IFT436 - Série d'exercices #7 : programmation dynamique

Manuel Lafond

Pour chacun des exercices, je recommande d'écrire:

a. une récurrence permettant de définir une solution optimale en fonction de sous-instances;

b. une procédure récursive qui implémente cette récurrence;

c. une procédure de backtracking qui trouve une solution optimale concrète;

d. une version itérative de la procédure récursive.

Exercice 1: Considérez le problème du sous-tableau contigu maximum de la série précédente (trouver un sous-tableau $[t_i, t_{i+1}, \ldots, t_j]$ de somme maximum dans un tableau T). Donnez un algorithme en temps $O(n)$ de programmation dynamique pour ce problème.

Exercice 2: Vous avez un commerce de vente de bâtons de bois, et le prix d'un bâton dépend de sa longueur. On vous donne un tableau de prix T, où $T[i]$ représente le prix de vente d'un bâton de longueur i. Vous avez un bâton de bois de longeur ℓ que vous pouvez couper en plusieurs morceaux de tailles différentes. Vous voulez trouver le découpage qui va maximiser la somme des prix de vente des sous-bâtons.
Par exemple, si $T = [1, 3, 5, 7]$ et $\ell = 10$, vous pouvez couper votre bâton en 3 morceaux: deux de taille 3 et un de taille 4, pour un prix total de 17. Vous pourriez aussi couper deux bâtons de taille 4 et un de taille 2, totalisant aussi 17.

Donnez un algorithme qui calcule le prix de vente maximum que l'on peut atteindre étant donné un tableau de prix T et un bâton de longueur ℓ. La complexité de votre algorithme peut dépendre de $|T|$ et ℓ.

Exercice 3: On se rappelle du problème de la sous-somme: on reçoit en entrée un tableau d'entiers $S = \{s_1, \ldots, s_n\}$ et une somme cible t, et on veut savoir s'il existe un sous-ensemble $S' \subseteq S$ tel que $\sum_{s \in S'} s = t$.
Donnez un algorithme en temps $O(n^2 t)$ qui résout ce problème.

Indice: ceci est similaire au problème de la monnaie, mais on a le droit d'utiliser chaque élément de S une seule fois. Une façon est de calculer une table D, où $D[k, j]$ exprime si oui ou non on peut atteindre une somme de j en utilisant seulement les éléments $\{s_1, \ldots, s_k\}$. Trouvez une récurrence permettant de calculer D.

Exercice 4: Soient A et B deux matrices, respectivement de dimensions $n \times m$ et $m \times p$. Le produit AB donne une matrice C de dimension $n \times p$, où $c_{i,j}$ est défini par $\sum_{i=1}^{m} a_{i,k} \cdot b_{k,j}$. Le calcul de toutes les entrées de C nécessite $n \cdot m \cdot p$ opérations.

Supposons maintenant qu'on veut calculer le produit de k matrices, disons le produit $A_1 A_2 A_3 \ldots A_k$, où pour tout $i \in \{1, \ldots, k-1\}$, la matrice A_i est de dimension $n_i \times m_i$ et A_{i+1} est de dimension $m_i \times m_{i+1}$. Le produit de matrices est associatif, c'est- a-dire que l'on peut appliquer n'importe quel parenthésage valide et effectuer les produit dans l'ordre prescrit par ces parenthésages.

Les parenthésages ne donnent pas tous la même complexité totale. Par exemple, prenons le produit ABC, où A est de dimension 10×60, B de dimension 60×5 et C de dimension 5×20. Le produit $(AB)C$ nécessite $10 \cdot 60 \cdot 5 + 10 \cdot 5 \cdot 20 = 4000$ opérations. Le produit $A(BC)$ nécessite $60 \cdot 5 \cdot 20 + 10 \cdot 60 \cdot 20 = 18,000$ opérations.

Donnez un algorithme qui retourne le nombre minimum d'opérations à effectuer pour multiplier $A_1 A_2 \ldots A_k$.

Indice: considérez les $O(n^2)$ façons de séparer $A_1 A_2 \ldots A_k$ en deux parties, puis combinez les paires de résultats obtenues pour garder la minimum. Mémorisez les valeurs pour éviter de les recalculer plusieurs fois.

Exercice 5: Considérez l'algorithme pour trouver un ensemble indépendant de taille maximum dans un arbre. La version des notes de cours ne fait que retourner la taille, mais ne donne pas d'ensemble indépendant. Écrivez une procédure permettant de retrouver un ensemble indépendant concret au lieu de seulement sa taille. (recommandé: calculez M_0 et M_1 comme dans les notes de cours, puis commencez de la racine jusqu'aux feuilles pour construire une

solution).

Exercice 6: Soit S une chaîne de caractères. Une *sous-séquence* de S est une chaîne dont les caractères apparaissent dans le même ordre que dans S, mais pas nécessairement de façon contigüe. Par exemple, si $S = salutations$, alors $luttons$ est une sous-séquence de S. Dit autrement, une sous-séquence est une chaîne qu'on peut obtenir de S en y supprimant certains positions.
Soient S et T deux chaînes de caractères de la même longueur n. Dans le problème de la *plus longue sous-séquence commune*, on cherche la plus longue chaîne qui est à la fois une sous-séquence de S et de T.
Donnez un algorithme en temps $O(n^2)$ pour ce problème.
Indice. Pas trivial comme problème! Exprimez $M[i, j]$ comme la plus longue sous-séquence commune des chaînes $S[1..i]$ et $T[1..j]$. Considérez les cas possibles des caractères $S[i]$ et $T[j]$ pour émettre votre récurrence.

Exercice 7: Considérez le problème du commis voyageur. Il existe un algorithme de programmation dynamique pour résoudre le problème en temps $O(n^2 2^n)$ (ce qui est nettement mieux que l'algorithme naïf $O(nn!)$). L'idée est de choisir un sommet de départ x arbitraire et de calculer, pour chaque sous-ensemble S et pour chaque sommet $s \in S$, le coût minimum d'un chemin qui démarre à x, passe une fois par tous les sommets de S et termine à s. Donnez un algorithme en temps $O(n^2 2^n)$ pour le problème du commis voyageur.