Programmation dynamique

Exercice 1 Triangle de Pascal

On veut calculer les coefficients binomiaux $C_n^k = \left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n!}{k!(n-k)!}$. Rappellons les propriétés suivantes :

- $\left(\begin{array}{c} n \\ k \end{array} \right) = \left(\begin{array}{c} n-1 \\ k-1 \end{array} \right) + \left(\begin{array}{c} n-1 \\ k \end{array} \right)$ pour $0 < k < n$,
- $\left(\begin{array}{c} n \\ n \end{array} \right) = 1$ et $\left(\begin{array}{c} n \\ 0 \end{array} \right) = 1$.

Question 1.1 Donner un algorithme récursif du calcul de $\left(\begin{array}{c} n \\ k \end{array} \right)$. Evaluer sa complexité.

Question 1.2 Ecrire l'algorithme qui retourne $\left(\begin{array}{c} n \\ k \end{array} \right)$ en utilisant la technique de la programmation dynamique. Evaluer sa complexité.

Exercice 2 Problème du stockage

Considérons n programmes $P_1, P_2, \ldots, P_n$ qui peuvent être stocker sur un disque dur de capacité D gigabytes.

- Chaque programme P_i a besoin s_i gigabytes pour être stocké et a une valeur v_i
- Tous les programmes ne peuvent pas être stockés sur le disque : $\sum_{i=1}^{n} s_i > D$.

Les programmes stockés dans le disque dur doivent maximiser la valeur totale, sans dépasser la capacité du disque dur. L'objectif est de concevoir un algorithme qui permet de calculer un tel ensemble.

Nous allons construire un tableau T dans lequel les lignes seront indexées par les programmes et les colonnes par les valeurs. L'élément $T[i, j]$ représentera la valeur maximale pour un disque dur de capacité j à l'aide des i premiers programmes.

Question 2.1 Donner la formule de récurrence.

Question 2.2 Donner l'algorithme utilisant la programmation dynamique.

Question 2.3 Donner la complexité de cet algorithme.

Remarque : Le problème de stockage peut se formuler sous forme du problème du Sac à Dos qui est un problème classique en informatique. Il modélise une situation analogue au remplissage d'un sac. Une personne veut remplir un sac à dos ne pouvant pas supporter plus d'un certain poids $D \in \mathbb{N}$, et elle dispose de n objets (On note l'ensemble des objects par $\mathcal{P} = \{1, \ldots, n\}$). Chaque objet i a une valeur $v_i \in \mathbb{N}\backslash\{0\}$ et un poids $s_i \in \mathbb{N}\backslash\{0\}$. Le problème est de trouver un ensemble d'objets tels que

- tous les objets de cet ensemble puissent être mis dans le sac.
- la somme des valeurs de ces objets soit maximale.

Exercice 3 Problème Le chemin le plus long dans un graphe

Soit $G = (V, E)$ un graphe orienté avec $V = \{v_1, \ldots, v_n\}$. On dit que G est ordonné si il vérifie les deux propriétés suivantes :

1. Chaque arc de ce graphe est de la forme $(i \to j)$ si $i < j$
2. Tous les sommets sauf le sommet v_n ont au moins un arc sortant.

Ici, par souci de simplification, nous supposerons qu'il existe un chemin allant de v_i vers v_n pour tout $i = 1, \ldots, n$.

L'objectif est de trouver le chemin le plus long entre les sommets v_1 et v_n.

Question 3.1 Montrer que l'algorithme glouton suivant ne résoud pas correctement le problème.

1. $u \leftarrow v_1$;
2. $L \leftarrow 0$;
3. Tant qu'il existe un arc sortant du sommet u
   (a) choisir l'arc $(u \to v_j)$ tel que j est le plus petit possible
   (b) $u \leftarrow v_j$;
   (c) $L \leftarrow L + 1$;
4. retourner L

Question 3.2 Donner la formule de récurrence qui permet de calculer la longueur du chemin le plus long commençant par v_1 finissant par v_ℓ.

Question 3.3 Donner un algorithme qui retourne la longueur du chemin le plus long commençant par v_1 finissant par v_n.

