'entrée est le nombre $x$ à exponentier et le second l'exposant $n$. L'hypothèse est-elle réaliste sur une machine physique et peut-on considérer que la fonction $I$ est la fonction de complexité de l'algorithme ?

**Solution.** On range $x$ dans le registre $R_1$ et $n$ dans le registre $R_2$ qui sera décrémenté au fur et à mesure du calcul. Les puissances successives de $x$ sont rangées dans le registre $R_3$ initialisé à 1 puisque par convention $x^0 = 1$ :

```
 00 |   READ     ; ACC ← ENTREE[I++]
 01 |   STORE 1  ; R[1] ← ACC (ENTREE X)
 02 |   READ     ; ACC ← ENTREE[I++]
 03 |   STORE 2  ; R[2] ← ACC (EXPOSANT N)
 04 |   LOAD #1  ; ACC ← 1
 05 |   STORE 3  ; R[3] ← ACC (INITIALISATION RES.)
>06 |   LOAD 2   ; ACC ← R[2]
 07 |   JUMZ 13  ; SI R[2]=0 ALORS SAUTER A  #13 (FIN)
 08 |   LOAD 1   ; SINON ACC ← R[1] (ON CHARGE X)
 09 |   MUL 3    ; ACC ← ACC * R[3]}
 10 |   STORE 3  ; R[3] ← ACC
 11 |   DEC 2    ; R[2] ← R[2] - 1
 12 |   JUMP 6   ; REVENIR A #06
>13 |   LOAD 3   ; CHARGEMENT R[3]
 14 |   WRITE    ; AFFICHAGE RESULTAT
 15 |   STOP     ; ARRET
```

Les instructions d'initialisation (#00 à #05) et d'affichage du résultat (#13 à #15) ne sont exécutées qu'une seule fois. Le chargement #06 et le test de boucle #07 sont effectués $n+1$ fois alors que les instructions de mise à jour #08 à #12 le sont $n$ fois. On a

$$C(n) = 9 + 2(n+1) + 5n = 7n + 11.$$

1. Version du 13 avril 2023, 07 : 22

L'hypothèse n'est pas réaliste car les opérations arithmétiques sur une machine physique ne sont valides que sur des registres de taille bornée, par exemple 64 bits. Autrement dit le coût unitaire d'une multiplication n'a de sens que si les opérandes sont bornés, le nombre d'instructions décodées $C(n)$ ne peut donc mesurer la complexité d'un algorithme de multiplication réel. Il faut d'ailleurs noter que la fonction de complexité dépend de la *taille* des données, or ici avec cette hypothèse la taille des données est fixe égale à 2, alors que sur une machine physique il faudrait considérer le nombre de registres nécessaires pour encoder $x$ et l'exposant $n$.

**EXERCICE 2.** On appelle *chaîne d'additions* de longueur $\ell$ toute séquence $(a_k)_{k\in[\![0,\ell]\!]}$ strictement croissante de $\ell+1$ entiers naturels telle que $a_0 = 1$ et telle que chaque entier de la séquence est la somme de deux valeurs précédentes quelconques de cette séquence.

**(1)** Définissez une chaîne d'additions en logique des prédicats.

**(2)** Écrivez l'algorithme `DoubleAdd` qui est l'adaptation additive de l'algorithme `SquareMultiply` pour un monoïde $(X, +)$ où l'on a simplement remplacé la loi de composition multiplicative par une loi addive.

**(3)** Soit $(X, \times)$ un monoïde. Montrez que le nombre minimum de multiplications requises pour calculer $x^n$ est la longueur minimale d'une chaîne d'additions $(a_i)_{i\in[\![0,\ell]\!]}$ telle que $a_\ell = n$.

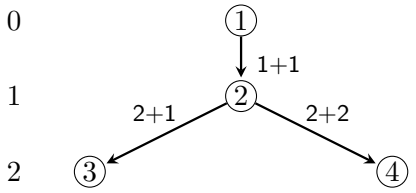


FIGURE 1. Arbre des chaînes d'additions de longueur $k = 2$.

**(4)** On construit inductivement l'arbre des chaînes d'additions de la manière suivante : la racine contient la valeur $a_0 = 1$ et on crée en chaque nœud de l'arbre autant de fils que de nouvelles valeurs obtenues en additionnant deux valeurs quelconques sur le chemin qui le relie à la racine.

Construisez l'arbre partiel de profondeur $k := 4$ (La figure 1 montre cet arbre partiel pour la profondeur $k = 2$).

