Algorithmique

Alexandre Duret-Lutz
adl@lrde.epita.fr

27 février 2012

Cinquième partie V

Trois problèmes

Exponentiation rapide

Exponentiation classique

$$a^b = \underbrace{a \times a \times \cdots \times a}_{b-1 \text{ multiplications}}$$

L'algorithme de calcul de a^b naïf demande une boucle de $\Theta(b)$ multiplications.

Exponentiation rapide (Intro)

Notations

On note $\overline{d_{n-1} \dots d_1 d_0}$ la représentation d'un nombre de n bits en base 2.

Par exemple $19 = \overline{10011}$.

Exponentiation

On veut calculer a^b sachant que $b = \overline{d_{n-1} \dots d_0}$.

$$a^b = a^{\overline{d_{n-1} \dots d_0}} = a^{2 \times \overline{d_{n-1} \dots d_2 d_1}} \times a^{d_0} = (a \times a)^{\overline{d_{n-1} \dots d_2 d_1}} \times a^{d_0}$$

Par exemple $5^{19} = 5^{\overline{10011}} = 25^{\overline{1001}} \times 5^1$.

On calcule récursivement : $25^{\overline{1001}} = 625^{\overline{100}} \times 25^1$.

$625^{\overline{100}} = 390625^{\overline{10}} \times 625^0$.

$390625^{\overline{10}} = 152587890625^{\overline{1}} \times 390625^0$.

Finalement $5^{19} = 152587890625 \times 25 \times 5$

Combien ce calcul demande-t-il de multiplications ?

Exponentiation rapide (Algo)

$$power(a,b) = \begin{cases} 1 & \text{si } b = 0 \\ power(a \times a, \lfloor b/2 \rfloor) & \text{si } b \text{ est pair} \\ power(a \times a, \lfloor b/2 \rfloor) \times a & \text{si } b \text{ est impair} \end{cases}$$

```
FastPower(a, b)
1   if b = 0
2         return 1
3   if odd(b) then
4         return a×FastPower(a × a,⌊b/2⌋)
5   else
6         return FastPower(a × a,⌊b/2⌋)
```

On fait deux multiplications par bit à 1 et une multiplication par bit à 0 dans la représentation binaire de b. Soit $\Theta(\log b)$ multiplications.

Généralisation (1/2)

Monoïde

Un monoïde $(E, \star, e)$ est un ensemble E munie d'une loi interne associative $\star$ et d'un élément neutre e. On a

- associativité : $\forall x, \forall y, \forall z,\ x \star (y \star z) = (x \star y) \star z$
- élément neutre : $\forall x,\ x \star e = e \star x = x$

Puissance

Pour $n \in \mathbb{N}$ et $x \in E$ on définit la loi externe

$$x^n = \begin{cases} e & \text{si } n = 0 \\ x^{n-1} \star x & \text{sinon} \end{cases}$$

L'algorithme d'exponentiation rapide peut alors être utilisé pour calculer x^n en $\Theta(\log n)$ opérations. Les propriétés du monoïde sont importantes pour l'algorithme, car par exemple x^4 s'y calcule comme $((x \star e) \star (x \star e)) \star ((x \star e) \star (x \star e))$ au lieu de $(((e \star x) \star x) \star x) \star x$ comme dans la définition.

Généralisation (2/2)

On peut aussi définir les puissances négatives si l'on travaille sur un groupe.

Groupe

Un monoïde $(E, \star, e)$ est un groupe si tout élément $x \in E$ possède un inverse pour $\star$, noté $\overline{x}$: on a $x \star \overline{x} = \overline{x} \star x = e$.

Puissance

Pour $n \in \mathbb{N}$ et $x \in E$ on définit alors

$$x^n = \begin{cases} e & \text{si } n = 0 \\ x^{n-1} \star x & \text{si } n > 0 \\ \overline{x^{-n}} & \text{si } n < 0 \end{cases}$$

Lesquels de ces monoïdes sont des groupes ? $(\mathbb{N}, \times, 1)$, $(\mathbb{N}, +, 0)$, $(\mathbb{Z}, \times, 1)$, $(\mathbb{Z}, +, 0)$, $(\mathbb{Q}^*, \times, 1)$, $(\mathbb{Z}/p\mathbb{Z}, \times, 1)$, $(\mathbb{U}, \times, 1)$, $(\Sigma^*, \cdot, \varepsilon)$, $(\mathcal{M}_n(\mathbb{K}), \times, I_n)$, $(GL_n(\mathbb{K}), \times, I_n)$, $(\mathbb{K}[X], \times, 1)$, $(\mathrm{RatE}(\Sigma^*), \cdot, \varepsilon)$.

Exponentielle de matrices

Question

Soient A une matrice de taille $n \times n$ et b un entier positif.
Combien d'opération sont nécessaires pour calculer A^b ?

Réponse

Il faut $\Theta(\log b)$ multiplications pour calculer la puissance.
Or une multiplication matricielle demande n^3 multiplications scalaires.
A^b demande donc $\Theta(n^3 \log b)$.

