Synthèse du cours 6 : programmation dynamique (3)

18 octobre 2022

François Laroussinie

NB : Ces synthèses ont pour but de compléter les notes prises en cours. Elles ne les remplacent pas ! En particulier, la plupart des preuves n'y figurent pas. Rappel : il faut programmer les algorithmes vus en cours.

1 Le problème du sac à dos

Input : Un poids maximal $W \in \mathbb{N}$, un ensemble de n objets ayant chacun un poids w_i et une valeur v_i pour $i = 1, \ldots, n$. On suppose $w_i > 0$ et $v_i > 0$.
Output : La **valeur** maximale pouvant être stockée dans le sac sans dépasser sa capacité W.

On peut distinguer deux variantes selon que l'on autorise à mettre plusieurs fois un même élément dans le sac (on a, dans ce cas, plutôt une liste de *types* d'objets) ou pas. La variante la plus classique est celle sans répétition.

Exemple avec $W = 10$ et

w	5	5	6
v	20	2	3

La valeur maximale pour un sac de capacité 10 est 22 pour la version sans répétition (avec les objets 1 et 2) et 40 si on autorise des répétitions (avec deux objets 1).

1.1 Sans répétition

Chacun des n objets peut être mis au plus une fois dans le sac.

On va calculer le tableau $K[i, w]$, avec $0 \leq i \leq n$ et $0 \leq w \leq W$, défini comme étant la **valeur** maximale pouvant être stockée dans un sac de **capacité** w avec les objets $1, 2, 3, \ldots, i$. Le résultat du problème est alors la valeur $K[n, W]$: la valeur maximale pour un sac de capacité W en prenant des objets parmi tous les objets possibles.

On a clairement $K[0, w] = 0$ car si aucun objet n'est disponible, le sac sera vide, donc de valeur 0. De même on a $K[j, 0] = 0$ car tout objet à un poids strictement positif et aucun de peut être mis dans un sac de capacité nulle. Il reste à établir le cas général :

$$K[j, w] \stackrel{\text{def}}{=} \max\Big(K[j-1, w], v_j + K[j-1, w - w_j]\Big)$$

D'où l'algo :

Procédure `SaDssRepet` (O, W)
begin
  `//`O `correspond à la liste des` n `objets de poids` w_i `et de valeur` v_i
  `K` : tableau de booléens de taille $(n+1) \times (W+1)$
  pour $w = 1 \dots W$ **faire** $\texttt{K}[0, w] := 0$
  pour $i = 0 \dots n$ **faire** $\texttt{K}[i, 0] := 0$
  pour $i = 1 \dots n$ **faire**
    pour $w = 1 \dots W$ **faire**
      si $w_i \leq w$ **alors** $\texttt{K}[i, w] := \max(\texttt{K}[i-1, w], v_i + \texttt{K}[i-1, w-w_i])$
      else $\texttt{K}[i, w] := \texttt{K}[i-1, w]$
  return $\texttt{K}[n, W]$

Que l'on peut améliorer en utilisant une seule ligne du tableau (en veillant à énumérer correctement les indices !) :

Procédure `SaDssRepet2` (O, W)
begin
  `//`O `correspond à la liste des` n `objets de poids` w_i `et de valeur` v_i
  `K` : tableau de booléens de taille $W+1$
  pour $w = 0 \dots W$ **faire** $\texttt{K}[w] := 0$
  pour $i = 1 \dots n$ **faire**
    pour $w = W \dots w_i$ **faire**
      $\texttt{K}[w] := \max(\texttt{K}[w], v_i + \texttt{K}[w-w_i])$
  return $\texttt{K}[n, W]$

1.2 Avec répétition

Là encore, on peut calculer des coefficients $[i, w]$. On a toujours $K[0, w] = 0$ et $K[j, 0] = 0$ et le cas général s'énonce ainsi :

$$K[j, w] \stackrel{\text{def}}{=} \max\Big(K[j-1, w], v_j + K[j, w-w_j]\Big)$$

NB : c'est de prendre $K[j, w-w_j]$ (et non $K[j-1, w-w_j]$) qui permet de mettre plusieurs occurrences du même objet j !