**(5)** Vérifiez que le nombre de fils d'un nœud de profondeur $k \geqslant 1$ est minoré par $k + 1$ et majoré par $(k + 1)(k - 2)/2$. Est-il envisageable de chercher une chaîne d'addition minimale pour atteindre une valeur $n$ à l'aide de l'arbre des chaînes d'additions ?

**(6)** On se donne un élément $x$ d'un monoïde $(X, \times)$. Quel est le nombre minimal de multiplications requises pour calculer $x^{15}$ ?

**(7)** Comparez le au nombre de multiplications effectuées par l'algorithme *square & multiply*, le calcul d'un carré étant comptabilisé comme une multiplication.

**Solution.** **(1)** Une séquence d'entiers naturels $(a_k)_{k\in[\![0,\ell]\!]}$ est une chaîne d'addition de longueur $\ell$ si et seulement si elle satisfait la proposition suivante :

$$(a_0 = 1) \wedge \big(\forall k \in [\![1, \ell]\!] \;\; (a_k > a_{k-1}) \wedge (\exists (i, j) \in [\![0, k-1]\!]^2 \; a_k = a_i + a_j)\big).$$

**(2)** L'algorithme Square & Multiply, répond *déjà* à la question en substance puisque `*` désigne la loi de composition du monoïde considéré et `e` son élément neutre. En utilisant la notation additive, il s'agit de calculer $k.x$ au lieu de $x^k$. La réécriture additive est immédiate :

```
ALGORITHME DoubleAdd(x,k):
DONNEES
 · x: valeur
 · k: entier de n bits
VARIABLES
 · R: valeurs
 · i: entier
DEBUT
 · R ← 0
 · i ← 0
 · TQ (i < n) FAIRE
 ·  · R ← R + R
 ·  · SI (k[n - (i + 1)] > 0) ALORS
 ·  ·  · R ← R + x
 ·  · FSI
 ·  · i ← i + 1
 · FTQ
 · RENVOYER R
FIN
```

**(3)** La propriété $x^a\, x^b = x^{a+b}$ montre comment passer d'une écriture multiplicative à une écriture additive du problème. La séquence de produits commençant par la valeur $x^1$ et s'achevant par la valeur $x^n$ se traduit par une séquence de sommes commençant par la valeur 1 et s'achevant par la valeur $n$.

**(4)** L'arbre partiel de profondeur $k = 4$ est donné en figure 2. On a représenté les trois nœuds aux profondeurs 5 et 6 qui sont présents dans deux chaînes se terminant par la valeur 15 pour illustrer les réponses suivantes.

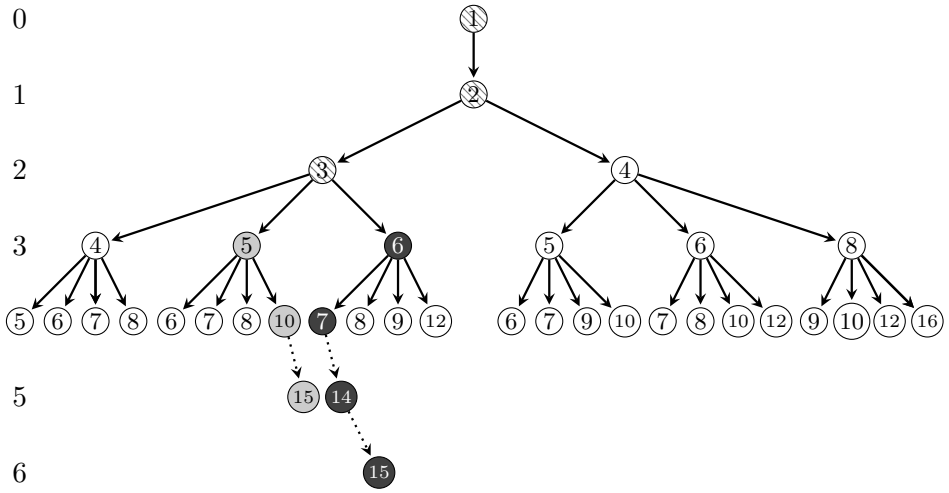


Figure 2. Arbre des chaînes d'additions de longueur $k = 4$.