Note

On peut multiplier deux matrices avec moins de n^3 opérations. Par exemple l'algorithme de Strassen le fait avec $\Theta(n^{log_2(7)})$ opérations (il y a encore mieux). Dans ce cas la complexité est bien sûr encore moindre.

Exponentielle de polynômes

Question

Soient $A(X)$ un polynôme de degré n et b un entier positif.
Combien d'opération sont nécessaires pour calculer $A^b(X)$?

Réponse

Il faut $\Theta(\log b)$ multiplications pour calculer la puissance.
Multiplier « à la main » deux polynômes de degrés i et j demande $\Theta(ij)$ opérations. On ne connaît pas tous les degrés qui vont être utilisés par l'algorithme d'exponentiation, mais au pire le plus haut degré est celui du polynôme final : bn.
Calculer $A^b(X)$ demande donc $\mathrm{O}((bn)^2 \log b)$ opérations.

Note

Avec la multiplication de Karatsuba, vue précédemment, on fait seulement $\mathrm{O}((bn)^{\log_2 3} \log b)$ opérations.

Cette approche est-elle optimale ?

Calculons a^{31}. On a $31 = \overline{11111}$.

Avec l'algo :

$a^2 = a \star a$,
$a^4 = a^2 \star a^2$,
$a^8 = a^4 \star a^4$,
$a^{16} = a^8 \star a^8$,
$a^{24} = a^{16} \star a^8$,
$a^{28} = a^{24} \star a^4$,
$a^{30} = a^{28} \star a^2$,
$a^{31} = a^{30} \star a$,
soit 8 opérations $\star$.

Plus vite :

$a^2 = a \star a$,
$a^3 = a^2 \star a$,
$a^5 = a^3 \star a^2$,
$a^{10} = a^5 \star a^5$,
$a^{11} = a^{10} \star a$,
$a^{21} = a^{11} \star a^{10}$,
$a^{31} = a^{21} \star a^{10}$,
soit 7 opérations $\star$.

Les puissances calculées forment une chaîne additive.

Chaînes additives

Un chaîne additive pour b est un séquence d'entiers $1 = p_1, p_2, p_3, \ldots, p_k = b$ telle que pour tout $i > 1$ il existe j et k dans $[\![1, i[\![$ tels que $p_i = p_j + p_k$.

$1, 2, 4, 8, 16, 24, 28, 30, 31$ et
$1, 2, 3, 5, 10, 11, 21, 31$ sont deux chaînes additives pour 31.

On montre que la taille de la chaîne additive la plus courte pour b est en $\Theta(\log b)$, ce qui signifie que l'algorithme d'exponentiation rapide est asymptotiquement optimal. L'exemple ci-dessus montre seulement que le nombre effectif de multiplications n'est pas optimal.

Chaînes additives/soustractives

Pour la puissance sur un groupe, on peut chercher une chaîne additive/soustractive.
Par exemple si $1, 2, 3, 5, 10, 11, 21, 31$ est la chaîne additive la plus courte pour 31, la chaîne additive/soustractive la plus courte est $1, 2, 4, 8, 16, 32, 31$. On en déduit :
$a^2 = a \star a$,
$a^4 = a^2 \star a^2$,
$a^8 = a^4 \star a^4$,
$a^{16} = a^8 \star a^8$,
$a^{32} = a^{16} \star a^{16}$,
$a^{31} = a^{32} \star \overline{a}$. Soit 6 opérations, mais un calcul de $\overline{a}$ en plus.
Sur $\mathbb{Q}^*$ où calculer $\overline{a}$ est immédiat, cette méthode a du sens.
Pour des matrices, $A^{32} \times A^{-1}$ est un peu plus coûteux...
Sur $(\mathbb{N}, \times, 1)$, où $\overline{a}$ n'est pas défini, on sait pourtant écrire $a^{31} = a^{32}/a$. Donc le groupe est la mauvaise abstraction.

Éléments simplifiables (1/2)

Un élément x d'un monoïde est dit

simplifiable à droite ssi $\forall y, \forall z,\ y \star x = z \star x \implies y = z$

simplifiable à gauche ssi $\forall y, \forall z,\ x \star y = x \star z \implies y = z$

simplifiable ssi il est simplifiable à gauche et à droite.

Dans $(\Sigma^*, \cdot, \varepsilon)$ tout élément est simplifiable : on peut par exemple noter $(a \cdot b)/b = a$ et $a\backslash(a \cdot b) = b$ les simplifications à droite et à gauche.

Dans $(\mathbb{N}, \times, 1)$ tout élément non nul est simplifiable. on note $(a \times b)/a = b$ et $(a \times b)/b = a$ sans distinguer droite et gauche parce que $\times$ est commutatif.

Dans $(\mathbb{K}[X], \times, 1)$ tout polynôme non nul est simplifiable.

