III. Arithmétique

Commeçons par un rappler concernant la division euclidienne.

Proposition 1. Pour tout entiers a et b de $\mathbb{Z}$ avec $b \neq 0$, il existe des uniques entiers q et r vérifiant $a = bq + r$ avec $0 \leq r \leq |b|$.

On notea $\text{quo}(a,b) = \lfloor a/b \rfloor$ et $\text{res}(a,b) = a - \text{quo}(a,b)\, b$ le quotient et le reste de la division de a par b

1. Opération élémentaires sur les entiers.

Théorème 2. Soit β un entier ≥ 2. Pour tout $a \in \mathbb{N}$ il existe une unique suite d'entiers $(c_k)_{k\in\mathbb{N}}$ de $\{0, \ldots, \beta - 1\}$ telle qu'on ait

$$a = \sum_{i=0}^{+\infty} c_i \beta^i. \tag{3.1}$$

Démonstration. Fixons une base β. Montrons par récurrence sur $n \in \mathbb{N}$

H_n : Pour tout $a \in \{0, \ldots, n\}$ il existe un unique suite $(c_k)_{k\in\mathbb{N}}$ d'éléments de $\{0, \ldots, \beta\}$ telle qu'on ait (3.1).

Traitons le cas $n = 0$. Soit $(c_k)_{k\in\mathbb{N}}$ une suite d'éléments de $\{0, \ldots, \beta\}$. Si l'un des c_k est non nul alors la somme (3.1) l'est aussi. Ainsi la seule suite possible pour $a = 0$ est la suite nulle, ce qui établit (H_0). Supposons (H_n) pour $n \geq 0$ et montrons (H_{n+1}). Soit a un entier de $\{0, \ldots, n+1\}$. Pour $a \leq n$, on conclut avec (H_n). Supposons donc $a = n + 1$. Par division euclidienne, il existe un unique couple d'entiers tel qu'on ait $a = q\,\beta + r$ avec $r \in \{0, \ldots, \beta - 1\}$. De $q = (a - r)/\beta$ et $\beta \geq 2$, nous obtenons $q < a$. Par H_n il existe une suite (c'_k) d'éléments de $\{0, \ldots, \beta\}$ vérifiant

$$q = \sum_{i=0}^{+\infty} c'_i \beta^i.$$

Nous avons alors

$$a = q\,\beta + r = \left(\sum_{i=0}^{+\infty} c'_i \beta^{i+1} \right) + r = \left(\sum_{i=1}^{+\infty} c'_{i-1} \beta^i \right) + r = \sum_{i=0}^{+\infty} c_i \beta^i,$$

en posant $c_0 = r$ et $c_i = c'_{i-1}$ pour $i \geq 1$. Montrons l'unicité. Soient $(c_k)_{k\in\mathbb{N}}$ et $(d_k)_{k\in\mathbb{N}}$ deux suites de $\{0, \ldots, \beta - 1\}$ vérifiant

$$a = \sum_{k=0}^{+\infty} c_k \beta^k = \sum_{k=0}^{+\infty} d_k \beta^k.$$

La division euclidienne de a par β donne $r = c_0 = d_0$ ainsi que

$$q = \sum_{k=1}^{+\infty} c_k \beta^k = \sum_{k=1}^{+\infty} d_k \beta^k,$$

avec $q < a$. L'hypothèse (H_n) implique alors $c_k = d_k$ pour $k \geq 1$. D'où l'unicité puis (H_{n+1}). □

Comme la suite β^k tend vers $+\infty$ avec k, pour tout entier a il existe un plus grand $\ell \in \mathbb{N}$ tel que c_ℓ soit non nul. Nous avons alors

$$a = \sum_{i=0}^{\ell} c_i \beta^i. \tag{3.2}$$

La suite $(c_0, \dots, c_\ell)$ est l'*écriture* de a en base β, on note alors $a = \overline{c_\ell \cdots c_0}^\beta$. L'entier $\ell + 1$ est la *taille* de a en base β.

En machine un entier naturel pourra être donc être représenté, une fois une base β fixée, par son écriture en base β, qu'on codera à l'aide d'un tableau. En ajoutant un signe, nous défnissons de même une représentation machine en base β des entiers de $\mathbb{Z}$.

Exercice. Quelles bases de numérations utilisez-vous quotidiennement (ou presque) ?

