▶ **MP & MP* – Option Informatique**
Sixième TP Caml
Mardi 6 et 13 novembre 2007

Programmation dynamique

Tout comme la méthode « diviser pour régner », l'idée de la programmation dynamique consiste à résoudre des gros problèmes en combinant les solutions de sous-problèmes. Mais alors que la méthode « diviser pour régner » divise le problème en sous-problèmes indépendants (quitte à faire des calculs en double), la programmation dynamique va utiliser les dépendances entre les sous-problèmes pour aller plus vite.

Notions : programmation dynamique, arbres binaires de recherche optimaux, plus longue sous-séquence commune, chaînes de montage

1 Ordonnancement de chaînes de montage

Nous nous placerons d'abord dans le cadre d'un atelier de production d'une usine automobile qui dispose de deux chaînes complètes de montage. Chaque chaîne est constituée de plusieurs postes de montage, et sur chacune des chaînes, un châssis arrive au début, passe par les différents postes de montage et sort par l'autre extrémité de la chaîne.

Chaque chaîne de montage a n postes, numérotées de façon à peu près logique $j = 1, \ldots, n$. $S_{i,j}$ sera donc le j-ème poste de montage de la chaîne i (avec $i = 1$ ou 2), et quelle que soit la chaîne choisie, le j-ème poste fait toujours la même opération.

Tous les postes de montage n'ont pas été construits à la même époque, et deux postes similaires ne vont pas mettre le même temps pour faire la même opération. Ce temps de montage sera noté $a_{i,j}$, et on notera également e_i le temps d'arrivée du châssis qui entre sur la chaîne i, et x_i le temps de sortie pour la voiture complète qui sort de la chaîne i. Si jamais une voiture donnée doit être construite plus rapidement, on peut décider de la faire passer d'une chaîne à l'autre entre deux postes. La voiture passera toujours les n étapes de montage dans cet ordre. Naturellement, passer d'une chaîne à l'autre à un coût non négligeable, noté $t_{i,j}$, avec $i = 1, 2$ et $j = 1, 2, \ldots, n-1$ (il y a n postes, donc $n-1$ transitions d'un poste à l'autre. Si une voiture passe d'un poste au suivant sans changer de chaîne, on supposera que le temps pris est nul.

2 Plus longue sous-séquence commune

On considère une suite finie $x = (x_0, \ldots, x_{m-1})$ formée de m éléments d'un ensemble E. Une *sous-suite* de x de longueur k est une suite x' obtenue à partir de x en supprimant $m-k$ éléments, tout en conservant l'ordre. x' est donc une suite finie de la forme $x' = (x_{\sigma(0)}, \ldots, x_{\sigma(k-1)})$, avec $0 \leq \sigma(0) \leq \ldots \leq \sigma(k-1) < m$. On dit que z est une *sous-suite commune* de x et de y ssi z est une sous-suite de y et une sous-suite de x.
Nous représenterons en CamlLight les suites finies par des tableaux. Nous cherchons maintenant à trouver une plus longue sous-suite commune à deux suites $x = (x_0, \ldots, x_{m-1})$ et $y = (y_0, \ldots, y_{n-1}$.

▶ **Question 1** *Une méthode naïve pour résoudre le problème est de construire l'ensemble des sous-listes de x et de y, de calculer leur intersection et de déterminer le plus grand élément. Évaluez la complexité, en fonction de la taille des listes x et y, de cette méthode.*

Dans un premier temps, nous nous limitons au problème du calcul de la *longueur* d'une plus longue sous-suite commune de deux suites x et y. On note $l(i, j)$ la longueur de la plus longue sous-suite commune des suites $(x_0, \ldots, x_{i-1})$ et $(y_0, \ldots, y_{j-1})$. La longueur recherchée est donnée par $l(m, n)$, où m et n sont les longueurs respectives de x et y. De plus, l vérifie la relation de récurrence suivante :

$$l(i,j) = \max(l(i-1, j-1) + \delta(x_{i-1}, y_{j-1}), \\ l(i, j-1), l(i-1, j))$$

où $\delta(i,j) = 1$ si $x_i = y_j$, et 0 sinon.
Il est facile de vérifier que si on utiliser cette relation pour écrire une fonction récursive calculant la valeur de $l(i,j)$, on obtiendrait à nouveau une fonction dont le coût de calcul serait exponentiel en la taille de l'entrée : l'algorithme récursif rencontre en effet chaque sous-problème de nombreuses fois dans différentes branches de l'arbre récursif, sans réutiliser toutefois les calculs déjà effectués. La programmation dynamique contourne ce problème en stockant les résultats intermédiaires dans un tableau auxiliaire à deux dimensions la case (i,j) de cette matrice contient la longueur d'une plus longue sous-suite commune de $(x_0,\ldots,x_{i-1})$ et $(y_0,\ldots,_{i-1})$, c'est-à-dire $l(i,j)$.