Dans $(\mathcal{M}_n(\mathbb{K}), \times, I_n)$ seules les matrices inversibles sont simplifiables. Par ex. $\begin{bmatrix} 2 & 1 \\ -2 & -1 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ -2 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 3 & 0 \end{bmatrix}$

Éléments simplifiables (2/2)

L'exponentiation à l'aide de chaînes additives/soustractives marche pour tout élément simplifiable.

$a^2 = a \star a$,
$a^4 = a^2 \star a^2$,
$a^8 = a^4 \star a^4$,
$a^{16} = a^8 \star a^8$,
$a^{32} = a^{16} \star a^{16}$,
$a^{31} = a^{32}/a$.

Encore faut-il que l'opération de simplification soit peu coûteuse.

Poussons le bouchon...

En fait la propriété de simplifiabilité est plus que ce dont on a besoin pour pouvoir appliquer l'algorithme.

L'inverse de groupe (juste pour le fun)

Posons $A = \begin{bmatrix} 1 & 1 & 0 \\ 2 & 3 & 1 \\ 3 & 5 & 2 \end{bmatrix}$, et $A^{\#} = \frac{1}{16}\begin{bmatrix} 14 & 8 & -6 \\ -2 & 0 & 2 \\ -18 & -8 & 10 \end{bmatrix}$.

A et $A^{\#}$ ne sont pas inversibles. Pourtant, $\forall n > 0$, $A^{n+1} \times A^{\#} = A^n$.

$A^{\#}$ est l'*inverse de groupe* (ou *inverse de Drazin d'index 1*) de A.

$A^{\#}$ existe pour tout matrice telle que $\mathrm{rg}(A^2) = \mathrm{rg}(A)$.

On peut considérer qu'un endormorphisme f sur un espace vectoriel E tombe dans l'un des cas suivants (le second incluant le premier) :

- $\mathrm{Im}(f) = E$: f est inversible (et donc simplifiable) et $f^{\#} = f^{-1}$
- $\mathrm{Im}(f) \oplus \mathrm{Ker}(f) = E$: f n'est pas forcément inversible mais il existe un unique pseudo-inverse $f^{\#}$ tel que $ff^{\#}f = f$, $f^{\#}ff^{\#} = f^{\#}$, et $ff^{\#} = f^{\#}f$ (tout ceci implique en particulier $fff^{\#} = f$)
- $\mathrm{Im}(f) \cap \mathrm{Ker}(f) \neq \{\vec{0}\}$: $f^{\#}$ n'existe pas (on pourrait pousser le bouchon plus loin avec les *inverses de Drazin d'index k*)

Suite de Fibonacci

Introduction

En 1202, Fibonacci, un mathématicien italien, a posé la question suivante.

« Possédant initialement un couple de lapins, combien de couples obtient-on en douze mois si chaque couple engendre tous les mois un nouveau couple à compter du second mois de son existence ? »

Suite de Fibonacci

Définition

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2} \quad \text{pour } n \geq 2$$

Premiers termes

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, \ldots$

Problème

Pour un n donné, il faut calculer F_n.
Quel est le coût de ce calcul ?

Calcul récursif

Algorithme

```
RecFibonacci(n)
1   if n = 0 or n = 1
2           return n
3   else
4           return RecFibonacci(n − 1)
```

Complexité

$$T(n) = \begin{cases} 1 & \text{si } n < 2 \\ T(n-1) + T(n-2) & \end{cases}$$

Autrement dit calculer F_n avec cette méthode demande $T(n) = \Theta(F_{n+1})$ opérations...

Complexité du calcul récursif

$$T(n) = \begin{cases} 1 & \text{si } n < 2 \\ T(n-1) + T(n-2) \end{cases}$$

On peut encadrer cette valeur en posant $T''(n) < T(n) < T'(n)$ avec

- $T'(n) = 2T(n-1)$ donc $T'(n) = \Theta(2^n)$
- $T''(n) = 2T(n-2)$ donc $T''(n) = \Theta\left(2^{n/2}\right) = \Theta\left((\sqrt{2})^n\right)$

Finalement, on a $\Theta\left((\sqrt{2})^n\right) < T(n) < \Theta(2^n)$.
$T(n)$ (aussi bien que F_n) est bel et bien une fonction exponentielle.

Programmation dynamique

Personne ne programme Fibonacci récursivement.
Sa définition donne un algorithme itératif très naturellement. (En fait on met en pratique les principes de la programmation dynamique, mais c'est tellement naturel qu'on ne le réalise pas forcément.)

```
ProgDynFibonacci(n)
1  if n = 0 or n = 1
2        return n
3  a ← 0          // F_{i-1}
4  b ← 1          // F_i
5  for i ← 1 to n
6        (a, b) ← (b, a + b)
7  return b
```

Le calul de F_n demande maintenant $\Theta(n)$ opérations.

Les matrices débarquent...

La ligne $(a, b) \leftarrow (b, a + b)$ fait penser à une opération matricielle :

$$\begin{bmatrix} a \\ b \end{bmatrix} \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

On découvre que la suite de Fibonacci peut se définir matriciellement :

$$\begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} F_{n-2} \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n-1} \begin{bmatrix} F_0 \\ F_1 \end{bmatrix}$$

Soit finalement

$$\begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n-1} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Algorithme matriciel

$$\text{Posons} \quad \begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n-1} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

On a $F_n = d$ par définition.