Correction. Base 10 (numération décimale). Base 60 pour secondes, minutes pour la mesure du temps et les angles en degrés. Les processeurs de vos ordinateurs travaillent en base 2, 2^8, 2^{16}, 2^{32} et 2^{64} pour 1, 8, 16, 32 et 64 bits (voir évolution des consoles).

Proposition 3. Pour toute base β, la taille d'un entier a en base β est $\lfloor \log_\beta(a+1) \rfloor$ qui est dans $O(\log a)$ (le log étant en base quelconque).

Démonstration. Pour tout $k \in \mathbb{N}$, β^{k-1} est le plus petit nombre de taille k en base β et $\beta^k - 1$ est le plus grand. Pour $a \in \{\beta^{k-1}, \dots, \beta^k - 1\}$ nous avons $\log_\beta(a+1) \in [k-1, k[$ et donc $\lfloor \log_\beta(a+1) \rfloor = k$. Ainsi la taille de l'entier a en base β est $\lfloor \log_\beta(a+1) \rfloor$.

Regardons ce qu'il se passe si on change de base. Soit x un réel. On a $x = \beta^{\log_\beta(x)}$ ainsi que $x = \gamma^{\log_\gamma(x)}$. On obtient

$$x = \gamma^{\log_\gamma(x)} = \left(\beta^{\log_\beta(\gamma)}\right)^{\log_\gamma(x)} = \beta^{\log_\beta(\gamma)\ \log_\gamma(x)},$$

et donc $\log_\beta(x) = \log_\beta(\gamma)\ \log_\gamma(x)$. Ainsi passer d'une base β à une base γ change le nonbre de chiffres d'un entier a par une constante multiplicative ne dépend que des bases et pas de a. □

Exercice.

1. Écrire un alogirhme DECOMPOSE permettant d'obtenir l'écriture (sous forme de tableau) d'un entier positif en base β qu'on passera en paramêtre.
2. Écrire un algorithme faisant le contraire du précédant.

Correction.

1.

```
fonction DECOMPOSE(a, β)
    ℓ ← ⌈log_β(a + 1)⌉
    créer un tableau T de taille ℓ
    t ← a
    pour i de 0 à ℓ − 1 faire
        T[i] ← res(t, β)
        t ← quo(t, β)
    fin pour
    retourne T
fin fonction
```

2.

```
fonction COMPOSE(T, β)
    a ← 0
    p ← 1
    pour i de 0 à longueur de T faire
        a ← a + T[i] × p
        p ← p × β
    fin pour
    retourne a
fin fonction
```

Une fois une base β fixée, les opérations élémentaires $(<, >, =, +, -, \times)$ sur les chiffres se font en temps constant, c'est-à-dire, en $O(1)$.

Proposition 4. Soit a et b deux nombres entiers de taille au plus n pour une base β fixée. Nous avons les complexités suivantes pour les opérations élémentaires sur $\mathbb{Z}$:

comparaison	$a = b$, $a < b$, $a > b$	$O(n)$
addition	$a + b$	$O(n)$
soustraction	$a - b$	$O(n)$
multiplication par un chiffre	$a \times$ chiffre	$O(n)$
multplication	$a \times b$	$O(n^2)$
division avec reste	$\text{quo}(a, b)$, $\text{res}(a, b)$	$O(n^2)$

2. Algorithme de multiplication rapide

Pour un entier k donné, nous avons :

$$\left(a_1 + a_2 \times \beta^k\right) \times \left(b_1 + b_2 \times \beta^k\right) = a_1\, b_1 + (a_1\, b_2 + a_2\, b_1) \times \beta^k + a_2\, b_2 \times \beta^{2k},$$

ainsi avec l'algotihme de multiplication élémentaire la multiplication de gauche est obtenu à partir de 4 multiplication de taille plus petite.

En 1962, le mathématicien russe Anatolii Alexevich Karatsuba constate que la même multiplication peut être obetnu en utilisant seulement 3 multiplications de taille plus petite. En effet nous avons

$$(a_1 + a_2) \times (b_1 + b_2) = a_1 b_1 + a_1 b_2 + a_2 b_1 + a_2 b_2$$

et donc

$$a_1\, b_2 + a_2\, b_1 = (a_1 + a_2) \times (b_1 + b_2) - a_1 b_1 - a_2 b_2.$$

Il est donc suffisant de calculer les trois plus petits produits $a_1 b_1$, $a_2 b_2$ et $(a_1 + a_2) \times (b_1 + b_2)$.

