Université Mohamed Boudiaf de M'sila
Faculté de MI
Département d'informatique

Master 1 IDO 2021/2022
Optimisation combinatoire 1
TD 3 / PROGRAMMATION DYNAMIQUE

Exercice 1

Triangle de Pascal

On veut calculer les coefficients binomiaux $C_n^k = \begin{pmatrix} n \\ k \end{pmatrix} = \frac{n!}{k!(n-k)!}$.

Rappelons les propriétés suivantes :

$\begin{pmatrix} n \\ k \end{pmatrix} = \begin{pmatrix} n-1 \\ k-1 \end{pmatrix} + \begin{pmatrix} n-1 \\ k \end{pmatrix}$ pour $0 < k < n$, $\begin{pmatrix} n \\ n \end{pmatrix} = 1$ et $\begin{pmatrix} n \\ 0 \end{pmatrix} = 1$.

1. Donner un algorithme récursif du calcul de $\begin{pmatrix} n \\ k \end{pmatrix}$. Evaluer sa complexité.
2. Ecrire l'algorithme qui retourne $\begin{pmatrix} n \\ k \end{pmatrix}$ en utilisant la technique de la programmation dynamique. Evaluer sa complexité.

Exercice 2

Problème du stockage

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

- Chaque programme P_i a besoin de 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.

1. Donner la formule de récurrence.
2. Donner l'algorithme utilisant la programmation dynamique.
3. Donner la complexité de cet algorithme.

Exercice 3

Problème de chemin le plus long dans un graphe

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

1. Chaque arc de ce graphe est de la forme $(i \rightarrow j)$ 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 $i = 1,...,n$.

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

1. Montrer que l'algorithme glouton suivant ne résout pas correctement le problème.

   $u \leftarrow v_1$;
   $L \leftarrow 0$;
   Tant qu'il existe un arc sortant du sommet u
   choisir l'arc $(u \rightarrow v_j)$ tel que j est le plus petit possible
   $u \leftarrow v_j$;
   $L \leftarrow L + 1$;
   retourner L
2. Donner la formule de récurrence qui permet de calculer la longueur du chemin le plus long commençant par v1 finissant par v`.
3. Donner un algo qui retourne la longueur du chemin le plus long commençant par v1 finissant par vn.

4. Modifier l'algorithme précédent afin qu'il retourne le chemin.
5. Donner la formule de récurrence permettant de calculer le chemin de poids maximum commençant par v1 finissant par vn.
6. En d´déduire l'algorithme.
7. Donner la formule de récurrence qui permet de calculer le chemin de poids maximal de ` arcs commençant par v1 finissant par vi . 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.

a) Si le chef de projet choisit le projet qui n'est pas stressant durant la semaine i, alors l'entreprise reçoit un revenu j.
b) Si le chef de projet choisit le projet qui est stressant durant la semaine i, alors l'entreprise recroit 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 maximum.

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

1. Montrer que l'algorithme suivant ne résout pas correctement le problème.

pour chaque itération $i = 1,...,n$
  si $h_{i+1} > l_i + l_{i+1}$ alors
    Choisir Ne pas travailler à la semaine i
    Choisir le projet stressant à la semaine i+1
    Continuer à l'itération $i + 2$
  sinon
    Choisir le projet non-stressant à la semaine i
    Continuer l'itération $i + 1$

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

Exercice 5

Multiplications chainées de matrices.

On veut calculer le produit de matrices $M = M_1M_2 \cdots M_n$. Multiplier une matrice $p \times q$, par une matrice $q \times r$ en utilisant la méthode standard nécessite pqr produit scalaire.

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

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

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

3. Ecrire un algorithme utilisant la programmation dynamique

Exercice 6

1. Donner un algorithme de programmation dynamique pour résoudre le problème suivant :
   Entrée : une matrice A de taille $n \times m$ où les coefficients valent 0 ou 1.
   Sortie : la largeur maximum K d'un carré de 1 dans A, ainsi que les coordonnées (I, J) du coin en haut à gauche d'un tel carré (autrement dit pour tout i, j, $I \leq i \leq I + K - 1$, $J \leq j \leq J + K - 1$, $A[i, j] = 1$).
2. Quelle est sa complexité ?

Exercice 7

La bibliothèque planifie son déménagement. Elle comprend une collection de n livres $b_1, b_2, \ldots, b_n$. Le livre b_i est de largeur w_i et de hauteur h_i. Les livres doivent être rangés dans l'ordre donné (par valeur de i croissante) sur des étagères identiques de largeur L.

1. On suppose que tous les livres ont la même hauteur $h = h_i, 1 \leq i \leq n$. Montrer que l'algorithme glouton qui range les livres côte à côte tant que c'est possible minimise le nombre d'étagères utilisées.
2. Maintenant les livres ont des hauteurs différentes, mais la hauteur entre les étagères peut se régler. Le critère à minimiser est alors l'encombrement, défini comme la somme des hauteurs du plus grand livre de chaque étagère utilisée.
   - **(a)** Donner un exemple où l'algorithme glouton précédent n'est pas optimal.
   - **(b)** Proposer un algorithme optimal pour résoudre le problème, et donner son coût.
3. On revient au cas où tous les livres ont la même hauteur $h = h_i, 1 \leq i \leq n$. On veut désormais ranger les n livres sur k étagères de même longueur L à minimiser, où k est un paramètre du problème. Il s'agit donc de partitionner les n livres en k tranches, de telle sorte que la largeur de la plus large des k tranches soit la plus petite possible.
   - **(a)** Proposer un algorithme pour résoudre le problème, et donner son coût en fonction de n et k.
   - **(b)** On suppose maintenant que la la taille d'un livre est en $2^{o(kn)}$. Trouver un algorithme plus rapide que le précédent pour répondre à la même question.

Exercice 8

Étant donné un tableau T de n entiers relatifs, on cherche $\max\{\forall i, j \in \{1 \ldots n\} : \sum_{k=i}^{j} T[k]\}$.

Par exemple pour le tableau suivant :

2	18	−22	**20**	**8**	**−6**	**10**	−24	13	3

L'algorithme retournerait la somme des éléments 4 à 7 soit 32.

1. Donner un algorithme retournant la somme maximale d'éléments contigus par une approche *diviser pour régner*.
2. Donner un algorithme retournant la somme maximale d'éléments contigus par *programmation dynamique*.
3. Comparer la complexité Asymptotique au pire cas des deux approches.
4. Peut-on adapter l'algorithme de programmation dynamique en dimension 2? Plus formellement, étant donnée une matrice M de $n \times m$ entiers relatifs, on cherche $\max\{\forall i, j \in \{1 \ldots n\}, \forall k, l \in \{1 \ldots m\} : \sum_{k_1=i}^{j} \sum_{k_2=k}^{l} M[k_1][k_2]\}$.