Pour calculer F_n il suffit donc calculer $\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n-1}$.

En utilisant l'algorithme d'exponentiation rapide, cela demande $\Theta\left(2^3 \times \log(n-1)\right) = \Theta(\log n)$ opérations.

Le coût total de cette méthode est donc seulement de $\Theta(\log n)$.

Suites récurrentes linéaires

Ce type de transformation du problème sous forme de matrice s'applique à toutes les suites récurrentes linéaires.

Dans le cas de Fibonacci, qui n'est qu'un exemple de suite récurrente linéaire d'ordre 2, l'expression matricielle nous permet d'aller encore plus loin.

Diagonalisation

$A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ est une matrice symétrique à coefficients réels.

Théorème

Une matrice symétrique à coefficients réels est diagonalisable.

Si A est diagonalisable...

Il existe une matrice diagonale D et une matrice de passage P telles que $A = PDP^{-1}$. La diagonale de D est constituée des valeurs propre de A, et les colonnes de la matrice P sont les vecteurs propres de A.

Conséquence

$$A^n = \left(PDP^{-1}\right)^n = PD^nP^{-1}$$

Intérêt

D^n se calcule plus rapidement que A^n car il suffit de calculer les puissances des éléments de la diagonale.

Calcul des valeurs propres

λ est valeur propre si $A\vec{v} = \lambda\vec{v}$ pour $\vec{v} \neq \vec{0}$. En notant I la matrice identité on a :

$$\begin{aligned} & \exists \vec{v} \neq \vec{0}, \quad A\vec{v} = \lambda\vec{v} \\ \iff & \exists \vec{v} \neq \vec{0}, \quad (A - \lambda I)\vec{v} = 0 \\ \iff & \quad \det(A - \lambda I) = 0 \end{aligned}$$

or dans notre cas $\det(A - \lambda I) = \det \begin{bmatrix} -\lambda & 1 \\ 1 & 1-\lambda \end{bmatrix} = \lambda^2 - \lambda - 1.$

$$\iff \lambda^2 - \lambda - 1 = 0$$

Notons $\lambda = \dfrac{1+\sqrt{5}}{2}$ et $\lambda' = \dfrac{1-\sqrt{5}}{2}$ les racines de ce polynôme. Ce sont les valeurs propres de A et on sait maintenant que $D = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda' \end{bmatrix}$.

Calcul des vecteurs propres

Vecteur propre associé à λ

$$(A-\lambda I)\vec{v} = \vec{0} \iff \begin{bmatrix} -\lambda & 1 \\ 1 & 1-\lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0 \iff \begin{cases} x = y(\lambda - 1) \\ y = \lambda x \end{cases}$$

Une infinité de $\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix}$ résolvent cette équation, tous colinéaires.

Fixons $x = 1$. Alors $\vec{v} = \begin{bmatrix} 1 \\ \lambda \end{bmatrix}$.

Vecteur propre associé à λ'

Par un calcul identique $\vec{v}' = \begin{bmatrix} 1 \\ \lambda' \end{bmatrix}$.

Matrice de passage

On sait maintenant que $P = [\vec{v}; \vec{v}'] = \begin{bmatrix} 1 & 1 \\ \lambda & \lambda' \end{bmatrix}$.

Tous les ingrédients sont là

Calcul de P^{-1}

On inverse $P = \begin{bmatrix} 1 & 1 \\ \lambda & \lambda' \end{bmatrix}$, pour trouver $P^{-1} = \frac{1}{\lambda' - \lambda} \begin{bmatrix} \lambda' & -1 \\ -\lambda & 1 \end{bmatrix}$.

Calcul de $A^n = PD^nP^{-1}$

$$
\begin{aligned}
A^n &= \begin{bmatrix} 1 & 1 \\ \lambda & \lambda' \end{bmatrix} \begin{bmatrix} \lambda & 0 \\ 0 & \lambda' \end{bmatrix}^n \frac{1}{\lambda' - \lambda} \begin{bmatrix} \lambda' & -1 \\ -\lambda' & 1 \end{bmatrix} \\
&= \frac{1}{\lambda' - \lambda} \begin{bmatrix} 1 & 1 \\ \lambda & \lambda' \end{bmatrix} \begin{bmatrix} \lambda^n & 0 \\ 0 & \lambda'^n \end{bmatrix} \begin{bmatrix} \lambda' & -1 \\ -\lambda & 1 \end{bmatrix} \\
&= \frac{1}{\lambda' - \lambda} \begin{bmatrix} \lambda^n\lambda' - \lambda'^n\lambda & \lambda'^n - \lambda^n \\ \lambda^{n+1}\lambda' - \lambda'^{n+1}\lambda & \lambda'^{n+1} - \lambda^{n+1} \end{bmatrix}
\end{aligned}
$$

Rappel : si $A^{n-1} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, alors $F_n = d$.