L'algorithme de Karatsuba utilise ce fait pour exploiter la technique *diviser pour régner* et proposer une version rapide de la multiplication élémentaire. Cet algortihme est naturellement récursif, de base les nombres à au plus un chiffre.

Exercice.

1. Appliquer l'algorithme de Karatsuba pour la multiplication 2345×3654 en base 10

2. Écrire une version récursive de l'algorithme de Karatsuba.

3. Utiliser le master théorème pour cacluler la compléxité de l'algorithme de Karatsuba.

Correction.

1. On coupe 2345 et 3654 en $23\times100+45$ et $36\times100+54$. On pose $a = 45\times54$, $b = 23\times36$ et $c = (45+23)\times(54+36) = 68\times90$. Nous avons alors

$$2345\times3654 = a + (c-a-b)\times100 + b\times10000.$$

Pour calculer a, b et c on utilise à nouveau l'algorithme de Karatsuba. Pour a, on décomose 45×54 en $5+4\times10$ et $4+5\times10$. On pose $d = 5\times4$, $e = 4\times5$ et $f = (4+5)\times(4+5) = 9\times9$. Nous avons alors

$$a = 45\times54 = d + (f-d-e)\times10 + e\times100.$$

Comme les nombres intervenants dans les produits d, e et f n'ont qu'un seul chiffre, nous faisons les calculons avec une multiplication élémetaire par un chiffre. D'où $d = 20$, $e = 20$ et $f = 81$. Nous obtenons ainsi

$$\begin{aligned} a &= 20 + (81-20-20)\times10 + 20\times100 \\ &= 20 + 41\times10 + 20\times100 \\ &= 20 + 410 + 2000 = 2430. \end{aligned}$$

De même nous obtenons $b = 828$ et $c = 6120$

$$\begin{aligned} 2345\times3654 &= 2430 + (6120-2430-828)\times100 + 828\times10000 \\ &= 2430 + 2862\times100 + 828\times10000 \\ &= 2430 + 286200 + 8280000 = 8568630. \end{aligned}$$

2.

```
fonction KARATSUBA(T_a, T_b, β)
    ℓ ← max(longueur(T_a), longueur(T_b))
    si ℓ ≤ 1 alors
        retourne MULTIPLICATION_CHIFFRE(T_a, T_b[0], β)
    fin si
    ajoute (ℓ − longueur(T_a)) zéro à la fin T_a
    ajoute (ℓ − longueur(T_b)) zéro à la fin T_b
    k ← quo(ℓ, 2)
    T_{a1} ← T_a[0, ..., k − 1]
    T_{a2} ← T_a[k, ..., ℓ − 1]
    T_{b1} ← T_b[0, ..., k − 1]
    T_{b2} ← T_b[k, ..., ℓ − 1]
    T_1 ← KARATSUBA(T_{a1}, T_{b2}, β)
    T_2 ← KARATSUBA(T_{b1}, T_{a2}, β)
    A ← ADDITION(T_{a1}, T_{a2}, β)
    B ← ADDITION(T_{b1}, T_{b2}, β)
    T_3 ← KARATSUBA(A, B, β)
    T_3 ← SOUSTRACTION(T_3, T_1, β)
    T_3 ← SOUSTRACTION(T_3, T_2, β)
    ajoute k zéros au début de T_3
    ajoute 2k zéros au début de T_2
    T ← ADDITION(T_1, T_2, β)
    T ← ADDITION(T, T_3, β)
    retourne T
fin fonction
```

3. On reprend les notations du master théorème. Pour faire une multiplcation entre nombres de taille n, l'algorithme de Karatsuba fait trois appels récursifs pour des multiplications entre nombres de tailles $n/2$. Nous avons donc $a = 3$ et $b = 2$. Pour le coût hors appels récursifs nous avons des addition et des soustractions entre nonbre de taille n ainsi que l'ajout de 0. Comme chacune de ces instructions est en $O(n)$, nous avons $f(n) \in O(n)$ et donc d vaut 1. De $\log_2(3) > 1$ nous obtenons que l'algorithme de Karatsuba effecture le produit de deux nomnres de taille n en $O(n^{\log_2(3)}) \subseteq O(n^{1.585})$.

3. Algortihme d'Euclide

Nous commençons d'abord par un résultat classique sur le pgcd.

