INF431

Programmation dynamique
CORRIGÉ
Version: 243:2470M

Comme vu en cours, la programmation dynamique permet de calculer des solutions optimales à des problèmes pour lesquels ces solutions peuvent se construire à partir de solutions optimales de sous-problèmes. L'exercice consiste alors à découvrir la bonne décomposition du problème donné.

1 Travaillez efficacement

La fin de l'année approche. Vous devez réviser m matières différentes ; ces matières sont désignées par les numéros de 1 à m.

Dans chaque matière, vous obtiendrez une note entière comprise entre 0 et g avec $g > 1$. Vous avez pu déterminer, pour chaque matière, une fonction qui vous donne votre note à partir du nombre d'heures de révision que vous consacrez à cette matière. Si vous travaillez la matière i pendant h heures, vous obtiendrez la note $p_i(h)$. Pour simplifier, on suppose que l'on ne peut pas diviser les heures, c'est-à-dire que h est entier. On peut supposer que les fonctions p_i sont croissantes.

Il ne vous reste plus que n heures avant les examens, et vous devez les répartir au mieux entre les matières. C'est-à-dire que vous devez trouver pour chaque matière i un nombre h_i d'heures tel que :

$$\sum_{i=1}^{m} h_i = n \qquad \text{et} \qquad \sum_{i=1}^{m} p_i(h_i) \quad \text{est maximal.}$$

Question 1 Écrire le pseudo-code d'un algorithme qui calcule le nombre maximal de points que vous pouvez obtenir. Donnez la complexité en temps de votre algorithme, qui doit être polynomiale en n et en m. ◇

Solution. Soit $M_{i,j}$ le nombre maximal de points qu'on peut obtenir avec les matières $1, \dots, i$ et j heures de travail. Si, sur j heures de travail, on décide d'en consacrer ℓ à la matière i, alors on peut obtenir au mieux $M_{i-1,j-\ell}$ points sur les matières $1, \dots, i-1$. Ceci donne la base de la décomposition pour la programmation dynamique :

$$M_{0,j} = 0, \qquad M_{i,j} = \max_{\ell \le j}(p_i(\ell) + M_{i-1,j-\ell}) \quad \text{pour } i > 0.$$

On peut alors remplir itérativement un tableau à deux dimensions contenant les valeurs $M_{i,j}$.

procédure SCOREMAX(p, m, n)
 pour $j = 0$ **à** n
  faire $M[0, j] := 0$;
 pour $i = 1$ **à** m
  faire { **pour** $j = 0$ **à** n
   faire { $M[i, j] := 0$;
    pour $\ell = 0$ **à** j
     faire { $score := p[i](\ell) + M[i-1, j-\ell]$;
      si $M[i, j] < score$
      alors $M[i, j] := score$
 renvoyer $M[m, n]$

FIGURE 1 – Une plus longue sous-séquence commune à A,G,T,T,A,C,G et C,G,A,T,A,A,G est A,T,A,G.

Il y a trois boucles imbriquées au cœur desquelles un nombre constant d'opérations est effectué. La complexité est donc de $O(n^2 \cdot m)$ opérations.

Remarques :

— Il est possible de n'utiliser un tableau qu'à une dimension (en remarquant qu'on n'accède qu'à la dernière ligne.

— L'algorithme glouton, qui cherche n fois à trouver la matière où l'heure suivante donne le plus de points est faux. Par exemple si en Maths la 3e heure de révision donne beaucoup de points, alors que en Physique on gagne 2 points pour chaque heure supplémentaire. □

Question 2 A partir de là, écrivez le pseudo-code d'un algorithme qui calcule effectivement les h_i correspondant à la répartition optimale entre les matières. ◇

Solution. Il y a deux manières naturelles d'obtenir le résultat. L'une consiste à construire en même temps que le tableau M un tableau qui garde trace dans sa case (i, j) du nombre d'heures à affecter à la matière j si on dispose de i heures au total. Une autre solution souvent plus commode (mais parfois un peu plus coûteuse) consiste à reconstruire une solution optimale à l'aide du tableau M :

procédure SELECTION(p, m, n, M)
pour $(i = m, j = n; i > 0; i - -;)$
faire { **pour** $\ell = 0$ **à** j
faire { $score := p[i](\ell) + M[i-1, j-\ell]$;
si $M[i, j] = score$
alors $\{Sol[i] := \ell; j := j - \ell; break$;
renvoyer Sol □

2 Plus longue sous-séquence commune

Une mesure de similarité entre deux séquences d'ADN est la longueur de leur plus longue sous-séquence commune. Précisément, si les deux séquences sont $x_1, \ldots, x_n$ et $y_1, \ldots, y_m$ (où les x_i et y_i appartiennent à l'alphabet $\{A, C, G, T\}$), il s'agit de trouver l'entier k maximal tel qu'existent deux suites d'indices $j_0 < j_1 < \cdots < j_{k-1}$ et $\ell_0 < \ell_1 < \cdots < \ell_{k-1}$ avec $x_{j_i} = y_{\ell_i}$ pour tout $0 \leq i < k$.

