

TD : Programmation Dynamique

Olivier Raynaud

raynaud@isima.fr

Résumé

Exercice 1 (Pièces de monnaies).

Considérons le problème consistant à rendre la monnaie de S euros avec le moins de pièces possibles à prendre dans un jeu de pièces (a_1, a_2, \dots, a_n) .

Question 1. *Proposer un algorithme naïf pour résoudre ce problème. Existe-t-il des instances pour lesquelles la solution n'est pas optimale ou ne peut être produite par votre algorithme.*

Question 2. *En appliquant le paradigme de la programmation dynamique, concevoir un algorithme qui retourne une solution optimale pour n'importe quel ensemble de pièces.*

Question 3. *Insérer dans votre algorithme les instructions pour afficher 'Rendre k pièce de type i ' pour $k \geq 0$ et $0 \leq i \leq n$.*

Exercice 2 (Justification d'un texte en ligne).

Soit une suite de mots $(m_0, m_1, \dots, m_{n-1})$ que l'on désire couper en lignes de longueur L . On note $l(m)$ la longueur du mot m . Les mots sont séparés par des espaces dont la taille idéale est e , mais qui peuvent être comprimés ou étirés au besoin (sans provoquer de chevauchement de mots), de manière que la ligne m_i, m_{i+1}, m_j ait exactement la longueur L . Cependant on attribue une pénalité d'étirement ou de compression égale à la valeur absolue totale des étirements ou des compressions subies par les blancs de la ligne. Autrement dit le coût de la mise en forme de la ligne m_i, m_{i+1}, m_j est $|e' - e|$ où e' , la longueur réelle des espaces est :

$$(L - l(m_i) - \dots - l(m_j)) / (j - i)$$

Cependant, si $j = n - 1$ (on atteint la dernière ligne), le coût est nul à moins que $e' < e$, puisque on a pas à jouer sur l'élasticité des espaces de cette dernière ligne.

Le problème consiste à formater un texte de façon à avoir un coût minimal.

Question 1. Concevoir un algorithme de programmation dynamique pour répondre à ce problème.

Exercice 3 (Multiplication d'une chaîne de matrices).

Question 1. Trouver un parenthésage optimal du produit d'une suite de matrice dont la séquence de dimension est $\langle 5, 10, 3, 12, 5, 50, 6 \rangle$.

Question 2. Donner un algorithme efficace d'imprimer parenthésage permettant d'imprimer le parenthésage optimal d'une suite de matrice, connaissant le tableau $s[]$ calculé par l'algorithme donné en cours. Analyser votre algorithme.

Question 3. Soit $R(i, j)$ le nombre de fois que l'élément $m[i, j]$ est référencé par l'algorithme Ordonné Chaînes de Matrices pendant le calcul des autres éléments du tableau. Montrer que le nombre total de références pour le tableau tout entier vaut $(n^3 - n)/3$.

Question 4. Montrer qu'un parenthésage complet d'une expression à n éléments contient exactement $(n - 1)$ paires de parenthèses.

Exercice 4 (Sous-suite croissante maximale).

Soit une séquence s de nombres x_1, x_2, \dots, x_n entiers tous distincts. Une sous-suite est un sous-ensemble de ces nombres pris dans l'ordre de s , c'est à dire une séquence de la forme $x_{i_1}, x_{i_2}, \dots, x_{i_k}$ telle que $i_1 \leq i_2 \leq \dots \leq i_k$. Une sous suite est dite croissante si les x_i forment une suite croissante.

On cherche à calculer la plus longue sous-suite croissante de s . Par exemple pour $s = (6, 2, 5, 4, 9, 7, 1, 8, 5)$, une solution est $(2, 5, 7, 8)$.

Question 1. Proposer un algorithme naïf pour résoudre ce problème. Donner sa complexité.

Question 2. Soit une séquence s , soit $lg[i]$ la longueur de la plus longue sous-suite croissante dont le dernier élément est x_i . Vérifier les relations de récurrence suivantes :

- $lg[1] = 1$;
- $lg[i] = 1 + \text{Max}_{j < i \text{ \& } s_j < s_i} (lg[j])$;

Question 3. Concevoir un algorithme de programmation dynamique qui calcule la taille de la plus longue sous-suite d'une séquence s . Donner sa complexité.

Question 4. Modifier votre algorithme pour qu'il affiche la sous-suite de longueur maximale. La complexité est-elle changée ?

Exercice 5 (Sous-suite maximale).

On considère un tableau $tab[]$ d'entiers relatifs de taille N . On veut déterminer la valeur maximale de la somme d'une sous-séquence d'éléments de $tab[]$.

Question 1. Proposer un algorithme naïf qui répond au problème.

Question 2. Proposer un algorithme de type Diviser pour régner.

Propriété 1. Soient T un tableau d'entiers de taille n , u, v et w trois indices appartenant à l'intervalle $[1..n]$ tels que $u < v < w$ et $\forall i \in [(v+1)..w] T[i] \leq 0$. Si $S[u, v] = \max_{i \in [1..v]} S[i..v]$ et $S[u, v] + S[(v+1)..w] < 0$ alors il n'existe pas de séquence maximale contenant la sous-séquence $[(v+1)..w]$.