**(5)** Par construction, les $k+1$ valeurs d'une chaîne d'additions $(a_i)_{i\in[\![0,k]\!]}$ de longueur $k$ sont toutes différentes et strictement positives. Par conséquent, les $k+1$ valeurs $a_k + a_i$ pour $i \in [\![0, k]\!]$ sont toutes distinctes et strictement supérieures à $a_k$, donc le nœud $a_k$ contient au moins ces $k + 1$ fils .

Comme $a_{k+1} := a_i + a_j$ pour deux indices $(i, j) \in [\![0, k]\!]^2$ et l'addition étant commutative, le nombre de couples $(i, j) \in [\![0, k]\!]$ tels que $i \leqslant j$ majore le nombre de sommes distinctes :

$$\sum_{i=0}^{k}\sum_{j=i}^{k} 1 = \sum_{i=0}^{k}(k-i+1) = (k+1)^2 - \sum_{i=0}^{k} 1 = \frac{k(k+1)}{2}.$$

Mais par construction, les valeurs $a_i$ pour $i \in [\![0,k]\!]$ font nécessairement partie de l'ensemble de ces sommes et ne sont pas des fils de $a_k$, on peut donc affiner la majoration :

$$\frac{k(k+1)}{2} - (k+1) = \frac{(k+1)(k-2)}{2}.$$

Il n'est évidemment pas raisonnable de chercher le chemin le plus court pour atteindre une valeur fixée dans cet arbre puisque le nombre minimum de nœuds à la profondeur $k$ est égal à $k!$, la complexité croît donc au mieux factoriellement avec la longueur de la chaîne recherchée.

**(6)** La chaîne $1, 2, 3, 5, 10, 15$ réalise 5 additions (chaîne hachurée puis grise dans l'arbre de la figure 2) :

$$\begin{aligned} 2 &= 1+1 \\ 3 &= 2+1 \\ 5 &= 3+2 \\ 10 &= 5+5 \\ 15 &= 10+5 \end{aligned}$$

La question de la minimalité est loin d'être triviale, c'est même un problème éminemment compliqué puisque l'on ne connaît pas à l'heure actuelle d'algorithme efficace pour trouver une chaîne d'addition de longueur minimale (on conjecture même qu'il n'en existe pas !) Dans le cas particulier qui nous concerne, une étude de cas nous permet de montrer que la chaîne proposée est minimale.

En effet, on vérifie aisément que les valeurs 2, 3, 4 et 5 ne peuvent s'obtenir en moins de 1, 2, 2 et 3 additions respectivement. Comme 15 ne peut s'écrire comme la somme de deux entiers dans l'ensemble $\{1, 2, 3, 4, 5\}$, il faut nécessairement une 4-ème addition pour construire d'autres valeurs, ce qui entraîne que la valeur 15 ne pourra être obtenue, au mieux, qu'avec 5 additions, ce minimum étant atteint avec la chaîne $1, 2, 3, 6, 12, 15$.

**(7)** L'écriture binaire de 15 est 1111 et la chaîne d'additions constituée par les exposants successifs de $x$ durant l'exécution de l'algorithme *square & multiply* (en partant du bit de poids fort) est égale à $1, 2, 3, 6, 7, 14, 15$ (chaîne hachurée puis noire dans l'arbre de la figure 2).

$$\begin{aligned} 2 &= 1+1 \\ 3 &= 2+1 \\ 6 &= 3+3 \\ 7 &= 6+1 \\ 14 &= 7+7 \\ 15 &= 14+1 \end{aligned}$$

Elle nécessite 6 additions, l'algorithme n'est donc pas optimal.

**EXERCICE 3.** Écrivez une version de l'algorithme *square & multiply* dans laquelle les bits de l'exposant sont utilisés dans l'ordre inverse, autrement dit du bits de poids faible vers le bit de poids fort.

**Solution.** Il faut simplement se donner un registre (`P` dans l'algorithme) pour mémoriser les carrés successifs.

```
ALGORITHME SquareMultiply(x,k):
DONNEES
 · x: valeur
 · k: entier
VARIABLES
 · R, P: valeurs
 · i: entier
DEBUT
 · R ← e
 · P ← x
 · i ← 0
 · TQ (i < |k|) FAIRE
 ·  · SI (k[i] > 0) ALORS
 ·  ·  · R ← R * P
 ·  · FSI
 ·  · P ← P * P
 ·  · i ← i + 1
 · FTQ
 · RENVOYER R
FIN
```

**EXERCICE 4.** La suite de Fibonacci est une suite récurrente d'entiers naturels de premiers termes $F_0 := 0$ et $F_1 := 1$ définie par la relation

$$\forall n \in \mathbb{N} \quad F_{n+2} := F_{n+1} + F_n. \tag{1}$$

**(1)** Écrivez un algorithme sur la machine RAM qui pour l'entrée $n$, calcule le terme $F_n$ de la suite de Fibonacci. On suppose que les registres peuvent coder des entiers de taille arbitrairement grande. On supposera que $n \geqslant 2$. Calculez le nombre $C(n)$ d'instructions décodées par la machine RAM en fonction de $n$.

