TD Programmation Dynamique

Olivier Raynaud (raynaud@isima.fr)

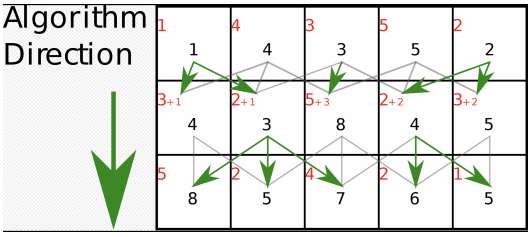

FIGURE 1 – Le seam carving, ou recadrage intelligent, est un algorithme de redimensionnement d'image développé par Shai Avidan et Ariel Shamir en 2007.

Exercice 1 (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 $dist(A, B)$ 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 $dist()$.*

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 2 (Plus longue sous-séquence commune)**.**

On considère deux séquences (textes formés de lettres) S et T. Leurs longueurs sont notées respectivement m et n. Ue 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 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 $S = algorithmique$ et $T = programmation$, 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 :*

1. *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 $T[1..n-1]$;*
2. *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 $T[]$;*
3. *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. *Modifier l'algorithme pour qu'il construise de faÁon effective la plus longue sous-séquence commune.*

Exercice 3 (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 4 (Sous-suite croissante maximale)**.**

Soit une séquence s de nombres $x_1, x_2, ..., x_n$ 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 $x_{i_1}, x_{i_2}, ..., x_{i_k}$ telle que $i_1 \leq i_2 \leq ... \leq i_k$. 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 s_i. Vérifier les relations de récurrence suivantes :*

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

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

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

Exercice 5 (Vente de batôns de bois)**.**

Vous avez un commerce de batôn de bois et le prix d'un batônn dépend de sa longueur. On donne un tableau de prix de vente $Prix[1..n]$ *où* $Prix[i]$ *représente le prix de vente d'un baton de longueur* i*. On supposera que les valeurs de* $Prix[]$ *sont strictement croissantes.*
Un long bâton de taille l *peut être coupé en plusieurs batôns de tailles inférieures. Vous voulez trouver le découpage qui maximise la somme des prix de vente des batôns obtenus.*

Par exemple, avec $Prix[] = [1, 3, 5, 7]$ *et* $l = 10$*, vous pouvez couper votre batôn en* 3 *morceaux dont deux de taille* 3 *et un de taille* 4*, pour un prix total de* $2 * 5 + 1 * 7 = 17$*.*

Question 1. *Concevoir un algorithme glouton pour le calcul approché du prix de vente maximum d'un batôn de taille* l *pour un tableau de prix* $T[1..n]$*. Donner une instance du problème pour laquelle l'algorithme ne donne pas la solution optimale. Donner la taille de codage des données d'entrée et évaluer le nombre d'opérations exécutées par l'algorithme. Donner la complexité.*

Question 2. *Soit* $P(i)$ *le prix de vente maximum que l'on peut atteindre avec un batôn de longueur* i*.*

$$P(i) = max(Prix[i], \max_{k \in [1..n]}(P([i-k) \ + \ Prix[k]))$$

Expliquer la formule de récurrence. Donner les cas d'arrêt de la récursion. Dessiner les premiers niveaux de l'arbre de appels récursif. Concevoir en LDA un algorithme récursif.

Question 3. *Proposer une table de programmation dynamique pour le calcul du prix de vente maximum. Remplir la table pour l'instance* $l = 10$ *et* $Prix[] = [1, 3, 5, 7]$*.*

Question 4. *Concevoir un algorithme de programmation dynamique pour le cacul du prix de vente maximum d'un batôn de taille* l *pour un tableau de prix* $T[1..n]$*. Evaluer la complexité.*

Question 5. *Concevoir à partir de l'algorithme de la question précédente un algorithme qui donne la liste des coupes à faire pour obtenir le prix de vente maximum d'un batôn de taille* l *pour un tableau de prix* $T[1..n]$*.*