Valeur de F_n

$$F_n = \frac{\lambda'^n - \lambda^n}{\lambda' - \lambda} \quad \text{or } \lambda - \lambda' = \sqrt{5}$$

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

Cette dernière équation est connue sous le nom de formule de Binet. Le calcul de F_n se fait toujours en $\Theta(\log n)$ à cause du calcul des puissances. Cette formule nous donne cependant une réponse plus satisfaisante pour la complexité de l'algo RecFibonacci, donc on avait dit qu'elle était $T(n) = \Theta(F_n)$. On sait maintenant que $F_n = \Theta\left(\left(\frac{1+\sqrt{5}}{2} \right)^n \right)$.

Ce nombre, $\frac{1+\sqrt{5}}{2}$, est le nombre d'or.

Autres façons de calculer F_n

Nous sommes passés par une diagonalisation pour trouver que

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Il y a d'autres façons d'y arriver.

- Une première est d'étudier la fonction génératrice de la suite. Cette technique est assez puissante et a beaucoup d'applications.
- Une seconde est d'étudier les équations de récurrentes linéaires

Fonction génératrice (1/2)

La méthode :

- On considère les termes F_n commes les coefficients d'une série : $F(x) = \sum_{n\geq 0} F_n x^n$. ($F(x)$ est la *fonction génératrice* de F_n.)
- À l'aide de la relation $F_{n+2} = F_{n+1} + F_n$, on cherche à déduire une équation définissant $F(x)$.
  Pour cela, on multiplie chaque côté par x^n, puis on somme sur tout n : $\sum_{n\geq 0} F_{n+2}x^n = \sum_{n\geq 0} F_{n+1}x^n + \sum_{n\geq 0} F_n x^n$, enfin on cherche à y faire apparaître $F(x)$. Ici on trouve que
  $\sum_{n\geq 0} F_{n+2}x^n = (F(x) - xF(1) - F(0))/x^2 = (F(x) - x)/x^2$
  $\sum_{n\geq 0} F_{n+1}x^n = (F(x) - F(0))/x = F(x)/x$
  $\sum_{n\geq 0} F_n x^n = F(x)$
  On déduit que $(F(x) - x)/x^2 = F(x)/x + F(x)$ d'où
  $F(x) = x/(1 - x - x^2)$.
- On développe $F(x)$ sous forme de série pour trouver les coefs.

Fonction génératrice (2/2)

Notons $r = (-1+\sqrt{5})/2$ et $r' = (-1-\sqrt{5})/2$ les racines de $1-x-x^2$.

$$F(x) = \frac{x}{1-x-x^2} = \frac{-x}{(x-r)(x-r')} = \frac{1}{r-r'}\left(\frac{-r}{x-r} + \frac{r'}{x-r'}\right)$$

Posons $\varphi = 1/r = (1+\sqrt{5})/2$ et $\varphi' = 1/r' = (1-\sqrt{5})/2$, ainsi :

$$F(x) = \frac{1}{\sqrt{5}}\left(\frac{1}{1-\varphi x} - \frac{1}{1-\varphi' x}\right)$$

On se rappelle que $\frac{1}{1-x} = \sum_{n\geq 0} x^n$ donc

$$F(x) = \frac{1}{\sqrt{5}}\left(\sum_{n\geq 0} \varphi^n x^n - \sum_{n\geq 0} \varphi'^n x^n\right)$$

Comme on a posé $F(x) = \sum_{n\geq 0} T_n x^n$ on retrouve finalement que $F_n = (\varphi^n - \varphi'^n)/\sqrt{5}$. Cette méthode de calcul est très puissante.

Équation de récurrence linéaire (1/2)

Toute relation de récurrence linéaire (d'ordre k) de la forme $f(n) = a_1 f(n-1) + a_2 f(n-2) + \cdots + a_k f(n-k)$ où les a_i sont des constantes, possède une solution qui est une *combinaison linéaire* de fonctions de la forme $f(n) = x^n$.
En remplaçant $f(n)$ par x^n, on a
$x^n = a_1 x^{n-1} + a_2 x n - 2 + \cdots + a_k x^{n-k}$, puis en divisant par x^{n-k} :

$$x^k - a_1 x^{k-1} - a_2 x^{k-2} + \cdots + a_k = 0$$

C'est le polynôme *caractéristique* de la relation de récurrence linéaire.

Les récurrences du premier ordre sont faciles à résoudre, on parle là de suites géométriques (niveau lycée).

Le second ordre est plus pénible.

Équation de récurrence linéaire (2/2)

Pour une récurrence d'ordre 2, notons r_1 et r_2 les racines du polynôme caractéristique $x^2 - a_1 x^k - a_2 = 0$.
Si $r_1 \neq r_2$ alors la solution est de la forme $f(n) = c_1 r_1^n + c_2 r_2^n$.
Si $r_1 = r_2 = r$ alors la solution est de la forme $f(n) = c_1 r^n + c_2 n r^n$.
Les valeurs c_1 et c_2 sont à trouver en fonction des termes de la suite.

