TD 3 (avec corrigé)

Programmation dynamique

Exercice 1. Le jeu avec deux piles de pièces.

On considère un jeu où on dispose de deux piles de pièces P et Q contenant respectivement p et q pièces (p, $q > 1$). Le but est d'atteindre une des deux situations ($p = 0$, $q = 1$) ou ($p = 1$, $q = 0$), sachant que l'on passe d'un état des piles au suivant de la façon suivante : on enlève deux pièces à P (resp. Q) et on en ajoute une à Q (resp. P).

Exemple. ($p = 4$, $q = 6$) $\rightarrow$ ($p = 2$, $q = 7$) ou ($p = 5$, $q = 4$).

On veut calculer par programmation dynamique le nombre de façons différentes $N[p, q]$ permettant d'atteindre l'un des des deux états terminaux ($p = 0$, $q = 1$) ou ($p = 1$, $q = 0$) à partir d'une situation où le tas P a p pièces et le tas Q en a q.

1. Donner la formule de récurrence complète exprimant $N[p, q]$.

Réponse.

N[0, 1] = N[1, 0] = 1; N[1, 1] = 0;
$\forall$ q $\geq$ 2 : N[0, q] = N[1, q - 2] et N[1, q] = N[2, q - 2];
$\forall$ p $\geq$ 2 : N[p, 0] = N[p - 2, 1] et N[p, 1] = N[p - 2, 2];
$\forall$ p, q $\geq$ 2 : N[p, q] = N[p - 2, q + 1] + N[p + 1, q - 2].

2. Préciser l'évolution du calcul permettant d'obtenir $N[p, q]$ (on ne demande pas l'algorithme).

On constate que toute valeur de somme k fait appel à une ou deux valeurs de somme (k – 1); il faut donc faire progresser le calcul par diagonale; on initialise les 3 cases N[0, 1], N[1, 0] et N[1, 1], puis pour chaque diagonale de somme k on initialise ses 4 éléments extrêmes (N[0, k], N[1, k – 1], N[k, 0] et N[k – 1, 1]) grâce à la valeur de la diagonale précédente dont elle a besoin (par exemple, N[0, k] à l'aide de N[1, k – 2]) qui a été calculé auparavant puisque dans la diagonale précédente, enfin on calcule les autres éléments N[p, k – p] pour p variant de 2 à (k – 2) grâce à la formule générale.

Remarque : le tableau obtenu est évidemment symétrique.

3. À l'aide d'un tableau calculer $N[4, 2]$ **en ne remplissant que les cases utiles**.

Réponse.

	0	1	2	3	4
0		1			1
1			1		
2	1			2	
3		1			
4			3		
5	1				

4. Que vaut $N[3, 3]$?

Réponse.

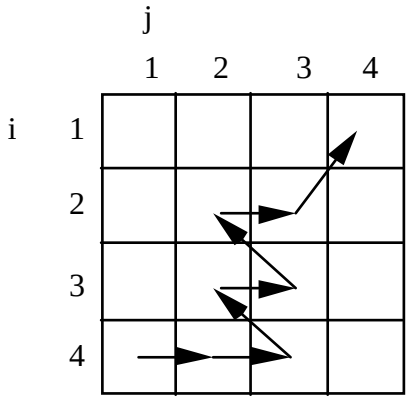
N[3, 3] = N[1, 4] + N[4, 1] = N[2, 2] + N[2, 2] = N[0, 3] + N[3, 0] + N[0, 3] + N[3, 0] =
N[1, 1] + N[1, 1] + N[1, 1] + N[1, 1] = 0.

Exercice 2. Chemin de coût minimal sur un échiquier avec 3 déplacements type et des pénalités.

On considère un échiquier $n \times n$ ($n > 1$) et on s'intéresse au calcul du **meilleur chemin** allant de la case $(n, 1)$ à la case $(1, n)$ sachant que :

i. **chaque case a un coût** (entier positif, négatif ou nul) et que le coût d'un chemin est la somme des valeurs des cases qu'il emprunte,
ii. un meilleur chemin est un **chemin de valeur minimale** et
iii. de la case (i, j) (avec $1 \leq i \leq n$ et $1 \leq j \leq n$) on peut aller en :
   - $(i, j + 1)$ si $1 \leq i \leq n$ et $1 \leq j + 1 \leq n$
   - $(i - 1, j - 1)$ si $1 \leq i - 1 \leq n$ et $1 \leq j - 1 \leq n$
   - $(i - 1, j + 1)$ si $1 \leq i - 1 \leq n$ et $1 \leq j + 1 \leq n$.

