Théorie des Graphes : Plus courts chemins

Souheib Baarir
sbaarir@lrde.epita.fr

26 février 2015

Deuxième partie II

Plus courts chemins

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 $\overline{d_{n-1} \ldots d_1 d_0}$ la représentation d'un nombre de n bits en base 2. Donc, $b = \overline{d_{n-1} \ldots d_1 d_0} = \sum_i d_i 2^i$.

Exponentiation

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

$$a^b = a^{\overline{d_{n-1} \ldots d_0}} = a^{\sum_i d_i 2^i} = \prod_{d_i = 1} a^{2^i}$$

Par exemple $5^{19} = 5^{\overline{10011}} = 5^{2^0} \times 5^{2^1} \times 5^{2^4}$.
$5^{19} = 5 \times 25 \times 152587890625$
Combien ce calcul demande-t-il de multiplications ?

Exponentiation rapide (Algo)

FastPower(a, b)

$h = 1$
$t = a$
if $b = 0$ then return h
while (b != 0)
    if $odd(b)$ then $h = h * t$
    $b = \lfloor b/2 \rfloor$
    $t = t^2$
return h

Exponentiation rapide (Algo)

FastPower(a, b)

$h = 1$
$t = a$
if $b = 0$ then return h
while (b != 0)
    if $odd(b)$ then $h = h * t$
    $b = \lfloor b/2 \rfloor$
    $t = t^2$
return h

Complexité : $\Theta(\log b)$.

Exponentiation rapide (Algo)

FastPower(a, b)

```
h = 1
t = a
if b = 0 then return h
for i = 0 to ⌊log(b)⌋
        if odd(b) then h = h * t
        b = ⌊b/2⌋
        t = t^2
return h
```

Complexité : $\Theta(\log b)$.

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.

Application sur les matrices d'entiers relatifs : $(\mathcal{M}_n(\mathbb{Z}), \times, I_n)$

Question

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

Application sur les matrices d'entiers relatifs : $(\mathcal{M}_n(\mathbb{Z}), \times, I_n)$

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 $\Theta(n^3 \log b)$.

Peut-on généraliser encore plus ? (1/2)

Peut-on généraliser encore plus ? (1/2)

Oui ! Grâce à la notion de **semi-anneaux.**

Peut-on généraliser encore plus ? (1/2)

Oui ! Grâce à la notion de **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. $\forall x \in E, x \otimes e = e \otimes x = e$)

Peut-on généraliser encore plus ? (2/2)

Pour un semi-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

$$
\begin{aligned}
c_{i,j} &= \bigoplus_{k=1}^{n} (a_{i,k} \otimes b_{j,k}) \\
d_{i,j} &= a_{i,j} + b_{i,j} \\
I_{i,j} &= \begin{cases} f & \text{si } i = j \\ e & \text{sinon} \end{cases} \\
O_{i,j} &= e
\end{aligned}
$$

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

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.

$$G : \quad \begin{tikzcd} A \arrow[r, "6"] \arrow[d, "1"'] & B \arrow[d, "2"] \\ C \arrow[r, "6"'] \arrow[ur, "3"'] & D \arrow[ul, "4"] \end{tikzcd} \qquad M_G = \begin{array}{c} \\ \\ \\ \\ \end{array}\!\!\begin{array}{cc} \begin{array}{cccc} A & B & C & D \end{array} & \\ \begin{bmatrix} 0 & 6 & 1 & \infty \\ \infty & 0 & \infty & 2 \\ \infty & 3 & 0 & 6 \\ 4 & \infty & \infty & 0 \end{bmatrix} & \begin{array}{c} A \\ B \\ C \\ D \end{array} \end{array}$$

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.

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

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.

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

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 :

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

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)
  n ← size(M)
  D ← M
  for k ← 2 to n − 1
      for X ← 1 to n
          for Y ← 1 to n
              m ← ∞
              for Z ← 1 to n
                  m ← min(m, D[X, Z] + M[Z, Y])
              D'[X, Y] ← m
      D ← D'
  return D
```

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

Application et remarque

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

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

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

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

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

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

Comparez cette formule à
$D_3[i,j] = \sum_k D_2[i,k] \times D_1[k,j]$.

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

Comparez cette formule à $D_3[i,j] = \sum_k D_2[i,k] \times D_1[k,j]$.

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.

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

Comparez cette formule à $D_3[i,j] = \sum_k D_2[i,k] \times D_1[k,j]$.

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.

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

Accélération 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 du semi-anneau $S = (\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$, et donc M est un élément du monoïde $(\mathcal{M}_n(S), \times, I_n)$.

$\Rightarrow$ 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.

Accélération 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 du semi-anneau $S = (\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$, et donc M est un élément du monoïde $(\mathcal{M}_n(S), \times, I_n)$.

$\Rightarrow$ 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.

Note : 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($M, 2^{\lceil \log(n-1) \rceil}$)

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