Question 3 Écrire le pseudo-code d'une solution par programmation dynamique en temps $\mathrm{O}(nm)$ au problème de la plus longue sous-séquence commune. ◇

Solution. On raisonne sur les sous-problèmes où les séquences sont $(x_1, \ldots, x_i)$ et $(y_1, \ldots, y_j)$ avec $i \leq n$ et $j \leq m$. Si les dernières lettres ne coïncident pas ($x_i \neq y_j$), alors il faut prendre la plus longue sous-séquence commune entre $x_1, \ldots, x_i$ et $y_1, \ldots, y_{j-1}$ ou entre $x_1, \ldots, x_{i-1}$ et $y_1, \ldots, y_j$. Sinon, il faut aussi considérer la possibilité d'augmenter de 1 la longueur de la plus longue sous-séquence commune de $x_1, \ldots, x_{i-1}$ et $y_1, \ldots, y_{j-1}$. On peut toujours choisir en effet de faire correspondre les deux dernières lettres sans perdre l'optimalité.

procédure LONGESTSUBSEQ(X, Y, n, m)

pour $i = 0$ **à** n
  faire $c[i, 0] := 0$;
pour $j = 0$ **à** m
  faire $c[0, j] := 0$;
pour $i = 1$ **à** n
  faire pour $j = 1$ **à** m
  faire $\begin{cases} \textbf{si } X[i] \neq Y[j] \\ \textbf{alors } c[i, j] := \max(c[i-1, j], c[i, j-1]); \\ \textbf{sinon } c[i, j] := c[i-1, j-1] + 1; \end{cases}$
renvoyer $c[n, m]$

□

3 Arbres de recherche binaires optimaux

Un arbre binaire de recherche est une structure de données pour des clés ordonnées. Il s'agit d'un arbre binaire, dont les nœuds contiennent des clés. Les clés inférieures à celle de la racine sont stockées dans des nœuds de son sous-arbre gauche, alors que les clés supérieures sont stockées dans son sous-arbre droit, la même règle étant appliquée récursivement (voir l'exemple Fig. 2 où les clés sont des lettres). Tester la présence d'une clé est très simple : soit elle est à la racine et la recherche est terminée, soit elle est inférieure à la clef de la racine et il faut la chercher récursivement dans le sous-arbre gauche, soit enfin il faut la chercher dans le sous-arbre droit.

Si un arbre binaire de recherche est utilisé pour de la vérification orthographique, avec les mots pour clés, il est avantageux de tenir compte des fréquences des mots dans la langue pour faire apparaître les plus fréquents vers la racine. Plus précisément, étant données n clés $x_1 < \cdots < x_n$ et leurs fréquences $f_1, \ldots, f_n$, il s'agit de construire un arbre binaire de recherche sur ces clés qui minimise le coût $f_1 d_1 + \cdots + f_n d_n$, où d_i représente la profondeur du nœud contenant la clé x_i (la profondeur de la racine étant 1).

Question 4 Écrire le pseudo-code d'une procédure calculant par programmation dynamique le coût de l'arbre binaire optimal, et estimer sa complexité. [Indication : les clés appartenant à un même sous-arbre sont consécutives.] ◇

Solution. Si la racine de l'arbre optimal construit sur les clés consécutives $x_i, \ldots, x_j$ est la clé $x_k \in [x_i, x_j]$, alors les arbres gauches et droits sont les arbres optimaux pour les clés $x_i, \ldots, x_{k-1}$ et $x_{k+1}, \ldots, x_j$. Leurs coûts c_g et c_d augmentent lorsqu'ils deviennent sous-arbres puisque la profondeur de chacun de leurs nœuds s'accroît de 1, et le coût de l'arbre est donc

$$f_k + (c_g + f_i + \cdots + f_{k-1}) + (c_d + f_{k+1} + \cdots + f_j) = c_g + c_d + f_i + \cdots + f_j.$$

