TD Info 5 : Programmation Dynamique

Michael Monerau

7 décembre 2010

1 Diviser pour régner (rappel)

Le méthode dite "*diviser pour régner*" est bien connue. Elle s'applique aux problèmes qui peuvent se découper en plusieurs sous-problèmes identiques mais plus petits (et indépendants). Le plus souvent, on coupe même le problème en deux parties de même taille. La complexité de l'algorithme $C(n)$ en fonction de n la taille de l'entrée suit donc une récurrence de la forme :

$$C(n) = 2C(n/2) + O(n)$$

où le $O(n)$ est le temps requis pour refusionner les résultats des deux sous-parties. Selon les algorithmes, cette quantité pourrait être plus grande. Ici nous restons dans ce cas-là.

On a alors $C(n) = O(n \ln n)$.

2 Rappel

Sujets, corrigés, dates des TDs sur www.michael.monerau.com.

Arrangez-vous avec vos colles pour venir à chaque TD.

3 Programmation dynamique

3.1 Principe

La programmation dynamique est une façon d'écrire un algorithme qui ressemble par beaucoup d'aspects à *diviser pour régner*. Le principe de base est toujours de diviser le problème à résoudre en plusieurs sous-problèmes. Mais cette fois-ci, leurs résolutions ne sont pas indépendantes, comme c'était le cas avant. C'est-à-dire que les sous-problèmes partagent des sous-sous-problèmes.

Dans ce cadre, *diviser pour régner* fait plus de travail que nécessaire en résolvant plusieurs fois les mêmes sous-sous-problèmes (penser à l'amélioration qu'apporte `option remember` en Maple).

3.2 Plus longue sous-suite commune

Commençons par quelques définitions.

Définition 1. *Soit $x = \langle x_1, \ldots, x_k \rangle$ une suite finie. On dit que $z = \langle z_1, \ldots, z_l \rangle$ est une sous-suite de x s'il existe des indices strictements croissants $i_1 < \ldots < i_l$ tels que $z_j = x_{i_j}$.*

Définition 2. *On dit que z est une sous-suite commune à deux suites x et y finies si z est une sous-suite à la fois de x et y.*

Par exemple, si $x = \langle A, C, E \rangle$ et $y = \langle C, D, E \rangle$, alors $z = \langle C, E \rangle$ est une sous-suite commune à x et y.

Le but du problème qu'on se pose est de trouver, étant données deux suites finies x et y la taille d'une plus longue sous-suite commune à x et à y.

Posons les notations, $X = \langle x_1, \ldots, x_m \rangle$ et $Y = \langle y_1, \ldots, y_n \rangle$. On suppose également qu'on a une plus longue sous-suite commune $Z = \langle z_1, \ldots, z_k \rangle$. Et on note $U_{[i]}$ pour les i premiers termes de la suite U.

On abrège Plus Longue Sous-Suite Commune en PLSSC.

On peut alors établir le :

Lemme 1. *Avec les notation ci-dessus :*
- *Si $x_m = y_n$, alors $z_k = x_m = y_n$ et $Z_{[k-1]}$ est une PLSSC de $X_{[m-1]}$ et $Y_{[n-1]}$.*
- *Si $x_m \neq y_n$, alors $z_k \neq x_m$ implique que Z est une PLSSC de $X_{[m-1]}$ et Y.*
- *Si $x_m \neq y_n$, alors $z_k \neq y_n$ implique que Z est une PLSSC de X et $Y_{[n-1]}$.*

Question 3.1. Expression récursive

On note $c_{i,j}$ la taille de la plus longue sous-suite commune à $X_{[i]}$ et $Y_{[j]}$. À partir du lemme précédent, trouver une expression de $c_{i,j}$ en fonction de $c_{i-1,j-1}$ et $c_{\ldots,\ldots}$.

Question 3.2. **Implémentation**
Implémenter ce calcul et donner la taille de la plus longue sous-suite commune de deux suites finies que vous choisirez.

Question 3.3. **Alors fais voir !**
Exhiber une plus longue sous-suite commune détectée dans l'algorithme précédent.

3.3 Plus court chemin : algorithme de Floyd

On suppose qu'on a un graphe $G = (V, E)$ pondéré (*ie.* on a une fonction de poids des arêtes $w : E \longrightarrow \mathbb{R}^+$).

Le problème est de trouver la longueur du plus court chemin entre chaque paire de sommets du graphe, la longueur d'un chemin étant la somme du poids de ses arêtes.

Plutôt que de calculer le plus court chemin indépendamment, on calcule tout en même temps. C'est adapté à la programmation dynamique car un plus court chemin est héréditaire : une partie d'un plus court chemin est un plus court chemin (entre d'autres sommets). Un calcul de plus court chemin entre plusieurs paires peut donc faire plusieurs fois appel à un plus court chemin particulier, qu'on n'a pas à recalculer si on l'a retenu. C'est l'algorithme de Floyd, il est du même type que l'algorithme de Warhsall du TD2 qui trouvait la clôture transitive d'un graphe.

Ici, on prend la représentation des graphes en matrice d'adjacence : $\forall i, j \in V,\ M(i, j) = w(i, j)$ (et qui est nul si $(i, j) \notin E$).

On définit $dist_k(i, j)$ comme la longueur du plus court chemin de i à j ne contenant que des sommets d'étiquette $\leq k$ (sauf peut-être i ou j), pour $k \geq 1$.

Question 3.4. $dist_1$ **?**
Que vaut $dist_1$?

Question 3.5. $dist_2$ **?**
Que vaut $dist_2$?

Question 3.6. $dist_k$ **?**
Définir $dist_k$ en fonction de $dist_{k-1}$ pour $k \geq 2$.

Question 3.7. $dist$ **?**
En quoi la suite de matrices $(dist_k)_{k\geq 0}$ nous donne-t-elle accès à la longueur du plus court chemin entre (i, j) que nous recherchons ?

Question 3.8. **Implémentation**
Implémenter l'algorithme de Floyd. Quelle est la complexité ?