On définit la matrice $\Phi := \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$.

**(2)** Démontrez que

$$\forall n \in \mathbb{N} \quad \Phi^{n+1} = \begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix}. \tag{2}$$

**(3)** Écrivez un algorithme `ProdMat(A,B)` qui renvoie le produit de deux matrices carrés avec $A[i][j]$ désignant le terme à la ligne $i$ et la colonne $j$.

**(4)** Écrivez un algorithme avec une complexité $\Theta(\log(n))$ pour calculer $F_n$.

**Solution.** **(1)** Les registres $R_1$ et $R_2$ contiennent respectivement $F_n$ et $F_{n+1}$, le registre $R_3$ la valeur de $n$ (décrémentée à chaque étape cf. #15), et le registre $R_4$ sert de variable auxilaire pour la mise à jour de $F_n$ et $F_{n+1}$ :

```
 00 |   READ     ; LECTURE n
 01 |   DEC 0    ; n ← n - 1
 02 |   STORE 3  ; INITIALISATION R[3], R[1] et R[2]
 03 |   LOAD #0  ;   ↓
 04 |   STORE 1  ;   ↓
 05 |   LOAD #1  ;   ↓
 06 |   STORE 2  ; FIN INITIALISATION
>07 |   LOAD 3   ; CHARGER n (n = R[3])
 08 |   JUMZ 17  ; SI n = 0 ALORS FIN (SAUTER A #17)
 09 |   LOAD 2   ; SINON ACC ← R[2]
 10 |   STORE 4  ; R[4] ← R[2] (SAUVEGARDER R[2])
 11 |   ADD 1    ; ACC ← R[2] + R[1]
 12 |   STORE 2  ; R[2] ← ACC (MISE A JOUR R[2])
 13 |   LOAD 4   ; ACC ← R[4] (CHARGER ANCIEN R[2])
 14 |   STORE 1  ; R[1] ← ACC (MISE A JOUR R[1])
 15 |   DEC 3    ; n ← n - 1
 16 |   JUMP 7   ; SAUT A #07 (ON CONTINUE)
>17 |   LOAD 2   ; ACC ← R[2]
 18 |   WRITE    ; AFFICHAGE RESULTAT
 19 |   STOP     ; ARRET
```

Les instructions d'initialisation #00 à #06 ainsi que l'affichage du résultat #17 à #19 ne sont exécutées qu'une seule fois. Les instructions #07 et #08 de test d'entrée dans la boucle sont réalisées $n$ fois (on rappelle que l'on a décrémenté $n$ à l'initialisation) et les instructions intra boucle #09 à #16 sont réalisées $n-1$ fois :

$$C(n) = 10 + 2n + 8(n-1) = 10n + 2.$$

**(2)** On raisonne par récurrence avec le prédicat $P(n)$ suivant :

$$\Phi^{n+1} = \begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix}.$$

Comme $F_2 = F_1 + F_0 = 1 + 0 = 1$, la propriété $P(0)$ est vérifiée (initialisation). Montrons à présent que $\forall n \in \mathbb{N}\ P(n) \Rightarrow P(n+1)$ (hérédité). On a

$$\Phi^{n+2} = \Phi^{n+1}\,\Phi.$$

D'après l'hypothèse de récurrence on en déduit que

$$\begin{aligned} \Phi^{n+2} &= \begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} F_{n+2} + F_{n+1} & F_{n+2} \\ F_{n+1} + F_n & F_{n+1} \end{pmatrix} \\ &= \begin{pmatrix} F_{n+3} & F_{n+2} \\ F_{n+2} & F_{n+1} \end{pmatrix} \quad \text{d'après (1)} \end{aligned}$$

La propriété $P(n+1)$ est donc vérifiée (Hérédité).

