Algorithmique 2 (INFO-F203)
Programmation dynamique

Jean Cardinal

Mai 2021

Fibonacci

$$
\begin{aligned}
F(1) &= 1, \\
F(2) &= 1, \\
F(n) &= F(n-1) + F(n-2).
\end{aligned}
$$

```
int F(int n) {
  if (n==1 || n==2) return 1;
  else return F(n-1) + F(n-2);
  }
```

Fibonacci

```
int F(int n) {
   int n1 = 0, n2 = 1, c = 0;
   while (c < n) {
      int n3 = n2 + n1;
      n1 = n2;
      n2 = n3;
      c++;
   }
   return n1;
}
```

Programmation dynamique

En retenant les valeurs précédentes, on évite de refaire les calculs déjà effectués, et la complexité passe d'exponentielle à linéaire !
Les deux ingrédients principaux d'un algorithme de programmation dynamique sont :

1. une **table**,
2. une **relation de récurrence**.

Plus longue sous-séquence commune

On se donne deux chaînes A et B de symboles dans un alphabet donné, et on souhaite trouver une sous-séquence commune aux deux chaînes de longueur maximum.

Exemple 1

- h**ou**seboa**t**
- c**o**mp**ut**er

(ou encore **oue**.)

Table

On définit $L_{i,j}$ comme la longueur de la plus longue sous-séquence commune aux deux préfixes de longueurs i et j de A et B.

Récurrence

$L_{i,0} = 0$ pour tout $i \geq 0$, $L_{0,j} = 0$ pour tout $j \geq 0$, et

$$L_{i,j} = \begin{cases} L_{i-1,j-1} + 1 & \text{si } A_i = B_j \\ \max\{L_{i-1,j}, L_{i,j-1}\} & \text{sinon.} \end{cases}$$

Produits de matrices

- n matrices notées M_i, $i = 1, 2, \ldots, n$.
- M_i a r_{i-1} lignes et r_i colonnes.
- On souhaite calculer le produit

$$\prod_{i=1}^{n} M_i$$

  de façon à minimiser le nombre de multiplications (de nombres) à effectuer.
- On dispose de la procédure de calcul du produit de *deux* matrices, disons A de taille $j \times k$ et B de taille $k \times \ell$, qui effectue $jk\ell$ multiplications de nombres.
- Quelle est la meilleure façon de calculer le produits des n matrices M_i ?
- Identifier un *parenthésage* optimal, qui minimisera le nombre total de multiplications.

Exemple

On souhaite calculer le produit $M = M_1 \times M_2 \times M_3 \times M_4$, où les tailles de matrices sont : $M_1 : 2 \times 4$, $M_2 : 4 \times 8$, $M_3 : 8 \times 1$, et $M_4 : 1 \times 12$. Si l'on calcule $M = M_1 \times (M_2 \times (M_3 \times M_4))$, on effectuera 576 multiplications. En revanche, l'ordre suivant :

$$M = \underbrace{\underbrace{(M_1 \times (\underbrace{M_2 \times M_3}_{4\times8\times1}))}_{2\times4\times1} \times M_4}_{2\times1\times12}$$

ne nécessite que $32 + 8 + 24 = 64$ multiplications !

Dénombrement

Rappel Mathématique 1

Le nombre de parenthésage distincts d'un produit de n facteurs est égal au *nombre de Catalan* C_{n-1}, avec

$$C_n = \frac{1}{n+1}\binom{2n}{n},$$

et asymptotiquement :

$$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}.$$

Le nombre C_n est égal au nombre d'arbres binaires complets à $n+1$ feuilles. La bijection entre les parenthésages et les arbres binaires complets est immédiate, chaque paire de parenthèses correspondant à un sous-arbre.

Table

La programmation dynamique va nous permettre de réduire cette complexité à un polynôme. Afin d'établir une notion de “sous-problème”, notons $m_{i,j}$ le coût minimum de l'évaluation du produit

$$M_i \times \ldots \times M_j,$$

pour $1 \leq i \leq j \leq n$.