D'où l'algo :

Procédure `SaDavecRepet` (O, W)
begin
  `//`O `correspond à la liste des` n `objets de poids` w_i `et de valeur` v_i
  `K` : tableau de booléens de taille $(n+1) \times (W+1)$
  pour $w = 1 \ldots W$ **faire** $\texttt{K}[0, w] := 0$
  pour $i = 0 \ldots n$ **faire** $\texttt{K}[i, 0] := 0$
  pour $i = 1 \ldots n$ **faire**
    pour $w = 1 \ldots W$ **faire**
      si $w_i \leq w$ **alors** $\texttt{K}[i, w] := \max(\texttt{K}[i-1, w], v_i + \texttt{K}[i, w - w_i])$
      else $\texttt{K}[i, w] := \texttt{K}[i-1, w]$
  return $\texttt{K}[n, W]$

Et son amélioration (en veillant à énumérer correctement les indices !) :

Procédure `SaDavecRepet2` (O, W)
begin
  `//`O `correspond à la liste des` n `objets de poids` w_i `et de valeur` v_i
  `K` : tableau de booléens de taille $W + 1$
  pour $w = 0 \ldots W$ **faire** $\texttt{K}[w] := 0$
  pour $i = 1 \ldots n$ **faire**
    pour $w = w_i \ldots W$ **faire**
      $\texttt{K}[w] := \max(\texttt{K}[w], v_i + \texttt{K}[w - w_i])$
  return $\texttt{K}[n, W]$

2 Un jeu

On considère un jeu de cartes à deux joueurs. Au début du jeu, on dispose une suite de n cartes $c_1, \ldots, c_n$ sur une table (face visible). Chaque carte c_i a une valeur $v_i \in \mathbf{R}$. A chaque tour, le joueur 1 choisit une des deux cartes situées à une extrémité (gauche ou droite) de la rangée. Le joueur 2 fait ensuite de même. Le jeu s'arrête lorsque toutes les cartes ont été prises par les deux joueurs. Le gagnant est celui dont la somme des valeurs de ses cartes est la plus grande.

A chaque tour, la position du jeu est définie par une paire (appelée configuration) (i, j) où $1 \leq i \leq j \leq n$: la carte la plus à gauche est c_i et celle la plus à droite est c_j. La configuration initiale est $(1, n)$. L'objectif est de calculer la meilleure stratégie possible pour le joueur 1.

On peut construire une table $\mathsf{TV}[-, -]$ de dimension $n \times n$ telle que pour toute configuration (i, j) $\mathsf{TV}[i, j]$ contient la valeur optimale que peut **assurer** le joueur 1 lorsqu'il a le trait (NB : « assurer » signifie ici que le joueur 2 joue le mieux possible) :

$$\mathsf{TV}[i,j] \;=\; \begin{cases} 0 & \text{si } i > j \\ v_i & \text{si j=i} \\ \max(v_i, v_j) & \text{si j=i+1} \\ \max(v_i + \min(\mathsf{TV}[i+2, j], \mathsf{TV}[i+1, j-1]), & \\ \qquad v_j + \min(\mathsf{TV}[i, j-2], \mathsf{TV}[i+1, j-1])) & \text{sinon} \end{cases}$$

Il reste à énumérer correctement des paires i, j afin de remplir le tableau $\mathsf{TV}[i, j] \ldots$ A faire en exercice !

Une fois calculé $\mathsf{TV}[1, n]$, on peut répondre à la question « est-ce que le joueur 1 dispose d'une stratégie gagnante ou non ? » (on compare $\mathsf{TV}[1, n]$ et $(\sum_{i=1..n} c_i) - \mathsf{TV}[1, n]$).

Rappel. On a vu en cours pourquoi le joueur 1 avait une « stratégie non perdante » (= il peut ne jamais perdre si il joue bien) lorsque le nombre de cartes est pair. Mais ce n'est pas vrai lorsque le nombre de cartes est impair.