▶ **Question 2** *Expliquez comment peut-on remplir de proche en proche le tableau auxiliaire.*

▶ **Question 3** *Déduisez-en une fonction qui permette de calculer la longueur de la plus longue sous-suite commune à deux suites. Votre fonction aura une complexité dominée par le produit des longueurs des deux listes passées en argument.*

```
subseq_length: 'a vect -> 'a vect -> int
```

On souhaite maintenant obtenir une des sous-suites communes de longueur maximale de deux suites. Pour cela, remarquons que la valeur de chaque case du tableau auxiliaire est obtenue à partir de celle de l'une de ses 3 voisines : celle située au-dessus, celle située à gauche ou celle située en diagonale en haut à gauche. En conservant (dans un second tableau, par exemple) pour chaque case une marque indiquant à partir de quelle voisine sa valeur a été calculée, il est possible à la fin de l'algorithme de reconstituer une plus longue sous-suite commune en parcourant le tableau obtenu en suivant les directions indiquées par les marques comme illustré sur la figure suivante.

	j	0	1	2	3
i		y_{j-1}	a	b	c
0	x_{i-1}	0	0	0	0
1	a	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1
2	a	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 1
3	c	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2
4	b	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2

Pour représenter les marques, vous pouvez définir en Caml le type suivant :

```
type mark = Vert | Horiz | Diag;;
```

▶ **Question 4** *Écrivez une fonction* subseq *qui retourne une des plus longues sous-suites commune de deux suites.*

```
subseq: 'a vect -> 'a vect -> 'a vect
```

3 Arbres binaires de recherche optimaux

▶ **MP & MP* – Option Informatique**
Sixième TP Caml
Mardi 6 et 13 novembre 2007

Programmation dynamique

Un corrigé

▶ **Question 1** On commence par supposer que tous les éléments de x sont distincts. Notons $n(k,m)$ le nombre de sous-suites de longueur k d'un mot de longueur n. On a $n(k,m) = n(k-1,m-1) + n(k,m-1)$ et on peut alors montrer $n(k,m) = \frac{k!}{(k-m)!m!}$. Bref, c'est gros, exponentiel et complètement infaisable en pratique.

▶ **Question 2** $l(i,j)$ fait appel à $l(i-1,j-1)$, $l(i,j-1)$ et à $l(i-1,j)$. Si on veut remplir $l(i,j)$, il faut donc avoir calculé ces trois valeurs. Il faut donc calculer les $l(i,j)$ par $i+j$ croissant.

▶ **Question 3**

```
let d = fun x y i j ->
    if x.(i-1) = y.(j-1) then 1 else 0
;;

let subseq_length = fun x y ->
    let m = vect_length x and n = vect_length y in
    let l = make_matrix (m+1) (n+1) 0 in
    for i = 1 to m do
        for j = 1 to n do
            l.(i).(j) <- max
                (l.(i-1).(j-1) + (d x y i j))
                (max l.(i).(j-1) l.(i-1).(j))
        done
    done;
    l.(m).(n)
;;

let t1 = [| 'a' , 'b' , 'c' |];;
let t2 = [| 'c' , 'b' , 'c' |];;
subseq_length t1 t2;;
```

▶ **Question 4** Notre fonction comporte deux parties. La première remplit incrémentalement les matrices s et t. Elle est analogue à la fonction subseq_length. La deuxième partie parcours le la matrice t de manière à retrouver comment construire une des plus longues sous-séquences communes.

```
let maxi = fun x y l i j ->
    if x.(i-1) = y.(j-1) then ((fst l.(i-1).(j-1)) + 1, Diag)
    else if (fst l.(i-1).(j)) > (fst l.(i).(j-1)) then
        ((fst l.(i-1).(j)), Vert)
    else ((fst l.(i).(j-1)), Horiz)
;;

let subseq = fun x y ->
(* On construit la meme matrice, mais en memorisant le chemin *)
    let m = vect_length x in
    let n = vect_length y in
    let l = make_matrix (m+1) (n+1) (0, Diag) in
    for i = 1 to m do
        for j = 1 to n do
            l.(i).(j) <- maxi x y l i j
        done
    done;
(* et on parcourt le chemin à l'envers *)
    let longueur = (fst l.(m).(n)) in
    let i = ref m in
    let j = ref n in
    let result = make_vect longueur x.(0) in
    let k = ref longueur in
    while (!k > 0) do
        if (snd l.(!i).(!j)) = Diag then (
            decr i; decr j; decr k; result.(!k) <- x.(!i);
        )
        else if (snd l.(!i).(!j)) = Horiz then decr j
        else decr i
    done;
    result
    ;;

let t1 = [| 'a' ; 'b' ; 'c' |];;
let t2 = [| 'a' ; 'd' ; 'b' ; 'c' |];;
subseq length t1 t2;;
subseq t1 t2;;
```