Exercice 6. *Alignement de séquences*

Un problème récurrent en bioinformatique est l'alignement de séquences. Si l'on a deux mots u et v formés à partir d'un alphabet $\mathcal{A}$, (par exemple les nucléotides : $\mathcal{A} = G, T, A, C$), on appelle alignement de u et v un couple de mots (u', v') sur l'alphabet $\mathcal{A} \cup e$ ($o^{\smile}$ $e \notin \mathcal{A}$ est le caractère d'espacement).

u' et v' satisfont les propriétés suivantes :

- *u' et v' ont la mÍme longueur ($|u'| = |v'| = n$) ;*
- *si on supprime tous les e dans u' et v' on obtient respectivement u et v ;*
- *les lettres e ne sont pas à la mÍme position dans u' et v'.*

Par exemple un alignement de $u = CGATTAG$ et $v = GATCGA$ est :

$$u' = CGATTeAG \text{ et } v' = eGATCGAe$$

Afin de déterminer le meilleur alignement, il nous faut une métrique. Soit p une fonction de similarité entre un couple de lettres. La valeur de $p(a, b)$ est un réel d'autant plus grand que les lettres sont considérées comme similaires d'un point de vue biologique.On prendra donc une valeur négative lorsque les lettres sont différentes. On a donc $b \neq a$, $p(a, a) > 0 > p(a, b)$, la fonction de similarité est également symétrique : $p(a, b) = p(b, a)$. De plus, on se donne un réel $q < 0$ qui exprime la similarité d'une lettre avec un espacement, elle est indépendante de la lettre, on a donc $\forall a \in \mathcal{A}, p(a, e) = p(e, a) = q$.

Le score d'un alignement est simplement la somme des similarités entre les lettres de u' et v' :

$$\sum_{i=1}^{n} p(u'_i, v'_i) \tag{1}$$

Question 1. *On pose $p(a, b) = -1$ pour $a \neq b$, $p(a, a) = 5$ et $q = -2$. Quel est le score de l'alignement donné en exemple ? Donnez le meilleur alignement et le score correspondant pour les chaïnes GATTACA et ATACGTA.*

Question 2. *Déterminer les équations récursives propres au problème posé et l'algorithme récursif correspondant qui calcule le meilleur alignement possible.*

Question 3. *Donnez un algorithme polynomial permettant de trouver le meilleur alignement (celui ayant le score le plus élevé). Expliquez comment vous reconstruisez la solution.*

Question 4. *Quelle est la complexité en temps et en espace de votre algorithme ?*

Exercice 7 (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 $(a_1, a_2, ..., a_n)$.

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 Ítre 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 8 (Justification d'un texte en ligne)**.**

Soit une suite de mots $(m_0, m_1, ..., m_{n-1})$ *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 Ítre 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 attribut 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 cout de la mise en forme de la ligne* $m_i, m_{i+1}, ..., m_j$ *est* $|e' - e|$ $o^{\smile}$ e'*, la longueur réelle des espaces est :*

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

Cependant, si $j = n-1$ *(on atteint la dernière ligne), le cout 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 faÁon à avoir un cout minimal.

Question 1. *Donner les équations récursives de la fonction de calcul du cout de la mise en forme optimale d'une liste de mots* $m_1, m_2, ..., m_{n-1}$*.*

Question 2. *Concevoir un algorithme de programmation dynamique pour répondre à ce problème.*

Exercice 9 (Multiplication d'une chaïne de matrices)**.**

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

Question 2. *Donner un algorithme efficace 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 Chaïnes 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 10 (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 2. On supposera qu'à chaque étape nécessitant un choix arbitraire de sommets, ce choix se fait suivant l'ordre alphabétique des étiquettes.*

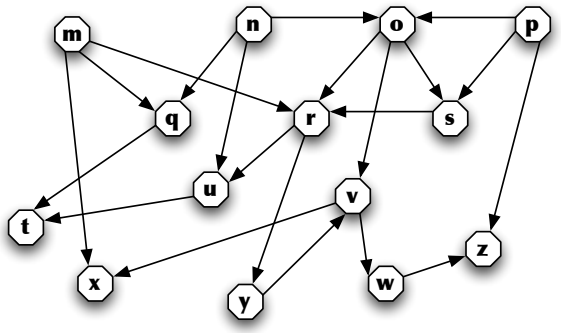

FIGURE 2 – $G(S, A)$ un graphe orienté sans circuit.

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.*