Question 3. Démontrer la propriété 1.

Question 4. En déduire un algorithme optimal pour résoudre le problème de la sous-suite maximale.

Exercice 6 (Plus longue sous-séquence commune).

On considère deux séquences (textes formés de lettres) S et T . Leurs longueurs sont notées respectivement m et n . Une sous-séquence d'un texte est obtenu en enlevant certaines lettres du texte. Par exemple le mot 'algorithmique' admet comme sous-séquence le mot 'gothique'. Une sous séquence commune à S et T est une séquence qui est sous-séquence des deux à la fois. On cherche à déterminer la plus longue sous-séquence commune (PLSSC) à S et T .

Pour cela on construit un tableau $LPLSSC[][]$ (avec $1 \leq i \leq m$ et $1 \leq j \leq n$) tel que $LPLSSC[i][j]$ est la plus longueur de la plus longue sous-séquence :

- commune à S et T ;
- terminant avant la i^{ieme} lettre de S (inclusivement) ;
- terminant avant la j^{ieme} lettre de T (inclusivement).

Par exemple pour $S = \text{algorithmique}$ et $T = \text{programmation}$, on a $LPLSSC[4][3] = 1$ ('o') et $LPLSSC[13][13] = 4$ ('orti').

Question 1. Notons $Q = Q[1]...Q[k]$ une plus longue sous-séquence commune à S et T . Montrer que :

- si $S[m] = T[n]$ alors $Q[k] = S[m] = T[n]$ et $Q[1..k-1]$ est la plus longue sous-séquence commune à $S[1..m-1]$ et $T[1..n-1]$;
- si $S[m] \neq T[n]$ et $Q[k] \neq S[m]$ alors Q est la plus longue sous-séquence commune à $S[1..m-1]$ et T ;

- si $S[m] \neq T[n]$ et $Q[k] \neq T[n]$ alors Q est la plus longue sous-séquence commune à S et $T[1..n-1]$.

Question 2. En déduire la formule de récurrence générale qui permet de calculer $LPLSSC[i][j]$ en fonction de $LPLSSC[i-1][j-1]$, $LPLSSC[i][j-1]$, $LPLSSC[i-1][j]$. Ne pas oublier de donner son initialisation.

Question 3. Concevoir un algorithme de programmation dynamique qui calcule la longueur d'une plus grande sous-séquences de deux séquences S et T .

Question 4. Comment trouver une plus longue sous-séquence commune en mémorisant des informations complémentaires.

Exercice 7 (Distance d'édition).

La distance d'édition entre deux mots est le nombre d'opérations élémentaires qu'il faut effectuer pour passer d'un mot à l'autre. Les opérations élémentaires possibles sont :

- insertion d'un caractère ;
- suppression d'un caractère ;
- substitution d'un caractère par un autre ;

Par exemple la distance d'édition entre les mots 'tourte' et 'tartre' est de 3. Le schéma de transformation à partir des opérations élémentaires définit ci-dessus est le suivant :

'tourte' \longrightarrow 'taurte' \longrightarrow 'tarte' \longrightarrow 'tartre'.

Le caractère 'o' est substitué en 'a', puis le caractère 'u' est supprimé et enfin le caractère 'r' est inséré.

Question 1. Soient A et B deux chaînes de caractères, et $\text{dist}(A, B)$ la fonction de distance d'édition entre A et B . Ecrire les relations de récurrence suivants les différents cas possibles : insertion, suppression, substitution. En déduire un algorithme récursif de la fonction $\text{dist}()$.

Question 2. Concevoir un algorithme de calcul de la distance d'édition entre deux chaînes de caractères en programmation dynamique sans appel récursif. Donner la complexité de votre algorithme.

Question 3. Exécuter votre algorithme pour calculer la distance d'édition entre les mots tourte et tartre. Retrouver la liste des opérations qui ont été exécutées pour passer du premier mot au second.

Question 4. Concevoir un algorithme récursif qui affiche la liste des opérations élémentaires effectuées pour passer du premier mot au second.

Exercice 8 (Tri topologique).

Question 1. Proposer un algorithme linéaire de tri des sommets d'un graphe orienté sans circuit (DAG).

Question 2. Donner la liste des sommets obtenue par application de votre algorithme sur le graphe de la Figure 1. On supposera qu'à chaque étape nécessitant un choix arbitraire de sommets, ce choix se fait suivant l'ordre alphabétique des étiquettes.

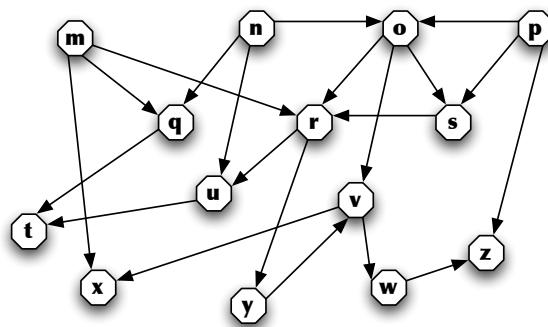


FIGURE 1 – $G(S, A)$ un graphe orienté sans circuit.

Question 3. Prouver sa correction et établir sa complexité.

Question 4. Montrer qu'un graphe orienté est sans circuit si et seulement si il admet un tri topologique.