Programmation Dynamique

Sandrine Vial
sandrine.vial@prism.uvsq.fr

Janvier 2014

Introduction

- Programmation Dynamique :
  Stratégie de résolution de problèmes semblable à diviser pour régner mais les sous-problèmes ne sont pas indépendants.
- Idée Centrale :
  On mémorise les solutions des sous-problèmes pour ne pas les recalculer.
- Application : **Problème d'optimisation**

Sac à dos à valeurs entières

- Données :
  - N objets
  - Chaque objet i : Poids w_i et Valeur c_i
  - Un sac à dos contenant un poids de W.
- Question : Maximiser la valeur des objets contenus dans le sac à dos.

Solutions

- Variante fractionnaire :
  Résolution par un algorithme glouton :
  - Pour chaque objet, on calcule $c_i / w_i = k_i$ et on prend les objets dans l'ordre décroissant de leur k_i.
- Variante du tout ou rien :
  - Algorithme glouton ne fonctionne plus
  - Utilisation programmation dynamique.

Solution : Programmation Dynamique

- $C(v, i)$: Meilleur cout pour remplir un sac de taille v avec les i premiers objets.
- Notre problème s'écrira : $C(W, N)$
- La récurrence :

$$C(v, i) = \max \begin{cases} C(v, i-1) & \text{si l'on ne prend pas l'ob} \\ C(v - w_i, i-1) + c_i & \text{si l'on prend l'objet } i \end{cases}$$

Multiplication de matrices

- Données : une suite $< A_1, A_2, \ldots, A_n >$ de n matrices
- Question : Calculer $A_1 \times A_2 \times \cdots \times A_n$ avec le moins possible de multiplications scalaires.

Propriétés

- Multiplication de n matrices :
  - Multiplication associative
  - N'importe quel parenthèsage aboutit au meme résultat.
- Sous-problème :
- Multiplication de deux matrices.

Exemple

- 4 matrices : A_1, A_2, A_3, A_4
- Résultat : $A_1 \times A_2 \times A_3 \times A_4$.
- Parenthèsages possibles :

$$
\begin{array}{l}
(A_1 \times (A_2 \times (A_3 \times A_4))) \\
(A_1 \times ((A_2 \times A_3) \times A_4) \\
((A_1 \times A_2) \times (A_3 \times A_4)) \\
((A_1 \times (A_2 \times A_3)) \times A_4) \\
(((A_1 \times A_2) \times A_3) \times A_4)
\end{array}
$$

Algorithme

Algorithme 1 Multiplication de 2 matrices A et B. $C = A \times B$

```
Début
si colonnes(A) = lignes(B)
    pour i de 1 à lignes(A) faire
        pour j de 1 à colonnes(B) faire
            C[i][j] ← 0
            pour k de 1 à colonnes(A) faire
                C[i][j] ← C[i][j] + A[i][k] × B[k][j]
            fin pour
        fin pour
    fin pour
fin si
Fin
```

Evaluation du cout

- A de dimensions : $p \times q$
- B de dimensions : $q \times r$
- $C = A \times B$ de dimensions : $p \times r$.
- Temps de calcul dépend du nombre de multiplications scalaires : $p \times q \times r$

Evaluation : exemple

- Trois matrices : $A_1(10 \times 100); A_2(100 \times 5); A_3(5 \times 50)$.
- $((A_1 \times A_2) \times A_3)$:
  - Cout $(A_1 \times A_2) = 10 \times 100 \times 5 = 5000$
  - Cout $(A_1 A_2 \times A_3) = 10 \times 5 \times 50 = 2500$
  - Total $=$ 7500 multiplications scalaires.
- $(A_1 \times (A_2 \times A_3))$:
  - Cout $(A_2 \times A_3) = 100 \times 5 \times 50 = 25000$
  - Cout $(A_1 \times A_2 A_3) = 10 \times 100 \times 50 = 50000$
  - Total $=$ 75000 multiplications scalaires.

Facteur 10 entre les 2 solutions !

Enoncé du problème

- Données :
  - Une suite $< A_1, A_2, \ldots, A_n >$ de n matrices
  - Matrice A_i de dimensions : $p_{i-1} \times p_i$
- Question :
  - Parenthéser $A_1 \times A_2 \times \cdots \times A_n$ de façon à minimiser le nombre de multiplications scalaires.

Nombre de parenthésages

- $P(n)$: nombre de parenthésages d'une suite de n matrices.
- $P(1) = 1$
- $P(n) = \sum_{k=1}^{n-1} P(k) \times P(n-k)$
- $P(n) = C(n-1)$: nombre de catalan où $C(n) = \frac{1}{n+1} C_{2n}^{n} = \Omega(4^n / n^{3/2})$

Nombre de solutions est donc exponentiel en n

Structure d'un parenthésage optimal

- $A_{i..j} = A_i \times A_{i+1} \times \ldots \times A_{j-1} \times A_j$
- Parenthésage optimal sépare le produit $A_1 \times A_2 \times \cdots \times A_n$ en 2 entre A_k et A_{k+1}
- Pour une valeur de k :
  Cout Final = Cout du calcul de $A_{1..k}$ + Cout du calcul de $A_{k+1..n}$ + Cout multiplication de $A_{1..k} \times A_{k+1..n}$.

Solution Récursive

On cherche à définir récursivement la valeur d'une solution optimale en fonction des solutions optimales aux sous-problèmes.

- $m[i,j]$ = nbe min de multiplications pour le calcul de $A_{i..j}$
- $m[1,n]$ = manière la plus économique pour calculer $A_{1..n}$

Définition récursive :

- $m[i,i] = 0$, $A_{i..i} = A_i$, pas de multiplications.
- $m[i,j] = \min_{i \leq k < j}(m[i,k] + m[k+1,j] + p_{i-1} \times p_j \times p_k)$ si $i < j$.
- $s[i,j] = k$: valeur de k correspondant au $m[i,j]$ optimal.

Nbe de sous-problèmes est réduit : un sous-pb pour chaque chois de i et j : de l'ordre de n^2.

Algorithme

En entrée : la suite de dimensions des matrices $p_0 p_1 \ldots p_n$.

Algorithme 2 Ordonner Suite de Matrices

```
Début
pour i de 1 à n
    m[i,i] ← 0
fin pour
pour l de 2 à n  faire
    pour i de 1 à n - l + 1  faire
        j ← i + l - 1
        m[i,j] ← ∞
        pour k de i à j - 1  faire
            q ← m[i,k] + m[k+1,j] + p_{i-1} × p_k × p_j
            si q < m[i,j]
                m[i,j] ← q
                s[i,j] ← k
            fin si
        fin pour
    fin pour
fin pour
Fin
```

Construction de la solution optimale

Algorithme 3 Multiplier (A,s,i,j)

```
Début
si i > j
    X ← Multiplier(A,s,i,s[i,j])
    Y ← Multiplier(A,s,s[i,j]+1,j)
    retourner MultiplierMatrice(X,Y)
sinon
    retourner A_i
fin si
Fin
```