Application à Fibonacci. On a $x^2 - x - 1 = 0$ on trouve que $r_1 = (1 + \sqrt{5})/2$ et $r_2 = (1 - \sqrt{5})/2$. Reste à résoudre

$$\begin{cases} f(0) = c_1 + c_2 = 0 \\ f(1) = c_1 r_1 + c_2 r_2 = 1 \end{cases}$$

On trouve rapidement que $c_1 = 1/\sqrt{5}$ et $c_2 = -c_1$, d'où $f(n) = (r_1^n - r_2^n)/\sqrt{5}$. Facile.

Pour les récurrences d'ordre supérieur, documentez-vous.

Utilité de F_n

La suite de Fibonacci a de nombreuses applications.

En calcul de complexité

Par exemple **Théorème de Lamé :** pour tout $k \geq 1$ si $a > b \geq 0$ et $b < F_{k+1}$ alors Euclide(a,b) engendre moins de k appels récursifs. Comme $F_k = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$ on en déduit que l'algorithme tourne en $\mathrm{O}(\log(b))$ opérations.

En dénombrement

Par exemple compter le nombre de pavages d'un rectangle de $2 \times n$ avec des dominos de 2×1.

Ou bien le nombre de sous ensembles de $1, 2, \ldots, n$ qu'on peut réaliser sans prendre deux nombres consécutifs.

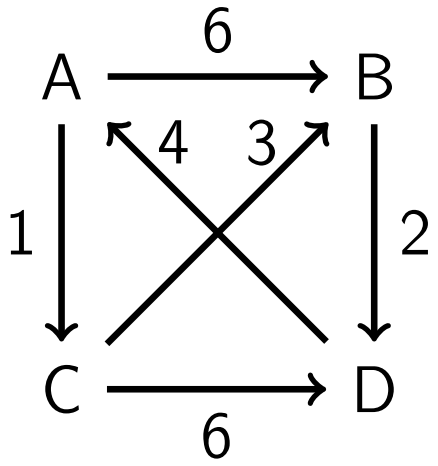
Ou encore le nombre de codes barre de n millimètres que l'on peut composer avec des barres de tailles 1mm ou 2mm.

Plus courts chemins

Plus courts chemins

On considère un graphe orienté, dans lequel les arcs ont des poids qui représentent par exemple des distances ou des durées de trajet.

G :



$$M_G = \begin{array}{c} \begin{matrix} A & B & C & D \end{matrix} \\ \begin{bmatrix} 0 & 6 & 1 & \infty \\ \infty & 0 & \infty & 2 \\ \infty & 3 & 0 & 6 \\ 4 & \infty & \infty & 0 \end{bmatrix} \end{array} \begin{matrix} A \\ B \\ C \\ D \end{matrix}$$

On souhaite pouvoir répondre à toutes les questions de la forme « quel est le plus court chemin de X à Y ». On veut faire les calculs une bonne fois pour toutes, pas à chaque question.

Première approche en programmation dynamique

On a une sous-structure optimale car si

$$X \to n_1 \to n_2 \to \cdots \to n_{k-1} \to n_k \to Y$$

est un plus court chemin entre X et Y, alors

$$X \to n_1 \to n_2 \to \cdots \to n_{k-1} \to n_k$$

est un plus court chemin entre X et n_k.

On note $D[X, Y, k]$ la distance minimale des chemins entre X et Y qui utilisent au plus k arcs.

$$D[X, Y, k] = \begin{cases} M[X, Y] & \text{si } k = 1 \\ \min_Z(D[X, Z, k-1] + D[Z, Y, 1]) & \text{sinon} \end{cases}$$

$$= \begin{cases} M[X, Y] & \text{si } k = 1 \\ \min_Z(D[X, Z, k-1] + M[Z, Y]) & \text{sinon} \end{cases}$$

Algorithme (1/2)

S'il y a n nœuds, le chemin le plus long fait au plus $n-1$ arcs. On veut donc connaître $D[X, Y, n-1]$ pour tout X, Y.

$$D[X, Y, k] = \begin{cases} M[X, Y] & \text{si } k = 1 \\ \min_Z(D[X, Z, k-1] + M[Z, Y]) & \text{sinon} \end{cases}$$

Pour k fixé il est clair que l'algorithme peut oublier toutes les valeurs $D[X, Y, k]$ une fois qu'il a calculé $D[X, Y, k+1]$. On aurait plutôt intérêt à écrire :

$$D_k[X, Y] = \begin{cases} M[X, Y] & \text{si } k = 1 \\ \min_Z(D_{k-1}[X, Z] + M[Z, Y]) & \text{sinon} \end{cases}$$

et dire qu'on veut calculer D_{n-1}. On n'utilisera donc qu'un tableau D de $n \times n$ valeurs pour stocker D_k et un tableau temporaire D' pour calculer D_{k+1}.