Exemple.



on a ici le chemin :

(4, 1), (4, 2), (4, 3), (3, 2),
(3, 3), (2, 2), (2, 3), (1, 4)

On veut calculer par programmation dynamique *MC*[*n*, 1], le coût associé au (à un) meilleur chemin allant de la case (*n*, 1) à la case (1, *n*).

1. Connaissant les valeurs associées aux cases de l'échiquier données dans le tableau *C*[*i*, *j*], établir **la formule de récurrence** pour calculer *MC*[*i*, *j*], le coût associé au (à un) meilleur chemin allant de la case (*i*, *j*) à la case (1, *n*) avec $1 \leq i, j \leq n$. Par hypothèse, on supposera que pour tout échiquier : *C*[1, *n*] = C[*n*, 1] = 0.

Réponse.

MC [1, n] = 0
MC [1, i] = C[1, i] + MC [1, i + 1] pour tout i ∈ [1, n - 1]
MC [i, n] = C [i, n] + MC [i - 1, n - 1] pour tout i > 1
MC [i, 1] = C [i, 1] + min(MC [i, 2], MC [i - 1, 2] pour tout i > 1
MC [i, j] = C [i, j] + min(MC [i, j + 1], MC [i - 1, j - 1], MC [i - 1, j + 1]) pour i > 1 et 1 < j < n.

2. **Expliciter l'évolution du calcul** effectué par l'algorithme mettant en œuvre ces formules en précisant comment on pourra mémoriser le meilleur chemin et le reconstituer.

Réponse.

On peut initialiser la ligne supérieure à grâce aux deux premières équations. Ensuite, pour toute ligne, l'élément de la colonne n peut être calculé à partir de l'élément de la colonne (n - 1) de la ligne précédente (3ème équation), tout autre élément à l'exception du premier peut être calculé à partir de son voisin supérieur gauche, son voisin supérieur droit et son voisin de droite (5ème équation), le premier élément de la ligne peut être calculé à partir de son voisin supérieur droit et de son voisin de droite (4ème équation).

En résumé, on fait un calcul **par ligne de haut en bas** et dans une ligne **de droite à gauche**.

On pourra mémoriser le chemin en gérant un second tableau DEP qui associe à chaque case la valeur du choix à faire pour le déplacement suivant (hg, hd ou d). La reconstitution du chemin se fait de façon naturelle par une procédure séquentielle partant de DEP[n, 1] en progressant vers le voisin correspondant à la valeur trouvée (hg, hd ou d).

3. Donner les valeurs de *MC* dans le cas du tableau *C* ci-dessous :

	1	2	3	4
1	2	-4	1	0
2	-6	2	-1	3
3	5	-2	-3	3
4	0	10	2	7

	1	2	3	4
1	-1	-3	1	0
2	-9	-2	-4	4
3	-6	-11	-5	-1
4	-11	1	-9	2

Exercice 3. Franchissement d'un escalier avec *n* sauts d'amplitude a_1 à a_p.

On considère un escalier de *m* marches que l'on peut gravir par des sauts de a_1, ..., a_p marches. On veut calculer le nombre *N*(*s*, *m*) de façons différentes de gravir *m* marches en *s* sauts.

Exemple. Soit $m = 12$ et $a_1 = 2$, $a_2 = 3$, $a_3 = 5$. On peut franchir les 12 marches par exemple par les enchaînements de sauts suivants :

2, 2, 2, 2, 2, 2 2, 3, 2, 3, 2
3, 3, 3, 3 2, 2, 3, 5
3, 2, 5, 2

1. Donner la formule de récurrence complète de calcul de *N*(*s*, *m*).

Réponse.

$N(1, m) =$ 1 s'il existe i tel que $a_i = m$
0 sinon

$N(s, m) = \Sigma_{i = 1..p \mid (m - a_i) > 0} N(s - 1, m - a_i)$ pour tout $s > 1$

$N(s, m) = 0$ si $m < \min_i a_i$

2. Préciser l'évolution du calcul de *N*(*s*, *m*).

Réponse.

