TD programmation dynamique

1 Carrés de 1 dans une matrice binaire

Etant donné une matrice $M \in \mathcal{M}_{n,p}(\mathbb{R})$ à coefficients égaux à 0 ou 1, on cherche à déterminer le plus grand carré composé uniquement de 1 dans cette matrice.

1. Pour $i \in [\![1\,,n]\!]$ et $j \in [\![1\,,p]\!]$, on note $t_{i,j}$ la taille du plus grand carré de 1 inclus dans M dont le sommet en bas à droite est à la position (i, j). Déterminer selon la valeur de $m_{i,j}$ l'expression de $t_{i,j}$ en fonction de $t_{i-1,j}$, $t_{i,j-1}$ et $t_{i-1,j-1}$.

2. Ecrire une fonction construisant la matrice $T = (t_{i,j}) \in \mathcal{M}_{n,p}(\mathbb{R})$.

3. En déduire une fonction donnant la position et la taille du plus grand carré recherché.

4. Estimer la complexité de cet algorithme.

5. Ecrire un algorithme tirant une matrice aléatoire de format $n \times n$ et calculant la taille du plus grand carré rempli de 1 puis calculer l'espérance de cette taille quand on effectue un nombre important de tirages. Conjecturer en fonction de n cette espérance.

2 Chemin de poids minimum dans un tableau

Soit M un tableau de n lignes et p colonnes de nombres entiers. On se donne la règle de déplacement suivante : si on se trouve dans la case de coordonnées (i, j), on peut se rendre soit dans la case de droite $(i + 1, j)$, soit dans la case du dessous $(i, j + 1)$, à condition de ne pas sortir du tableau.
Un *chemin* est une suite de couples d'indices, coordonnées de cases du tableau, telle que deux couples consécutifs de la suite vérifient la règle précédente.
Le *poids* d'un chemin est égal à la somme des nombres contenus dans les cases parcourues.
On se donne comme point de départ la case D située en haut à gauche (d'indice (1,1)) et comme case d'arrivée A celle située en bas à droite (d'indice (n, p)).

1. Indiquer le nombre de chemins possibles menant de D à A.

   Dans les explications qui suivent, on suppose que les lignes et colonnes du tableau M sont indexées à partir de 1. On cherche maintenant à déterminer le chemin de poids minimum parmi tous les chemins qui mènent de D à A. Enumérer tous les chemins possibles peut être très coûteux suivant les valeurs de n et p.

   La programmation dynamique suggère de procéder différemment en s'appuyant sur le principe suivant : *Toute politique optimale est composée de sous-politiques optimales.*
   Il est évident que lors de la dernière étape le poids du chemin optimal augmentera de la valeur de l'entier $M[n, p]$ qui se trouve dans la case d'arrivée.

   A l'avant dernière étape, le chemin optimal passe obligatoirement par l'une des deux cases $(n, p-1)$ ou $(n-1, p)$ contigües à la case d'arrivée, le poids de ce chemin optimal augmentera alors de la valeur atteinte plus l'entier qui se trouve en A. Notons $c_1 = M[n, p - 1] + M[n, p]$ l'augmentation dans le cas où le chemin optimal passe par la case $(n, p - 1)$ et $c_2 = M[n - 1, p] + M[n, p]$ l'augmentation dans le cas où le chemin optimal passe par la case $(n - 1, p)$.

   A l'antépénultième étape, le chemin optimal passe forcément par une des trois cases de l'antépénultième diagonale : $(n, p-2), (n-1, p-1), (n-2, p)$. Pour les cases de cette diagonale qui sont sur le bord du tableau, on n'a pas le choix, le chemin optimal augmente dans un cas de $M[n, p - 2] + c_1$, dans l'autre cas de $M[n - 2, p] + c_2$.

   En fait, lorsqu'on atteint le bord du tableau (dernière colonne ou dernière ligne), il n'y a plus de choix à faire, on ne peut que descendre ou aller à droite.

Le cas de la troisième case $(n-1, p-1)$ est plus intéressant car si le chemin optimal passe par elle, il y a un choix à faire. On peut en effet soit descendre, soit aller à droite. C'est la plus petite des deux valeurs entre c_1 et c_2 qui indique le bon choix. Le poids du chemin optimal, s'il atteint cette case, augmentera de $M[n-1, p-1] + \min(c_1, c_2)$. Il est également nécessaire de mémoriser quelque part le choix qui a été fait.

On continue de procéder ainsi en remontant dans les étapes (les étapes possibles de même rang sont sur une même diagonale) jusqu'à remonter à la case de départ D.

Les structures de données nécessaires à la mise en place de cet algorithme sont deux matrices de nombres entiers *cout* et *choix* de n lignes et p colonnes.

$cout[i, j]$ indique le poids du chemin optimal qui mène de la case (i, j) à la case d'arrivée A.
$choix[i, j]$ indique le bon choix à faire si on se trouve dans la case (i, j) pour continuer le chemin optimal. On adopte la convention que 0 indique de descendre et 1 d'aller à droite.

Exemple : Si $M = \begin{pmatrix} \mathbf{1} & 4 & 6 & 8 & 2 \\ \mathbf{2} & \mathbf{4} & \mathbf{0} & 2 & 4 \\ 3 & 5 & \mathbf{0} & 8 & 9 \\ 8 & 0 & \mathbf{7} & \mathbf{6} & \mathbf{0} \\ 0 & 9 & 5 & 9 & \mathbf{1} \\ 2 & 7 & 5 & 5 & \mathbf{2} \end{pmatrix}$, alors on obtient :

$$cout = \begin{pmatrix} 23 & 24 & 22 & 26 & 18 \\ 22 & 20 & 16 & 18 & 16 \\ 24 & 21 & 16 & 17 & 12 \\ 24 & 16 & 16 & 9 & 3 \\ 21 & 26 & 17 & 12 & 3 \\ 21 & 19 & 12 & 7 & 2 \end{pmatrix} \quad \text{et} \quad choix = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & \end{pmatrix}$$

La case en haut à gauche de la matrice *cout* contient le poids du chemin optimal et grâce à la matrice *choix*, il est possible de reconstituer le chemin optimal qui est matérialisé par les nombres figurant en gras dans le tableau M.

2. Ecrire une fonction `construire_tableaux : int array array -> int array array * int array array` qui renvoie les matrices *cout* et *choix*. Ces deux matrices se remplissent en parallèle une diagonale après l'autre, comme le suggèrent les explications précédentes.
   Remarque : on peut le faire plus simplement.

3. Ecrire une fonction `affiche_chemin : int array array -> (int * int) list` qui renvoie les couples (i, j) de coordonnées des cases empruntées par le chemin optimal.