Algorithme (2/2)

```
SlowShortestPath(M)
 1  n ← size(M)
 2  D ← M
 3  for k ← 2 to n − 1
 4        for X ← 1 to n
 5              for Y ← 1 to n
 6                    m ← ∞
 7                    for Z ← 1 to n
 8                          m ← min(m, D[X, Z] + M[Z, Y])
 9                    D′[X, Y] ← m
10        D ← D′
11  return D
```

C'est un algorithme en $\Theta(n^4)$.

Application et remarque

$$M = D_1 = \begin{bmatrix} 0 & 6 & 1 & \infty \\ \infty & 0 & \infty & 2 \\ \infty & 3 & 0 & 6 \\ 4 & \infty & \infty & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 4 & 1 & 7 \\ 6 & 0 & \infty & 2 \\ 10 & 3 & 0 & 5 \\ 4 & 10 & 5 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 4 & 1 & 6 \\ 6 & 0 & 7 & 2 \\ 9 & 3 & 0 & 5 \\ 4 & 8 & 5 & 0 \end{bmatrix}$$

$\min\{6+4, 0+10, 3+5, \infty+0\} = 8$

Observons le calcul de $D_3[i,j] = \min_k(D_2[i,k] + D_1[k,j])$.

Pour calculer D_3 on a fait une sorte de produit de matrices entre D_2 et D_1, dans lequel les opérations $\times$ et $+$ ont été remplacées respectivement par $+$ et min.

Comparez cette formule à $D^3_{i,j} = \sum_k D^2_{i,k} \times D^1_{k,j}$.

En fait nous calculons les puissances de M, mais avec des opérations différentes.

Anneaux et semi-anneaux

$(E, \oplus, \otimes, e, f)$ est un semi-anneau si

- $(E, \oplus, e)$ est un monoïde commutatif
- $(E, \otimes, f)$ est un monoïde
- $\otimes$ est distributif par rapport à $\oplus$
- e est absorbant pour $\otimes$ (i.e. $\forall x \in E, x \otimes e = e \otimes x = e$)

$(E, \oplus, \otimes, e, f)$ est un anneau si

- $(E, \oplus, e)$ est un groupe commutatif
- $(E, \otimes, f)$ est un monoïde
- $\otimes$ est distributif par rapport à $\oplus$

Dans ce cas e est forcément absorbant pour $\otimes$.

Lesquels de ces semi-anneaux sont des anneaux ? $(\mathbb{N}, +, \times, 0, 1)$, $(\mathbb{Z}, +, \times, 0, 1)$, $(\mathbb{B}, \vee, \wedge, \bot, \top)$, $(\mathbb{Q}, +, \times, 0, 1)$, $(\mathbb{K}[X], +, \times, 0, 1)$, $(\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$, $(\mathbb{N} \cup \{\infty\}, \max, \min, 0, \infty)$,

Matrices

Pour un semi-anneau (resp. anneau) $(E, \oplus, \otimes, e, f)$ et un entier $n > 0$, alors la structure $(\mathcal{M}_n(E), +, \times, I, O)$ où $c = a \times b$, $d = a + b$, I et O sont définis par

$$c_{i,j} = \bigoplus_{k=1}^{n} (a_{i,k} \otimes b_{j,k})$$

$$d_{i,j} = a_{i,j} + b_{i,j}$$

$$I_{i,j} = \begin{cases} f & \text{si } i = j \\ e & \text{sinon} \end{cases}$$

$$O_{i,j} = e$$

est un semi-anneau (resp. anneau).
En particulier cela signifie que $(\mathcal{M}_n(E), \times, I)$ est un monoïde et qu'on peut y calculer la puissance avec l'algorithme d'exponentiation rapide.

Puissances de matrices et semi-anneaux particuliers

$S = (\mathbb{R}, +, \times, 0, 1)$

Les matrices de $\mathcal{M}_n(S)$ décrivent les applications linéaires de $\mathbb{R}^n$ dans $\mathbb{R}^n$. Leurs puissances donnent les itérées.

$S = (\mathbb{N}, +, \times, 0, 1)$

Les matrices à valeurs dans $\{0, 1\}$ représentent des graphes non pondérés. Leurs puissances (à valeurs dans $\mathbb{N}$) comptent les chemins entre deux sommets.

$S = (\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$ (dit *semi-anneau tropical*)

Les matrices de $\mathcal{M}_n(S)$ représentent des cartes de distances (éventuellement négatives). Les puissances successives considèrent des séquences de plus en plus longues. C'est notre cas.

$S = (\mathbb{N} \cup \{\infty\}, \max, \min, 0, \infty)$

Les matrices de $\mathcal{M}_n(S)$ représentent des cartes de flots, mais où le flot n'emprunte qu'un chemin (pas de séparation).

$S = (\mathbb{B}, \vee, \wedge, \bot, \top)$ ... à vous de jouer !