Question 3.4 Modifier l'algorithme précédent afin qu'il retourne le chemin.

Nous allons considérer les graphes orientés et ordonnés possédant une fonction de poids w sur les arcs $w : E \to \mathbb{N}^+$.

L'objectif est de trouver le poids du chemin de poids maximal commençant par v_1 finissant par v_n (si il n'en existe pas, la valeur retournée doit être égale à $-\infty$). Formellement, on veut trouver $p_{max} = max\{\sum_{e \in \mathcal{P}} w(e) : \mathcal{P}$ est un chemin de v_1 à $v_n\}$

Question 3.5 Donner la formule de récurrence permettant de calculer le chemin de poids maximum commençant par v_1 finissant par v_n.

Question 3.6 En déduire l'algorithme.

Maintenant, l'objectif est de trouver le poids du chemin de poids maximal commençant par v_1 composé de k arcs. Nous utiliserons un tableau T dont la valeur de l'élément $T[i, \ell]$ correspond au poids du chemin de poids maximal commençant par v_1 finissant par v_i et composé de ℓ arcs.

Question 3.7 Donner la formule de récurrence qui permet de calculer le chemin de poids maximal de ℓ arcs commençant par v_1 finissant par v_i . En déduire l'algorithme.

Exercice 4 Planning

Considérons un chef de projet qui doit gérer une équipe en lui affectant un projet qui dure une semaine. Le chef de projet doit choisir si il prend un projet « stressant » ou « non-stressant » pour la semaine.

1. Si le chef de projet choisit le projet qui n'est pas « stressant » durant la semaine i, alors l'entreprise reçoit un revenu ℓ_i.
2. Si le chef de projet choisit le projet qui est « stressant » durant la semaine i, alors l'entreprise reçoit un revenu h_i et l'équipe ne travaille pas durant la semaine $i-1$

Exemple : Si chef de projet choisit de se reposer à la semaine 1, puis de prendre le projet « stressant » à la semaine 2, puis de prendre les projets « non-stressant » à la semaine 3 et 4. Le revenu total est de 70 et il correspond au maximun.

	semaine 1	semaine 2	semaine 3	semaine 4
ℓ	10	1	10	10
h	5	50	5	1

Question 4.1 Montrer que l'algorithme suivant ne résoud pas correctement le problème.

1. pour chaque itération $i = 1, \ldots, n$
   (a) si $h_{i+1} > \ell_i + \ell_{i+1}$ alors
      i. Choisir « Ne pas travailler à la semaine i »
      ii. Choisir « le projet « stressant » à la semaine i+1 »
      iii. Continuer à l'itération $i + 2$
   (b) sinon
      i. Choisir « le projet « non-stressant » à la semaine i »
      ii. Continuer à l'itération $i + 1$

Question 4.2 Donner un algorithme qui retourne le revenu maximal que peut obtenir le chef de projet.

Exercice 5 Multiplications chaînées de matrices.

On veut calculer le produit de matrices $M = M_1 M_2 \cdots M_n$. Multiplier une matrice $p \times q$, par une matrice $q \times r$ en utilisant la méthode standard nécessite pqr produit scalaire.

Question 5.1 Considérons 4 matrices $A : 20 \times 5$, $B : 5 \times 100$, $C : 100 \times 8$, $D : 5 \times 30$. On veut calculer le produit $ABCD$. En fonction des parenthétisations, le nombre de produits varie.

Déterminer le nombre de produits pour calculer $ABCD$, si on utilise les parenthétisations suivantes : $((AB)C)D$ ou $(A(BC))D$

L'objectif est de concevoir un algorithme de meilleure parenthétisation qui permet de minimiser le nombre de produits scalaires.

Nous noterons que la matrice M_i est de dimension $d_{i-1} \times d_i$.

Définissons le nombre minimal de produits scalaires nécessaires pour évaluer le produit des matrices $M_i \times M_{i+1} \ldots M_{j-1} \times M_j$ par $c(i, j)$.

Question 5.2 Ecrire une formule de récurrence pour calculer $c(i, j)$.

Question 5.3 Ecrire un algorithme utilisant la programmation dynamique