On voit que le calcul de N(s, m) requiert uniquement celui des éléments associés à (s – 1). On peut donc d'abord remplir la première ligne (1) qui vaut 1 pour toute valeur a_i et 0 sinon, puis remplir par ligne (2 à s) dans un ordre quelconque (gauche-droite en pratique par exemple).

3. Calculer les valeurs de *N*(*s*, *m*) pour $m \leq 12$ et $a_1 = 2$, $a_2 = 3$, $a_3 = 5$.

1	2	3	4	5	6	7	8	9	10	11	12
0	1	1	0	1	0	0	0	0	0	0	0
0	0	0	1	2	1	2	2	0	1	0	0
0	0	0	0	0	1	3	3	4	6	3	3
0	0	0	0	0	0	0	1	4	6	8	13
0	0	0	0	0	0	0	0	0	1	5	10
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0

4. On remarque dans l'exemple précédent que les cases de certaines zones valent 0. Expliquer et proposer une amélioration au niveau de l'algorithme.

Réponse.

De façon générale, avec n sauts on franchit au moins $n * a_1$ et au plus $n * a_p$ marches (si on suppose les a_i ordonnés, ou au moins $\min_i a_i$ et au plus $\max_i a_i$). Ceci se traduit par le fait qu'à la ligne s, on n'a qu'à remplir les cases allant de $s * a_1$ à $s * a_p$.

Exercice 4. Nombres de Stirling.

On considère $S(i, N)$ le nombre de partitions à i éléments d'un ensemble ayant N éléments. Ainsi, avec l'ensemble $\{a, b, c, d, e\}$, $[\{a, d\}, \{b, c, e\}]$ est une partition à deux éléments et $[\{a\}, \{b, c, d\}, \{e\}]$ une partition à trois éléments.

1. Donner la formule de récurrence permettant de calculer $S(i, j)$ avec $i > 0$, $j > 0$, $i \leq j$.

Réponse.

S(N, N) = 1 pour tout N > 0
S(1, N) = 1 pour tout N > 0
S(i, N) = i * S(i, N – 1) + S(i – 1, N – 1) pour i < N, i > 0 et N > 1.

2. Donner le principe d'une méthode de calcul tabulaire (programmation dynamique) de $S(i, j)$ ($1 \leq i \leq M$, $1 \leq j \leq N$) en précisant la nature du tableau utilisé et l'évolution du calcul.

Réponse.

On constate que l'on a la contrainte i ≤ N, donc on utilisera une matrice triangulaire supérieure pour calculer et stocker les valeurs de S. Le calcul de S(i, N) nécessite la connaissance de S(i, N - 1) le voisin de gauche et S(i - 1, N - 1) le voisin diagonal haut gauche. On peut donc procéder de la façon suivante :

calcul par ligne (la première étant initialisée à 1)
dans une ligne, calcul de gauche à droite à partir de l'élément diagonal qui vaut 1.

3. Ecrire la procédure correspondante en langage de description.

procédure stirling (ent M, N);
var ent j, k;

```
début
   pour j de 1 à N faire S[1, j] ← 1 fait;
   pour j de 2 à M faire
      S[j, j] ← 1;
      pour k de (j+1) à N faire
         S[j, k] ← j * S[j, k-1] + S[j-1, k-1]
      fait
   fait
fin;
```

4. Donner les complexités temporelle et spatiale du calcul de $S(i, N)$.

Complexité spatiale : tableau S[1:M, 1:N] donc θ(M*N),
Complexité temporelle liée aux deux boucles imbriquées soit θ(M*N).

5. Effectuer le calcul de $S(i, 6)$ pour $i = 1$ à 6.

Ce calcul requiert le remplissage complet de la matrice triangulaire supérieure S, soit les valeurs :

1	1	1	1	1	1
	1	3	7	15	31
		1	6	25	90
			1	10	65
				1	15
					1

Exercice 5. Coût d'un arbre binaire de recherche.

