

Algorithmique - ROB3

TD4 - Programmation dynamique

Exercice 1 (Sous-séquence de plus grande somme) On dispose d'un tableau d'entiers relatifs de taille n . On cherche à déterminer la suite d'entrées consécutives du tableau dont la somme est maximale. Par exemple, pour le tableau $T = [5, 15, -30, 10, -5, 40, 10]$, la somme maximale est 55 (somme des éléments $[10, -5, 40, 10]$).

Question 1.1 Dans un premier temps, on s'intéresse uniquement à la valeur de la somme maximale.

- Quels sous-problèmes considérer pour résoudre ce problème ?
- Identifier la relation de récurrence entre les valeurs optimales des sous-problèmes.
- En déduire la valeur de la somme maximale.
- Ecrire un algorithme de programmation dynamique pour calculer cette somme.
- Quelle est la complexité de cet algorithme ?

Question 1.2 Modifier l'algorithme pour qu'il retourne également les indices de début et de fin de la somme. Sa complexité est-elle modifiée ?

→ **Exercice 2 (Découpe d'une corde)** Le magasin "A la vieille campeuse" achète des cordes d'escalade de longueur n et les découpe en cordes plus petites pour les vendre à ses clients. On souhaite déterminer un découpage optimal pour maximiser le revenu, sachant que les prix de ventes p_i d'une corde de i mètres sont donnés. Par exemple, supposons qu'on dispose d'une corde de $n = 10$ mètres, avec les prix de ventes indiqués dans le tableau suivant :

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	26

Question 2.1 Dans un premier temps, on suppose que chaque découpe est gratuite. On définit la densité d'une corde de longueur i comme étant le rapport p_i/i , c'est-à-dire son prix au mètre. Une stratégie gloutonne naturelle consisterait à découper une première corde de longueur i dont la densité soit maximale. On continuerait ensuite en appliquant la même stratégie sur la portion de corde restante de longueur $n - i$. Montrer que cette stratégie ne conduit pas toujours à un découpage optimal.

Question 2.2 On cherche à établir un algorithme de programmation dynamique pour déterminer le revenu de ventes maximal que l'on peut obtenir pour une corde, en supposant toujours que chaque découpe est gratuite.

- Quels sous-problèmes considérer pour résoudre ce problème ?
- Identifier la relation de récurrence entre les valeurs optimales des sous-problèmes.
- En déduire le revenu de ventes maximal qu'on peut tirer d'une corde.

- d. Ecrire un algorithme de programmation dynamique pour calculer ce revenu.
e. Quelle est la complexité de cet algorithme ?
f. Appliquer l'algorithme à l'exemple.

Question 2.3 Modifier l'algorithme précédent afin qu'il renvoie également la découpe optimale. Appliquer à l'exemple.

Par intuïtion **Question 2.4** On suppose maintenant que chaque découpe a un coût c (le même pour chaque découpe). Le revenu associé à un découpage particulier est alors la somme des prix de ventes moins les coûts de découpe. Reprendre la question 2 dans ce nouveau cadre. Le découpage optimal est-il le même ?

- **Exercice 3 (Jeu à deux joueurs)** Considérons une rangée de n pièces de valeurs v_1, \dots, v_n et le jeu suivant à deux joueurs : à chaque tour, le joueur peut prendre soit la pièce la plus à gauche, soit la pièce la plus à droite. Le jeu s'arrête quand il n'y a plus de pièce. On cherche à calculer le gain optimal du joueur qui commence ; on considérera que les deux joueurs adoptent une stratégie qui maximise leur profit final. Par exemple, pour 5, 3, 7, 11 le gain optimal pour celui qui commence est 16 (= 11 + 5) ; pour 8, 15, 3, 7 ce sera 22 (= 7 + 15).

Question 3.1 Indiquer une rangée de pièces pour laquelle il n'est pas optimal pour le premier joueur de commencer par prendre la pièce de plus grande valeur. Autrement dit, la stratégie gloutonne n'est pas optimale.

Question 3.2 Donner un algorithme en $O(n^2)$ pour calculer une stratégie optimale pour le premier joueur. Etant donnée la rangée initiale, l'algorithme calculera en $O(n^2)$ les informations nécessaires, et ensuite le premier joueur pourra déterminer en $O(1)$ le choix optimal à chaque tour en consultant les données précalculées.

Exercice 4 (Robot collecteur de pièces) Des pièces sont placées sur les cases d'un échiquier $\ell \times c$ (ℓ lignes et c colonnes). Il y a au plus une pièce par case. Un robot, positionné sur la case en haut à gauche de l'échiquier, va collecter le plus de pièces possibles et les placer sur la case en bas à droite. A chaque pas, le robot peut se déplacer d'une case vers la droite ou vers le bas. Quand le robot passe sur une case avec une pièce, il prend la pièce. Le but de cet exercice est de concevoir un algorithme pour collecter un nombre maximum de pièces.

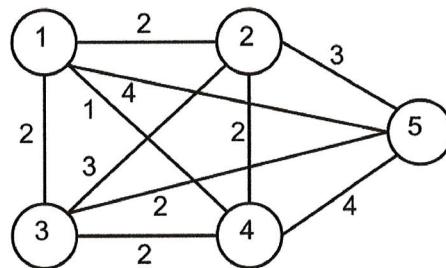
				○	
	○		○		
			○		○
		○			○
○				○	

— **Question 4.1** Dans un premier temps, on s'intéresse uniquement au nombre maximum de pièces qu'on peut collecter sans chercher à identifier la trajectoire correspondante.