```
FastShortestPath(M)
  n ← size(M)
  D ← M
  for k ← 2 to ⌈log(n − 1)⌉
      for X ← 1 to n
          for Y ← 1 to n
              m ← ∞
              for Z ← 1 to n
                  m ← min(m, D[X, Z] + D[Z, X])
              D′[X, Y] ← m
      D ← D′
  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 $X \rightarrow P[X, P[X, \cdots P[X, P[X, Y]]]] \rightarrow \cdots \rightarrow P[X, P[X, Y]] \rightarrow P[X, Y] \rightarrow Y$.

FastShortestPath(M)

$n \leftarrow \mathrm{size}(M)$
$D \leftarrow M$
for $X \leftarrow 1$ to n
  for $Y \leftarrow 1$ to n
    $P[X, Y] \leftarrow X$
for $k \leftarrow 2$ to $\lceil \log(n-1) \rceil$
  for $X \leftarrow 1$ to n
    for $Y \leftarrow 1$ to n
      $m \leftarrow \infty$
      for $Z \leftarrow 1$ to n
        if $D[X, Z] + D[Z, Y] < m$
          $m \leftarrow D[X, Z] + D[Z, Y]$
          $P[X, Y] \leftarrow Z$
      $D'[X, Y] \leftarrow m$
  $D \leftarrow D'$
return D, P

Deuxième approche en programmation dynamique (1/2)

Soit $S = \{1, 2, 3, 4, \ldots, n\}$ l'ensemble des sommets du graphe et soient i et j deux sommets de S. On considère un chemin c entre i et j de poids minimal dont les sommets intermédiaires sont dans $\{1, 2, 3, \ldots, k\}$.

L'algorithme de Floyd-Warshall est basé sur l'observation suivante :

- soit c n'emprunte pas le sommet k ;
- soit c emprunte exactement une fois le sommet k et c est donc la concaténation de deux chemins, entre i, k et k, j respectivement, dont les sommets intermédiaires sont dans $\{1, 2, 3, \ldots, k-1\}$.

Deuxième approche en programmation dynamique (2/2)

Notons maintenant D_K la matrice telle que $D_K[X, Y]$ est la plus courte distance entre X et Y en passant seulement par les sommets de $\{1, 2, 3, \ldots, k\}$. L'observation précédente se traduit alors par :

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

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

L'algorithme de Floyd-Warshall

```
FloydWarshall(M)
n ← size(M)
D ← M
for K ← 1 to n
    for X ← 1 to n
        for Y ← 1 to n
            D'[X, Y] ← min(D[X, Y], D[X, K] + D[K, Y])
    D ← D'
return D
```

Application

$$M = D_0 = \begin{bmatrix} 0 & 6 & 1 & \infty \\ \infty & 0 & \infty & 2 \\ \infty & 3 & 0 & 6 \\ 4 & \infty & \infty & 0 \end{bmatrix} \qquad D_3 = \begin{bmatrix} 0 & 4 & 1 & 6 \\ \infty & 0 & \infty & 2 \\ \infty & 3 & 0 & 5 \\ 4 & 8 & 5 & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 6 & 1 & \infty \\ \infty & 0 & \infty & 2 \\ \infty & 3 & 0 & 6 \\ 4 & 10 & 5 & 0 \end{bmatrix} \qquad D_4 = \begin{bmatrix} 0 & 4 & 1 & 6 \\ 6 & 0 & 7 & 2 \\ 9 & 3 & 0 & 5 \\ 4 & 8 & 5 & 0 \end{bmatrix}$$

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

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)
  n ← size(M)
  D ← M
  for X ← 1 to n
    for Y ← 1 to n
      P[X, Y] ← X
  for K ← 1 to n
    for X ← 1 to n
      for Y ← 1 to n
        if D[X, K] + D[K, Y] < D[X, Y]
          D′[X, Y] ← D[X, K] + D[K, Y]
          P[X, Y] ← P[K, Y]
        else
          D′[X, Y] ← D[X, Y]
    D ← D′
  return D, P
```