On se donne n éléments $x_1, x_2, ..., x_n$ tels que $x_1 < x_2 < ... < x_n$ suivant l'ordre < et l'on désire organiser ces éléments en arbre binaire de recherche (ABR). Supposons que p_i soit la probabilité pour que la recherche porte sur x_i. Alors, pour tout ABR, le coût moyen d'une recherche est

$$\Sigma_{i = 1..n}\, p_i\, (d_i + 1)$$

où d_i est la profondeur du noeud contenant x_i. Etant données les p_i et en supposant que les x_i ne sont pas susceptibles d'être modifiés, il est possible de construire un ABR minimisant le coût des recherches.

1. Exprimer le coût d'un arbre en fonction du coût de sa racine et de celui de ses sous-arbres.

Soit rac(A) la racine de l'arbre A, soit sag(A) (resp. sad(A)) le sous-arbre gauche (resp. droit) de A.

coût(A) = coût(sag(A)) + coût(sad(A)) + somme des poids (p_i) de tous les noeuds de A.

2. Trouver un algorithme dynamique pour déterminer l'arbre de coût minimal.

On calcule, pour tous les couples (i, t) le coût optimal d'un arbre ne contenant que x_i, x_{i+1}, ..., x_{i+t-1}, c'est-à-dire les t éléments à partir de x_i. Ces résultats sont rangés dans un tableau C. La solution se

trouvera donc dans C[1, n]. Les probabilités se trouvent dans un tableau P. On supposera disponible un tableau S tel que S[i, t] contient la somme des poids de p_i à p_{i+t-1}.

Initialisation : C[i, 1] = P[i] $\forall i \in 1..n$

Cas général : calcul de C[i, t] avec t > 1

1er choix possible : la racine est x_i

$coût_1$ = C[i+1, t-1] + S[i, t]

2ème choix possible : la racine est x_{i+t-1}

$coût_2$ = C[i, t-1] + S[i, t]

3ème choix possible : la racine $\in [x_{i+1} .. x_{i+t-2}]$

$coût_3 = \min_{k = 1..t-2} (C[i, k] + S[i, t] + C[i+k+1, t - k - 1])$

D'où : C[i, t] = min($coût_1$, $coût_2$, $coût_3$)

3. Quelle est la complexité de cet algorithme?

Calcul de S : (n – 1) + (n – 2) + ... + 1 = n(n – 1)/2 donc en $\theta(n^2)$

Calcul de C : nécessite trois boucles imbriquées, donc en $\theta(n^3)$

Complexité globale en $\theta(n^3)$.

Exercice 6. L'allocation optimale de stations-service sur une autoroute.

On obtient la concession de stations-service **dans un seul sens** le long d'une autoroute en construction. Le gestionnaire de l'autoroute indique de possibles emplacements d'implantation des stations-service (repérés par un entier i) caractérisés par leur kilométrage par rapport à l'entrée de l'autoroute pour le sens considéré. On connaît l'estimation du gain annuel (en millions d'euros) d'une station-service à un emplacement donné. Cependant, il existe une règle d'implantation qui impose une distance minimale entre deux stations, c'est-à-dire qu'on ne peut implanter une station à l'emplacement i que si la distance avec celle **qui la précède** (i-1) est au moins égale à la valeur spécifiée.

Exemple.

i	Position (km)	Gain annuel	Distance à préserver
1	40	5	40
2	90	7	70
3	130	1	30
4	200	5	120

Ici, si une station-service est implantée à l'emplacement 4 alors les emplacements 2 et 3 sont interdits puisqu'on doit respecter une distance de 120 km par rapport à la précédente (qui ne peut donc être au-delà du km 80, ce que ne vérifient pas les emplacements 2 et 3). On ne peut donc dans cette hypothèse qu'équiper l'emplacement 1. Par contre, l'implantation dans l'emplacement 3 n'empêche pas une implantation dans l'emplacement 2. Les seules combinaisons d'implantations possibles sont : (1), (2), (3), (4), (1, 3), (2, 3) et (1, 4).

On s'intéresse à la détermination de la **combinaison optimale d'implantations** de stations-service pour un ensemble de N emplacements définis avec leurs données associées (position, estimation de gain annuel et distance à préserver).

1. On cherche à calculer gain-opt(i) la valeur du gain optimal d'implantation de stations-service des emplacements 1 à i inclus et on suppose disponibles les tableaux pos[1 : N], gain-an[1 : N] et dist-a-pres[1 : N]. En remarquant que les seuls choix sont l'implantation ou (exclusif) la non implantation d'une station à l'emplacement i, donner la récurrence complète définissant le calcul de gain-opt(i).