**(3)** L'algorithme est le suivant :

```
ALGORITHME ProdMat(A,B):matrice
DONNEES
 · A,B: matrices carrées 2x2
VARIABLES
 · C: matrice carrée 2x2
```

```
 · i: entier
DEBUT
 · C[1][1] ← A[1][1]*B[1][1]+A[1][2]*B[2][1]
 · C[1][2] ← A[1][1]*B[1][2]+A[1][2]*B[2][2]
 · C[2][1] ← A[2][1]*B[1][1]+A[2][2]*B[2][1]
 · C[2][2] ← A[2][1]*B[1][2]+A[2][2]*B[2][2]
 · RENVOYER C
FIN
```

**(4)** Il suffit de s'appuyer sur l'algorithme *square & multiply* pour calculer $\Phi^{n-2}$ et obtenir le résultat souhaité :

```
ALGORITHME Fibonacci(n):entier
DONNEES
 · n: entier
VARIABLES
 · P: matrice carré 2x2
DEBUT
 · P[1][1] ← 1
 · P[1][2] ← 1
 · P[2][1] ← 1
 · P[1][1] ← 0
 · P ← SM(P,n-2)
 · RENVOYER P[1][1]
FIN
```

Notons que l'on peut améliorer la complexité de l'algorithme en remarquant que $\Phi[2,1] = \Phi[1,2]$ et économiser ainsi un produit à chaque itération.

**EXERCICE 5.** Soit $N \geqslant 1$ et $b \geqslant 2$ deux entiers naturels. Démontrez que le nombre de chiffres de l'écriture de $N$ en base $b$ est égal à $\lfloor \log_b N \rfloor + 1$. Indication : on admet que pour tout réel $x \geqslant 0$, il existe un unique entier $N$ tel que $N \leqslant x < N+1$, appelé *partie entière* de $x$ que l'on note $\lfloor x \rfloor$.

**Solution.** Notons $k$ le nombre de chiffres de $N$ en base $b$, autrement dit, en notant $a_0, a_1, \dots a_{k-1}$ ces chiffres, du moins significatif au plus significatif, on a

$$N = \sum_{i=0}^{k-1} a_i b^i.$$

La plus petite valeur d'un nombre à $k$ chiffres en base $b$ est $b^{k-1}$ (tous les $a_i$ sont nuls sauf $a_{k-1} = 1$) et la plus grande $b^k - 1$ (tous les $a_i$ sont égaux à $b-1$). On a donc

$$b^{k-1} \leqslant N < b^k.$$

La fonction logarithme en base $b$ étant croissante, on en déduit

$$k - 1 \leqslant \log_b(N) < k.$$

D'après la définition de la partie entière, on a $\lfloor \log_b N \rfloor = k - 1$ d'où l'on tire le résultat.

**EXERCICE 6.** On rappelle l'algorithme de Hörner :

```
ALGORITHME Horner(P,x):réel
DONNEES
 · P[0,n]: liste de n+1 réels
 · x: réel
VARIABLES
 · R: réel
 · i: entier
DEBUT
 · R ← P[n]
 · i ← 0
 · TQ (i < n) FAIRE
 ·  · i ← i + 1
 ·  · R ← R * x + P[n - i]
 · FTQ
 · RENVOYER R
FIN
```

Démontrez que l'algorithme s'arrête et démontrez sa justesse. Indication : considérez le prédicat $P(i)$ suivant :

$$R = \sum_{k=0}^{i} a_{n-k} x^{i-k}. \tag{3}$$

**Solution.** Arrêt : la variable $i$ est initialisée à 0 avant la boucle et n'est modifiée qu'à la ligne #12 où elle est incrémentée, et atteindra la valeur $n$ faisant échouer l'entrée dans la boucle. Justesse : on note $S_i$ la somme dans (3). Le prédicat $P(i)$ est vrai avant la boucle quand $i = 0$, en effet $S_0 = a_n$. Supposons que la proposition $P(i)$ soit vraie en entrant dans la boucle, i.e. $R = S_i$, alors après l'incrémentation en ligne #12, c'est la

proposition $R = S_{i-1}$ qui est vraie soit

$$R = \sum_{k=0}^{i-1} a_{n-k} x^{i-1-k}.$$

$$\text{donc} \quad R\,x + a_{n-i} = (\sum_{k=0}^{i-1} a_{n-k} x^{i-k})x + a_{n-i}$$

$$= \sum_{k=0}^{i} a_{n-k} x^{i-k}.$$

et on a à nouveau (3). Quand on sort de la boucle, on a $i = n$ et en remplaçant dans (3), on obtient le résultat désiré.