Récurrence

Proposition 1

Cette quantité vérifie la récurrence suivante :

$$m_{i,j} = \begin{cases} 0 & \text{si } i = j \\ \min_{i \leq k < j}\{m_{i,k} + m_{k+1,j} + r_{i-1} r_k r_j\} & \text{si } i < j. \end{cases}$$

Exemple

Pour notre exemple avec $n = 4$, et $r_0 = 2$, $r_1 = 4$, $r_2 = 8$, $r_3 = 1$ et $r_4 = 12$, on obtient le tableau suivant :

	1	2	3	4
1	0	64	40	64
2		0	32	80
3			0	96
4				0

Algorithme

Proposition 2

Le problème de parenthésage du produit de n matrices peut se résoudre en temps $O(n^3)$.

Code

```
int produit (int r[], int n) {
  int m[n + 1][n + 1];
  for (int i = 0; i <= n; ++i) m[i][i] = 0;

  for (int l = 1; l <= n - 1; ++l)
    for (int i = 1; i <= n - 1; ++i) {
      int j = i + l, min = +MAXINT;
      for (int k = i; k < j; ++k) {
        int test = m[i][k] + m[k+1][j] + r[i-1]
             * r[k] * r[j];
        if (test < min) min = test;
      }
      m[i][j] = min;
    }
    return m[1][n];
}
```

Problème du sac à dos

étant donnés n objets, chacun associé à un poids p_i et une valeur v_i, et un poids maximum M, quel est la valeur totale maximum d'un sous-ensemble de ces objets de poids total au plus M ?

Exemple

On se donne $n = 4$ objets, avec les poids et valeurs suivantes :

$$\begin{aligned} p_1 &= 2 & v_1 &= 4 \\ p_2 &= 1 & v_2 &= 3 \\ p_3 &= 4 & v_3 &= 7 \\ p_4 &= 3 & v_4 &= 4, \end{aligned}$$

et un poids maximum $M = 5$. Le sous-ensemble $\{1, 4\}$ donne un poids total $p_1 + p_4 = 2 + 3 = 5 \leq M$, et une valeur $v_1 + v_4 = 4 + 4 = 8$. Mais une meilleure solution est de choisir les objets $\{2, 3\}$, avec un poids total de 5 mais une valeur de 10.

Table

On considère les sous-problèmes paramétrés par deux entiers $i \leq n$ et $\ell \leq M$, et qui consistent à trouver la meilleure solution restreinte aux i premiers objets et avec un poids maximum ℓ. Notons $C_{i,\ell}$ la valeur de la meilleure solution pour ce sous-problème. On suppose ici que tous les nombres définissant le problème (poids et valeurs) sont des nombres entiers.

Récurrence

Proposition 3

$C_{0,\ell} = 0$ pour tout ℓ, et pour $i \geq 1$:

$$C_{i,\ell} = \begin{cases} \max\{C_{i-1,\ell-p_i} + v_i, C_{i-1,\ell}\} & \text{si } p_i \leq \ell \\ C_{i-1,\ell} & \text{sinon.} \end{cases}$$

Complexité

En remplissant une table C avec les valeurs optimales pour chaque sous-problème, on obtient un algorithme de complexité $O(n \times M)$. Cet algorithme est efficace dans les cas où M est une valeur suffisamment petite. En revanche, ce n'est pas un algorithme de complexité polynomiale.

Complexité polynomiale ?

Rappel Mathématique 2

Un algorithme est dit de complexité *polynomiale* si celle-ci est bornée supérieurement par un polynôme de degré constant en la taille des données en entrée.

Dans le problème du sac à dos, on peut exprimer la taille des données en entrée par le nombre de symboles nécessaires pour encoder les poids et les valeurs des objets, et le poids maximum M. Or, pour écrire le nombre M, nous n'avons besoin que de $O(\log M)$ symboles – par exemple exactement $\lceil \log_2 M \rceil$ bits. Une complexité proportionnelle à M est donc bien **exponentielle** en la taille des données en entrée ! Un algorithme dont la complexité est bornée par un polynôme en la *valeur* des données en entrée est dit **pseudo-polynomial**. Ces algorithmes ne sont donc efficaces que si ces valeurs sont suffisamment petites.