- Quels sous-problèmes considérer pour résoudre ce problème ?
- Identifier la relation de récurrence entre les valeurs optimales des sous-problèmes.
- En déduire le nombre maximum de pièces qu'on peut collecter.
- Ecrire un algorithme de programmation dynamique pour calculer ce nombre.
- Quelle est la complexité de cet algorithme ?

Question 4.2 Modifier l'algorithme précédent afin qu'il retourne également la trajectoire optimale.

Exercice 5 (Voyageur de commerce) Dans le problème du voyageur de commerce, on dispose d'une matrice (ici supposée symétrique) d des distances entre n villes indiquées de 1 à n (d_{ij} et d_{ji} représentant la distance entre la ville i et la ville j), et le but est, en partant de 1, de visiter chaque ville exactement une fois et de revenir en 1 en minimisant la distance totale parcourue. Par exemple, dans le graphe ci-dessous, la distance totale parcourue pour la tournée $(1, 2, 3, 4, 5, 1)$ est 15, alors que la distance totale parcourue pour la tournée $(1, 3, 5, 2, 4, 1)$ est seulement 10.



Question 5.1 Donner la matrice d pour le graphe de l'exemple.

Question 5.2 Combien y a-t-il de tournées possibles pour n villes ?

Question 5.3 On cherche à établir un algorithme de programmation dynamique pour déterminer la valeur d'une tournée optimale.

- Quels sous-problèmes considérer pour résoudre ce problème ?
- Identifier la relation de récurrence entre les valeurs optimales des sous-problèmes.
- En déduire la valeur d'une tournée optimale.
- Ecrire un algorithme de programmation dynamique pour calculer cette valeur.
- Quelle est la complexité de cet algorithme ?

TD 4

Exercice 2:

p... 11

i | 1 2 3 4

1+3 → 12 €

(Globale)

p(i) | 1 7 11 12

2+2 → 14 €

(Optimale)

p_i/i | 1 3,5 > 3,5 3

21

$\pi(i)$ le revenu maximal qui on peut obtenir une corde de i métres.

b-

→ appart optimal du reste.

$$\pi(i) = \max_{1 \leq j \leq i} \{ p_j + \pi(i-j) \}$$

$$\pi(0) = 0.$$

c-

$\pi(n)$ on calcule $\pi(i)$ de $\pi(1)$ à $\pi(i)$.

Réénoncer le découpage-

i | 1 2 3 4 5 6 7 8 9 10

Prémière → t(i) | 2 3 2 2 6 1 2 3 2

corde coupée.

Découpage optimal: 2, 2, 6.

d-

Du bas vers le haut.

$$\pi[0] = 0$$

pour i allant de 1 à n faire

$$\pi[i] = \max -\infty$$

pour j allant de 1 à i faire

$$\text{si } p[j] + \pi[i-j] > \pi[i] \text{ alors}$$

$$\pi[i] = p[j] + \pi[i-j]$$

$$t[i] = j$$

// question 3.

retourner [n]

3 -

Reconstituer la décomposition optimale à partir de t .

$L \leftarrow$ Liste vide

$i \leftarrow m$.

Tant que $i \neq 0$ faire

$L \leftarrow L[i]$

$i \leftarrow i - t[i]$

retourner L

2 - e.

Complexité: m sous problèmes. $\Theta(m)$ par sous problème
 $\rightarrow O(m^2)$.

Exercice 4

$F(i, j)$: nb maxima de pièces qui on peut collecter en allant de $(l, 1)$ à (c, j)

$$F(l, j) = \sum_{k=1}^j c_{ik}$$

$$F(l, l) = \sum_{k=1}^l c_{lk}$$

si (l, j) contient une pièce sinon.

$$F(i, j) = c_{ij} + \max(F(i-1, j), F(i, j+1))$$

Valeur recherchée: $F(l, c)$

Complexité: $\Theta(l.c)$ $l.c$ sous problèmes. $\Theta(1)$

$$F(l, 1) = c[l, 1]$$

Pour j allant du 2 à c faire:

$$F[1, j] = c[1, j] + F[1, j-1]$$

Pour i allant du 2 à l faire:

$$F[i, 1] = F[i-1, 1]$$

Pour i allant du 2 à l faire

Pour j allant du 2 à c faire

$$F[i, j] = \max[F[i-1, j], F[i, j-1]] + c[i, j]$$

retourner $F[l, c]$

Exercici 3

11

8 15 3 7

21

$\text{opt}(i, j)$: différence entre le score du joueur 1 et 2 s'ils suivent chacun la différence de score stratégique optimale sur la rangée v_i, \dots, v_j

$\text{opt}(i,i) = v_i$ car le 1er paramètre égal au 2ème.

$$\text{opt}(i, j) = \max \left\{ v_i + \text{opt}(i+1, j), v_j + \text{opt}(i, j+1) \right\}$$

$$\text{chain } \rightarrow o(i, j) = i \text{ if } v_i - \text{opt}(i+1, j) > v_j - \text{opt}(i, j-1)$$

à prendre

$$\text{nimcm } n(i,j) = j.$$

pam : allant de l à m gain $\text{opt}(i,i') = v[i]$

pour k' allant de $\frac{1}{2}$ à $n-1$ faire.

pain i allant l à m-k* faire

$$\text{opt}[i, i+k] = \max(v_i - \text{opt}(i+1, i+k), v_{i+k} - \text{opt}(i, i+k-1))$$

Complexity : $\Theta(n^2)$.

