Stanislas
T.D. 8

Programmation dynamique

MPSI
1 & 2
2020-2021

■ ■ ■

Exercice 1. (Ordonnancement de tâches pondérées) Le problème d'ordonnancement de n tâches pondérées (*weighted interval scheduling*), consiste à trouver un sous-ensemble de poids maximum de tâches qui s'exécutent successivement. Soit J un ensemble de tâches. À toute tâche $j \in J$ est associée un instant de début s_j, un instant final f_j et un poids w_j. On cherche un sous-ensemble optimal $\mathscr{O}$ de J tel que

$$\mathscr{O} = \underset{K \subset J \ ; \ \forall \ i,j \in K, f_i \leqslant s_j \ \text{ou} \ f_j \leqslant s_i}{\operatorname{argmax}} \sum_{i \in K} w_i.$$

1. On commence par supposer que tous les poids valent 1. On considère l'algorithme consistant, à chaque instant, à choisir la tâche admissible se terminant le plus tôt.

a) Montrer que cet algorithme est optimal.
Cet algorithme est appelé algorithme glouton.

b) On suppose qu'il existe 2 tâches. La première commence à l'instant 1, se termine à l'instant 3 et a un poids 1. La deuxième, commence à l'instant 2, se termine à l'instant 4 et a un poids 100. Montrer que l'algorithme précédent est sous-optimal.

Nous allons, dans le cas où les poids sont distincts, considérer un algorithme de programmation dynamique répondant au problème.

2. On suppose que les tâches sont ordonnées par ordre croissant d'instant de fin. Soit $i \in [\![1, n]\!]$. On note $p(i)$ le nombre de tâches se terminant avant que commence celle d'indice i et $opt(i)$ le poids maximal de tâches non simultanées dont les indices sont inférieurs à i. Montrer que

$$opt(i) = \max\left\{w_i + opt(p(i)), opt(i-1)\right\}.$$

3. Proposer un algorithme permettant de calculer $opt(n)$.

4. Modifier l'algorithme précédent pour renvoyer l'ordonnancement des tâches à effectuer.

5. Évaluer la complexité de cet algorithme.

Exercice 2. (Distance d'édition) Pour comparer des séquences d'ADN (successions de lettres A, C, T, G) on considère une distance (la *distance d'édition*) entre 2 séquences comme *le nombre d'insertions, de suppressions et de substitutions minimal nécessaire pour passer d'une séquence à l'autre.*
Par exemple, si $S_1 = AC$ et $S_2 = AGC$, les différents alignements (sans superposition de deux suppressions) possibles sont

$$\begin{pmatrix} AC- & A-C & -AC & AC-- & AC-- & A--C & A-C- & -A-C & \cdots \\ AGC & AGC & AGC & -AGC & A-GC & -AGC & AG-C & AGC- & \cdots \end{pmatrix}$$

1. Notons $f(n,m)$ le nombre d'alignements entre deux séquences $a = a_1 \cdots a_n$ et $b = b_1 \cdots b_m$.

a) Déterminer $f(0,0)$, $f(0,m)$ et $f(n,0)$.

b) Écrire une relation de récurrence entre $f(n,m)$, $f(n-1,m)$, $f(n-1,m-1)$ et $f(n,m-1)$.
On pourrait montrer que $f(n,n) \sim (1+\sqrt{2})^{2n+1} \cdot \sqrt{n}$.

Étant données 2 séquences $S = s_1 \dots s_m$ et $T = t_1 \dots t_n$, on définit $D(i,j)$ la distance d'édition entre le préfixe de taille i de S, i.e. $s_1 \dots s_i$ et le préfixe de taille j de T, i.e. $t_1 \dots t_j$.

2. Calculer $D(i,0)$ et $D(0,j)$.

3. Lorsque i et j sont non nuls, exprimer $D(i,j)$ en fonction de $D(i-1,j)$, $D(i,j-1)$ et $D(i-1,j-1)$.

4. Proposer un algorithme permettant de répondre à la question.

5. Évaluer la complexité de cet algorithme.

6. Déterminer la distance entre les chaînes `NICHE` et `CHIENS` ainsi qu'une suite d'opérations qui permette de passer de l'une à l'autre.