Accelération du calcul des plus courtes distances

On sait maintenant qu'on veut calculer $D_{n-1} = M^{n-1}$ sachant que les éléments de M utilisent les lois de $(\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$.
Au lieu d'utiliser SlowShortestPath(M), qui est en $\Theta(n^4)$ on va utiliser FastPower($M, n-1$), qui est en $\Theta(n^3 \log n)$ sur les matrices. [1]

Si le graphe ne contient aucun cycle de poids négatifs, alors on est sûr que $\forall b \geq n-1$, $A^b = A^{n-1}$. Cela signifie qu'il est inutile de calculer A^{n-1} exactement : on peut le dépasser. Avec FastPower il sera plus rapide est de calculer la puissance de 2 suivante : FastPower($M, 2^{\lceil \log(n-1) \rceil}$) car le nombre $2^{\lceil \log(n-1) \rceil}$ ne possède qu'un seul bit à 1 dans sa représentation binaire.

1. L'optimisation du produit matriciel avec par exemple l'algorithme de Strassen en $\Theta(n^{\log_2(3)})$ au lieu de $\Theta(n^3)$ est aussi valable avec notre semi-anneau tropical.

Algorithme en $\Theta(n^3 \log n)$

```
FastShortestPath(M)
 1   n ← size(M)
 2   D ← M
 3   for k ← 2 to ⌈log(n − 1)⌉
 4           for X ← 1 to n
 5                   for Y ← 1 to n
 6                           m ← ∞
 7                           for Z ← 1 to n
 8                                   m ← min(m, D[X, Z] + D[Z, X])
 9                           D′[X, Y] ← m
10           D ← D′
11   return D
```

Comment retrouver le chemin ?

Pour chaque paire (X, Y) on retient le père de Y. Notons le $P[X, Y]$. Le meilleurs chemin est alors $X \to P[X, P[X, \cdots P[X, P[X, Y]]]] \to \cdots \to P[X, P[X, Y]] \to P[X, Y] \to Y$.

```
FastShortestPath(M)
 1  n ← size(M)
 2  D ← M
 4  for X ← 1 to n
 5    for Y ← 1 to n
 6      P[X, Y] ← X
 7  for k ← 2 to ⌈log(n − 1)⌉
 8    for X ← 1 to n
 9      for Y ← 1 to n
10        m ← ∞
11        for Z ← 1 to n
12          if D[X, Z] + D[Z, X] < m
13            m ← D[X, Z] + D[Z, X]
14            P[X, Y] ← P[m, Y]
15        D′[X, Y] ← m
16    D ← D′
17  return D, P
```

Deuxième approche en programmation dynamique

Un peu comme dans le problème de la loutre et ses 10kg de poisson, on considère le problème avec un nombre de nœuds croissant. Imaginons les nœuds du graphe numérotés de 1 à n et notons maintenant $D_K[X, Y]$ la plus courte distance entre X et Y en passant seulement par les nœuds $\leq K$.

$$D_0[X, Y] = M[X, Y]$$
$$D_K[X, Y] = \min(\underbrace{D_{K-1}[X, K] + D_{K-1}[K, Y]}_{\text{on passe par } K}, \underbrace{D_{K-1}[X, Y]}_{\text{on ne passe pas par } K})$$

Ceci suggère un algorithme de complexité $\Theta(n^3)$ en temps et $\Theta(n^2)$ en espace.

L'algorithme de Floyd-Warshall

FloydWarshall(M)

1 $n \leftarrow \text{size}(M)$
2 $D \leftarrow M$
3 for $K \leftarrow 1$ to n
4 for $X \leftarrow 1$ to n
5 for $Y \leftarrow 1$ to n
6 $D'[X, Y] \leftarrow \min(D[X, K] + D[K, Y], D[X, Y])$
7 $D \leftarrow D'$
8 return D

Application

$$M = D_0 = \begin{bmatrix} 0 & 6 & 1 & \infty \\ \infty & 0 & \infty & 2 \\ \infty & 3 & 0 & 6 \\ 4 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 6 & 1 & \infty \\ \infty & 0 & \infty & 2 \\ \infty & 3 & 0 & 6 \\ 4 & 10 & 5 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 6 & 1 & 8 \\ \infty & 0 & \infty & 2 \\ \infty & 3 & 0 & 5 \\ 4 & 10 & 5 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 4 & 1 & 6 \\ \infty & 0 & \infty & 2 \\ \infty & 3 & 0 & 5 \\ 4 & 8 & 5 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 4 & 1 & 6 \\ 6 & 0 & 7 & 2 \\ 9 & 3 & 0 & 5 \\ 4 & 8 & 5 & 0 \end{bmatrix}$$

Retrouver le chemin ?

Comme dans l'algorithme précédent, on note $P[X, Y]$ le père de Y dans le chemin le plus court de X à Y.

```
FloydWarshall(M)
1   n ← size(M)
2   D ← M
3   for X ← 1 to n
4     for Y ← 1 to n
5       P[X, Y] ← X
6   for K ← 1 to n
7     for X ← 1 to n
8       for Y ← 1 to n
9         if D[X, K] + D[K, Y] < D[X, Y]
10          D'[X, Y] ← D[X, K] + D[K, Y]
11          P[X, Y] ← P[K, Y]
12        else
13          D'[X, Y] ← D[X, Y]
14    D ← D'
15  return D, P
```