Proposition 5. Pour tout entiers a et b avec $b \neq 0$ nous avons $\text{pgcd}(a, b) = \text{pgcd}(b, \text{res}(a, b))$.

Démonstration. Posons $r = \text{res}(a, b)$ et $q = \text{quo}(a, b)$. Nous avons donc $a = bq + r$. De plus nous posons $d = \text{pgcd}(a, b)$ et $d' = \text{pgcd}(b, \text{res}(a, b)) = \text{pgcd}(b, r)$. Par définition d' est un diviseu de b er r puis de a car nous avons $a = bq + r$. Ainsi d' est aussi un diviseur commun de a et b et donc d' divise d. Montrons d divise d'. Toujours par définition, d est un diviseur de a et b et donc aussi de $r = a - bq$. L'entier d étant un diviseur commun de a et r, il divise d'. Finalement, nous avons obtenu $d = d'$. □

Exercice. Exploiter la proposition précédente pour proposer un algorithme récursif `pgcd_rec` retournant le pgcd de deux entiers positifs a et b non tous deux nuls.

Correction.

```
fonction PGCD_REC(a, b)
    si a = 0 alors
        retourne b
    sinon
        retourne pgcd_rec(b, res(a, b))
    fin si
fin fonction
```

Nous allons maitenant construire une version non récursive de l'algorithme précédent, appelé algorithme d'Euclide. Pour cela nous construisons une suite $(r_n)_{n\geq 0}$ définie par $r_0 = a$, $r_1 = b$ et pour $n \geq 0$ par

$$r_{n+2} = \text{res}(r_n, r_{n+1}) = r_n - q_{n+2}r_{n+1}, \quad \text{avec } q_{n+2} = \text{quo}(r_n, r_{n+1}).$$

Comme la suite (r_n) est strictement décroissante pour $n \geq 1$, et minorée par 0, il existe un entier $N \in \mathbb{N}$ tel que nous ayons $r_N = 0$. D'après la proposition nous avons $\text{pgcd}(a, b) = \text{pgcd}(r_n, r_{n+1})$ pour tout $n \in \{0, \ldots, N-1\}$. De $\text{pgcd}(r_{N-1}, r_N) = \text{pgcd}(r_{N-1}, 0) = r_{N-1}$, nous obtenons $r_{N-1} = \text{pgcd}(a, b)$.

Exercice.

1. Utiliser la suite $(r_n)_n$ pour le calcul du pgcd de 216 et 126.

2. En déduire un algorithme non récursif PGCD_EUCLIDE pour le calcul du pgcd de deux nombres a et b non tous deux nuls.

Correction.

1. Nous avons

n	0	1	2	3	4	5
r_n	216	126	90	36	18	0
q_n			1	1	2	2

2.

```
fonction PGCD_EUCLIDE(a, b)
    r_p ← a
    r ← b
    tant que r ≠ 0 faire
        q ← quo(r_p, r)
        t ← r
        r ← r_p − q × r
        r_p ← r
    fin tant que
    retourne r_p
fin fonction
```

Théorème 6 (de Bezout)**.** Pour tous entiers a et b non tous deux nuls, il existe des entiers u et v tels qu'on ait $au + bv = \text{pgcd}(a, b)$. Réciproquement, pour tous entiers u et v nous avons que $\text{pgcd}(a, b)$ est un diviseur de $au + bv$.

Exercice. Déterminer u et v tels que nous ayons $216u + 126v = \text{pgcd}(216, 126)$.

Correction. Pour le calcul de $\text{pgcd}(216, 126)$ nous avons

$$216 = 126 + 90, \quad 126 = 90 + 36, \quad 90 = 2 \times 36 + 18, \quad 36 = 2 \times 18,$$

ce qui en remontant à partir de 18 donne

$$\begin{aligned} 18 &= 90 - 2 \times 36 \\ 18 &= 90 - 2 \times (126 - 90) = -2 \times 126 + 3 \times 90 \\ 18 &= -2 \times 126 + 3 \times (216 - 126) = 3 \times 216 - 5 \times 126. \end{aligned}$$

Les valeurs $u = 3$ et $v = -5$ sont donc solutions de $216u + 126v = \text{pgcd}(216, 126) = 18$.

