TD/TP: Programmation dynamique

24 mars 2014

Les 4 exercices qui constituent le TP se résolvent efficacement avec de la programmation dynamique.

Exercice 1 *Écrire une fonction qui prend un entier n en entrée, et qui remplit un tableau de taille n tel qu'en position i du tableau, on trouve la décomposition en facteur premier de i.*

Votre fonction pourra renvoyer par exemple pour $n = 10$: $[0, [2], [3], [2, 2], [5], [2, 3], [7], [2, 2, 2], [3, 3], [2, 5]]$. Testez votre fonction pour $n = 10^6$.

Un *mot* sur un aphabet Σ est représenté par un tableau tel qu'en position i on trouve la i-ème lettre (une *lettre* est un élément de Σ) du mot. Par exemple, sur l'alphabet $\Sigma = \{A, C, G, T\}$, $w_1 = [C, A, A, T, G, C]$ et $w_2 = [A, G, A, T, T, C]$ sont deux mots. Le plus long sous-mot commun (longest common subword) de deux mots u et v est le plus long mot qui est un sous-mot de u *et* de v. Un sous-mot est le préfixe d'un suffixe du mot, ou de façon équivalente une sous-séquence consécutives de lettres du mot. La plus longue sous-séquence commune (longest common subsequence) de deux mots u et v est le plus long mot qui est une sous-séquence (pas nécessairement contiguë) de u *et* de v. Le plus long sous-mot commun de w_1 et w_2 est $[A, T]$. La plus longue sous-séquence communne de w_1 et w_2 est $[A, A, T, C]$.

Exercice 2 *Écrire une fonction qui retourne un plus long sous-mot commun à deux mots (tableaux) passés en arguments.*

Exercice 3 *Écrire une fonction qui retourne une plus longue sous-séquence communne à deux mots (tableaux) passés en arguments.*

Testez vos fonctions sur des exemples simples.

Une matrice des distances entre n villes sera représenté par un tableau de n tableaux de n cases, tel que $dist[i][j]$ et $dist[j][i]$ contiennent la distance de la ville i à la ville j. Dans le problème du voyageur de commerce, le but est de visiter chaque ville exactement une fois en minimisant la distance parcourue. On peut imposer une ville de départ, ou non. Ceci vous est laissé comme choix.

Exercice 4 *Écrire une fonction qui prend une matrice des distances et qui renvoie un plus court parcours sous la forme que vous voulez.*

On n'attend pas un algorithme polynomial car il n'y en a très probablement pas. L'intérêt de la programmation dynamique sur ce problème, comparé à l'algorithme naïf en $O(n!)$, est d'obtenir un algorithme en $O(n^2 2^n)$. Testez votre fonction avec une dizaine de villes.