On construit à l'aide de cette formule un tableau $c[i, j]$ donnant le coût optimal de l'arbre de recherche binaire construit sur les clés $x_i, \ldots, x_j$, en procédant par tailles croissantes.

procédure OPTIMALBINARYSEARCHTREE(x, f, n)
pour $i = 1$ **à** $n + 1$
  faire $c[i, i-1] = 0$;
pour $\ell = 1$ **à** n
  faire pour $i = 1$ **à** $n - \ell$
  faire $\begin{cases} j := i + \ell; plus := 0; cost := maxint; \\ \textbf{pour } k = i \textbf{ à } i + \ell \\ \quad \textbf{faire} \begin{cases} plus := plus + f[k]; \\ s := c[i, k-1] + c[k+1, j]; \\ \textbf{si } s < cost \\ \textbf{alors } cost := s; opt := k; \end{cases} \\ c[i, j] := cost + plus; \end{cases}$
renvoyer $c[1, n]$;

□

Du fait des 3 boucles imbriquées, la complexité est en $O(n^3)$.

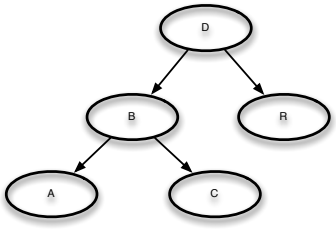

FIGURE 2 – Arbre binaire de recherche

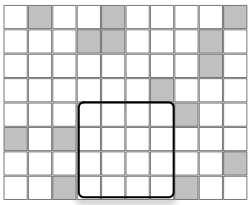

FIGURE 3 – Un plus grand sous-carré monochromatique sur une matrice à deux symboles *blanc* et *noir*.

4 Plus grand sous-carré monochromatique

Étant donné un tableau M de dimension $n \times m$ dont les entrées sont des couleurs (représentées par des entiers), un sous-carré monochromatique de côté k est donné par $i \leq n - k$ et $j \leq m - k$ tels que tous les $M[a, b]$ sont égaux pour $i \leq a < i + k$, $j \leq b < j + k$.

Question 5 Écrire le pseudo-code d'une procédure qui, étant donné M, renvoie un plus grand sous-carré monochromatique. Estimer sa complexité. ◇

Solution. Comme dans les exercices précédents, on raisonne sur des sous-problèmes. Ici, il s'agit de ramener le plus grand sous-carré monochromatique dans M aux plus grands carrés monochromatiques dans les sous-tableaux de M partant du coin supérieur gauche (de coordonnées $(1, 1)$). Si l'une des trois cases $(i - 1, j)$, $(i, j - 1)$ et $(i - 1, j - 1)$ n'a pas la même couleur que la case (i, j) alors le plus grand carré ayant cette case pour coin inférieur droit a taille 1. Sinon, si les plus grands carrés ayant pour coin inférieur droit les cases $(i - 1, j)$, $(i, j - 1)$ et $(i - 1, j - 1)$ ont tailles k, ℓ et m, alors le plus grand carré ayant (i, j) pour coin inférieur droit a taille $\min(k, \ell, m) + 1$.

Le code suivant s'en déduit :

procédure LARGESTSQUARE(M, n, m)
pour $j = 1$ **à** m
  faire $c[1, j] := 1$;
pour $i = 1$ **à** n
  faire $c[i, 1] := 1$;
$res := 1$;
pour $i = 1$ **à** n
  faire pour $j = 1$ **à** m
  faire si $M[i, j] = M[i - 1, j]$ **et** $M[i, j] = M[i, j - 1]$ **et** $M[i, j] = M[i - 1, j - 1]$
alors $\begin{cases} c[i, j] := \min(c[i - 1, j], c[i, j - 1], c[i - 1, j - 1]) + 1; \\ res := \max(res, c[i, j]) \end{cases}$
sinon $c[i, j] = 1$;
renvoyer res;

La complexité de l'algorithme est en $\mathrm{O}(nm)$. □

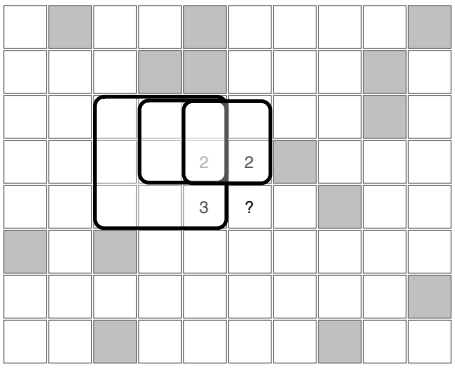

FIGURE 4 – Illustration de c_{ij}.