Reprenons la définition de la suite $(r_n)_{n\geq 0}$ pour constuire deux suites $(u_n)_{n\geq 0}$ et $(v_n)_{n\geq 0}$ vérifiant

$$u_n\, a + v_n\, b = r_n \quad \text{pour } 0 \leq n \leq N.$$

Nous posons $u_0 = 1$ et $v_0 = 0$ ainsi que $u_1 = 0$ et $v_1 = 1$. Pour $n \geq 0$ nous avons

$$\begin{aligned} u_{n+2}\, a + v_{n+2}\, b &= r_{n+2} \\ &= r_n - q_{n+2}\, r_{n+1} \\ &= u_n\, a + v_n\, b - q_{n+2}\, (u_{n+1}\, a + v_{n+1} b) \\ &= (u_n - q_{n+2} u_{n+1})\; a + (v_n - q_{n+2} v_{n+1})\; b. \end{aligned}$$

Ainsi, pour $n \geq 0$, nous posons

$$u_{n+2} = u_n - q_{n+2} u_{n+1} \quad \text{et} \quad v_{n+2} = v_n - q_{n+2} v_{n+1}.$$

Comme le pgcd de a est b est donné par r_{N-1}, nous obtenons

$$u_{N-1}\, a + v_{N-1}\, b = r_{N-1} = \text{pgcd}(a, b)$$

et nous trouvons que $u = u_{N-1}$, $v = v_{N-1}$ est solution de $u\, a + v\, b = \text{pgcd}(a, b)$.

Exercice.

1. Déterminer les suites $(u_n)_{n\geq 0}$ et $(v_n)_{n\geq 0}$ obtenues avec $a = 216$ et $b = 126$.
2. Déterminer un aglorithme PGCD_EUCLIDE_ETENDU retournant un triplet (d, u, v) tel que nous ayons $au + bv = d = \text{pgcd}(a, b)$.

Correction.

1. Nous avons

n	0	1	2	3	4	5
r_n	216	126	90	36	18	0
u_n	1	0	1	-1	3	-7
v_n	0	1	-1	2	-5	12
q_n			1	1	2	2

2.

```
fonction PGCD_EUCLIDE_ETENDU(a,b)
    r_p ← a
    r ← b
    u_p ← 1
    u ← 0
    v_p ← 0
    v ← 1
    tant que r ≠ 0 faire
        q ← quo(r_p, r)
        t ← r
        r ← r_p − q × r
        r_p ← t
        t ← u
        u ← u_p − q × u
        u_p ← t
        t ← v
        v ← v_p − q × v
        v_p ← t
    fin tant que
    retourne (r_p, u_p, v_p)
fin fonction
```

Théorème 7 (de Lamé)**.** Pour des entiers a et b de taille au plus n, les algorithmes PGCD_EUCLIDE et PGCD_EUCLIDE_ETENDU appliqué à a et b ont une compléxité dans le pire des cas en $O(n^3)$.

Démonstration. Le point clé est d'établir que l'entier minimal N tel qu'on ait $r_N = 0$ vérifie la relation $N \in O(n)$. Prenons le problème à l'envers. Fixons un entier N et déterminons les plus petits entiers a_N et b_N tel qu'on ait $r_N = 0$ et $r_{N-1} \neq 0$, c'est-à-dire, tels que l'algorithme d'Euclide nécessite N étapes.

Fait : $\text{pgcd}(a_N, b_N) = 1$**.**

Supposons que ce n'est pas le cas, c.-à-d., $\text{pgcd}(a_N, b_N) = d \neq 1$. Il existe alors des entiers a et b tels qu'on ait $a_N = \lambda\, a$ et $b_N = \lambda\, b$. Notons (r_k) et (r'_k) les suites associées au calcul des $\text{pgcd}(a, b)$ et $\text{pgcd}(\lambda a, \lambda b)$ respectivement. Notons $N(a, b)$ le plus petit entier vérifiant $r_{N(a,b)} = 0$. Montrons par récurrence sur $k \in \{0, \ldots, N(a, b)\}$ la relation

$$H_k \ : \ r'_k = \lambda\, r_k.$$

De $r_0 = a$, $r'_0 = \lambda\, a$, $r_1 = b$ et $r'_1 = \lambda\, b$ nous obtenons H_0 et H_1. Supposons H_k et H_{k+1} pour $k \in \{0, \ldots, N(a, b) - 2\}$ et montrons H_{k+2}. Posons $q_{k+2} = \text{quo}(r_k, r_{k+1})$. Nous avons alors $r_k = q_{k+2} r_{k+1} + r_{k+2}$ avec $0 \leq r_{k+2} < r_{k+1}$. Ainsi, par hyptohèses de récurrences