Réponse.

D'après la remarque faite, il y a deux possibilités lors de l'intégration de l'emplacement i (calcul de gain-opt(i)) :

a) on envisage d'y implanter une station-service et on doit respecter la règle de distance à préserver par rapport à la précédente ; on peut d'ailleurs associer à chaque station le numéro de l'emplacement associé à la station la plus proche implantable (nesppi(i)) donné dans l'exemple par : nesppi(1) = 0, nesppi(2) = 0, nesppi(3) = 2, nesppi(4) = 1 ; dans cette hypothèseon a donc : gain-opt(i) = gain-opt(nesppi(i)) + gain-an(i)

b) on ne met pas de station-service dans l'emplacement i et alors le gain-optimal est celui qu'on avait précédemment (pour l'emplacement i-1).

On en déduit la récurrence :

gain-opt(0) = 0
gain-opt(i) = max(gain-opt(nesppi(i)) + gain-an(i), gain-opt(i-1)) pour i > 0 et i ≤ N.

2. En déduire la structure tabulaire associée à la procédure de programmation dynamique, la localisation de la valeur recherchée et la progression du calcul pour remplir cette structure.

Réponse.

De la forme de la récurrence se dégage un vecteur G[0 : N] dans lequel la solution recherchée se trouve en position N (G[N]). Puisque le calcul au rang i fait appel à deux valeurs d'indice inférieur, un remplissage de gauche à droite (selon les valeurs croissantes de l'indice) convient.

3. Application. Soit le tableau de données suivant :

i	Position (km)	Gain annuel	Distance à préserver
1	50	4	0
2	100	3	60
3	140	6	30
4	200	2	110
5	250	4	70
6	320	5	150

Remplir la structure tabulaire associée à la résolution de cette situation.

Réponse.

On calcule tout d'abord les valeurs nesppi associées à ces données, soit : nesppi(1) = 0, nesppi(2) = 0, nesppi(3) = 2, nesppi(4) = 1, nesppi(5) = 3, nesppi(6) = 3. On remplit ensuite le tableau G, ce qui donne :

i	0	1	2	3	4	5	6
G	0	4	4	10	10	14	15

car : G[1] = max(0, 4 + 0) = 4, G[2] = max(4, 3 + 0) = 4, G[3] = max(4, 6 + 4) = 10,
G[4] = max(10, 2 + 4) = 10, G[5] = max(10, 4 + 10) = 14, G[6] = max(14, 5 + 10) = 15.

4. Proposer un mécanisme pour reconstituer les emplacements où sont implantées des stations-service pour la solution optimale trouvée. L'appliquer à l'exemple.

Une solution simple consiste à mémoriser dans un tableau "parallèle" IMP le choix effectué pour chaque emplacement en fonction de l'option qui a conduit à la valeur choisie. Ainsi, si le max est obtenu avec le choix de ne pas implanter de station, i.e., gain-opt(i) = gain-opt(i-1) IMP[i] ← -1 et dans l'autre cas on conserve le numéro d'emplacement de la station la plus proche implantable, soit IMP[i] ← nesppi(i). Il suffit à la fin de la procédure ayant rempli G et IMP de lire IMP de droite à gauche en partant de IMP[N] qui repère le choix pour le dernier emplacement dans la solution optimale, puis soit de passer au prédécesseur si la valeur est -1, soit de passer de visiter

l’emplacement donné afin de voir ce qui a été décidé pour celui-là et ainsi de suite jusqu’à atteindre l’emplacement 0. Appliqué à l’exemple, on obtient :

i	0	1	2	3	4	5	6
G	0	4	4	10	10	14	15
IMP	/	0	-1	2	-1	3	3

La solution optimale consiste à mettre une station-service dans les emplacements 6 (valeur 3), puis 3 (valeur 2), puis 1 car IMP[2] = -1 et on passe à l’emplacement précédent (1) qui est implanté (valeur 0) et arrêt. On peut vérifier qu’alors le gain obtenu est gain-an[1] + gain-an[3] + gain-an[6] = 4 + 6 + 5, valeur trouvée pour G[6].