$$r'_k = \lambda r_k = q_{k+2}(\lambda r_{k+1}) + \lambda r_{k+2} = q_{k+2} r'_{k+1} + \lambda r_{k+2}$$

avec $0 \leq \lambda r_{k+2} < \lambda r_{k+1} = r'_{k+1}$, ce qui, par unicité de la division euclidienne, implique que λr_{k+2} est le reste de la division euclidienne de r'_k par r'_{k+1} et donc $r'_{k+2} = \lambda\, r_{k+2}$. Nous avons

donc établi $r'_k = \lambda\, r_k$ pour tout $k \in \{0, \dots, N(a,b)\}$ et en particulier

$$\operatorname{pgcd}(a_N, b_N) = \operatorname{pgcd}(\lambda a, \lambda b) = r'_{N(a,b)-1} = \lambda r_{N(a,b)-1} = \lambda \operatorname{pgcd}(a,b).$$

Pour les deux couples (a_N, b_N) et (a,b) le nombre d'étapes dans l'algorithme d'Euclide est le même, à savoir N. Mais nous avons $a < a_N$ et $b < b_N$, ce qui est en contradiction avec la minimalité de a_N et b_N. La condition $\operatorname{pgcd}(a_N, b_N) = 1$ est donc nécessaire.

Fait : $a_N = F_N$ et $b_N = F_{N-1}$, deux termes consécutifs de la suite de Fibonacci.

En effet afin que les entiers a_N et b_N soient minimals nous devons nécessairement avoir $q_k = 1$ pour tout entier $k \in \{2, \dots, N\}$. Notons $r_0, \dots, r_N$ la suite des restes associée à (a_N, b_N). Posons $s_k = r_{N-k}$. Nous avons $s_0 = r_N = 0$ et $s_1 = r_{N-1} = \operatorname{pgcd}(a_N, b_N) = 1$. De

$$r_{k+2} = r_k - q_{k+2} r_{k+1} = r_k - r_{k+1},$$

nous obtenons $r_k = r_{k+2} + r_{k+1}$ et donc $s_{N-k} = s_{N-k-2} + s_{N-k-1}$ pour $k \in \{2, \dots, N\}$. La suite $(s_k)_k$ est donc définie pour $k \in \{0, \dots, N\}$ par $s_0 = 0$, $s_1 = 1$ et

$$s_n = s_{n-1} + s_{n-2} \qquad \text{pour } n \geq 0.$$

La suite (s_k) correspond donc à la suite de Fibonacci et nous avons donc $a_N = r_0 = S_N = F_N$ et $b_N = r_1 = S_{N-1} = F_{N-1}$. Le pire des cas dans l'algorithme d'Euclide est donc atteint pour deux nomnres de Fibonacci consécutifs.

Fait : pour tout $n \in \mathbb{N}$, nous avons la formule de Binet

$$F_n = \frac{1}{\sqrt{5}} \left(\varphi^n + (1-\varphi)^n\right) \sim \frac{\varphi^n}{\sqrt{5}},$$

où $\varphi = \frac{1+\sqrt{5}}{2}$ est le nombre d'or. La démostration de ce fait peut se faire à l'aide de l'algèbre linéaire et est repporté sous forme d'exercice à la fin de ce chapitre.

Bilan. Nous en concluons que le nombre d'étapes de l'algorithme d'Euclide pour des entiers de taille au plus n est en $O(n)$. Chaque étape est une division euclidienne entre deux entiers de taille au plus n et à donc une complexité en $O(n^2)$. La complexité totale, dans le pire des cas, est donc en $O(n^3)$. □

Exercice (Bonus)**.** Posons $U_n = \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix}$, où $(F_n)_n$ est la suite de Fibonacci.

1. Déterminer U_0.

2. Déterminer une matrice A telle que nous ayons $U_n = A \times U_{n-1}$ pour $n \geq 1$.

3. Pour $n \in \mathbb{N}$, déterminer U_n en fonction de A et U_0.

4. Déterminer les valeurs propres de A. En déduire que la matrice A est diagonalisable.

5. Trouver une matrice P inversible telle que $P^{-1}AP$ soit une matrice diagonale D.

6. Déterminer une formule explicite de A^n pour tout $n \in \mathbb{N}$.

7. Etablir la formule de Binet.