Cours d'Informatique
ECM 1A
TD6 Algo - Programmation Dynamique

November 27, 2024

I. 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 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 $i = 1, ..., n$. 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 résoud pas correctement le problème:

$u = v_1$
$L = 0$
while (il existe un arc sortant du sommet u) **do**
    Choisir l'arc $(u \to v_j)$ tel que j est minimal
    $u = v_j$
    $L = L + 1$
    Retourner L

(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.

(3) Donner un algorithme qui retourne la longueur du chemin le plus long commençant par v_1 finissant par v_n.

(4) Modifier l'algorithme précédent afin qu'il retourne le chemin.

(5) Dans la suite, nous considérerons 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 $pmax = max_{P \text{ chemin de } v_1 \text{ à } v_n} \sum_{e \in \mathcal{P}} w(e)$.
Donner la formule de récurrence permettant de calculer le chemin de poids maximum commençant par v finissant par v_n.

(6) En déduire l'algorithme.

II. Mise en page et impression d'un texte (Examen 2022).

On dispose d'un fichier texte sans passage à la ligne (il s'agit d'un seul paragraphe) que l'on veut imprimer sur une feuille de taille donnée. Chaque caractère imprimé occupe la même

largeur, y compris les espaces, comme sur une machine à écrire mécanique. Le problème est de découper le texte en lignes, pour minimiser les espaces qui restent à la fin de chaque ligne. Un mot ne peut pas être découpé.

Formellement, le texte est une suite de k mots de longueurs $(l_1, l_2, \ldots, l_k)$ mesurées en nombre de caractères. Chaque ligne contient exactement M caractères, blancs compris. On suppose que toutes les longueurs $(l_1, l_2, \ldots, l_k)$ sont inférieures ou égales à M (un mot peut tenir entièrement dans une ligne). Si une ligne contient les mots j à i (inclus), où $i \geq j$, et qu'on laisse exactement un espace entre deux mots, le nombre de caractères blancs à la fin de la ligne est:

$$B = M - j + i - \sum_{k=j}^{i} l_k$$

On veut minimiser la somme, sur toutes les lignes, des cubes (c'est-à-dire : à la puissance 3) des valeurs B de chaque ligne (nombres de caractères blancs présents à la fin de la ligne). L'utilisation du cube plutôt que juste la somme des valeurs B vient du fait que l'on veut pénaliser les grands espaces à la fin de lignes. Cette somme est appelée coût. Si on minimise ce coût on dit que la présentation du texte est « équilibrée ».

Exemple du problème:

- Longueur de la ligne $M = 14$
- Texte : "La récursion est un concept important"

Dans ce cas, on a *k=6* et : *l1 =2, l2 =9, l3 =3, l4 =2, l5 =7* et *l6 =9.*

Pour l'alignement optimal on trouve :
La récursion $\rightarrow B = 2$
est un concept $\rightarrow B = 0$
important $\rightarrow B = 5$

On a successivement des valeurs de B égales à 2, 0 et 5 et donc une somme des cubes de 133.

(1) Donnez un algorithme naïf (glouton) pour ce problème.

(2) Montrez que cet algorithme peut échouer en l'appliquant sur le texte suivant; pour des lignes de longueur 17 : "La programmation dynamique est une des bases algorithmiques"
avec un cout $= Cout = 1 + 4 * 4 * 4 + 4 * 4 * 4 + 3 * 3 * 3 = 156$

(3) Donner un algorithme de programmation dynamique qui résout le problème.
Indication : Donner une relation de récurrence pour $cout(j)$ qui dénote le coût de placer les mots de *1* à *j* (quel que soit le nombre de lignes utilisées pour cela). On pourra par exemple utiliser une quantité $cout\_ligne(j)$ qui est égale au coût d'une ligne contenant les mots de i à j.

(4) Donner la complexité en temps et en espace de l'algorithme précédent.

(5) Comment faut-il modifier l'algorithme pour connaître à la fin de l'algorithme le nombre de caractères écrit dans chacune des lignes pour la solution optimale (si votre algorithme ne le fait pas déjà)?

(6) Comment faut-il modifier l'algorithme pour que la dernière ligne ne soit pas prise en compte dans la somme des cubes des nombres de caractères blancs ?

III. **Montée des marches.**

On considère la montée des marches d'un escalier. Avant de le gravir, vous vous posez la question de comment vous allez le gravir.

(1) Quel est le nombre de manières de monter m marches en les montant une par une et/ou deux par deux ?

(2) Généralisons. Soit un escalier à m marches. On considère que l'on peut gravir l'escalier par différents sauts de α marches, pour $\alpha \in A$ avec $A = \{\alpha_1, \alpha_2, \ ..., \ \alpha_p\}$. Par exemple, dans la question 1, on a $p = 2$ et $A = \{\alpha_1 = 1, \alpha_2 = 2\}$. On suppose que les α_i sont rangés par ordre croissant et sont tous différents. On appelle $N(m)$ le nombre de manière de monter les m premières marches de cet escalier. Donner une formule de récurrence pour $N(m)$.

(3) Donner l'algorithme de programmation dynamique associé (en pseudo code) et la complexité de cet algorithme.

(4) On souhaite afficher une solution possible pour gravir l'escalier. Indiquez comment procéder. Ecrivez-un pseudo code correspondant.

IV. **Mission sur Mars (Examen S6 2023).**

Un projet de recherche spatiale doit résoudre un problème avant que des êtres humains ne puissent se rendre sur la planète Mars.

Quatre équipes de recherche indépendantes essaient des approches différentes dans cet objectif. On a estimé que leurs probabilités d'échec sont respectivement de 0.40 pour l'équipe 1, 0.60 pour l'équipe 2, 0.80 pour l'équipe 3, et 1.0 pour l'équipe 4. Ainsi la probabilité qu'aucune des équipes ne réussisse est de $0.40 \times 0.60 \times 0.80 \times 1.0 = 0.192$.

Pour minimiser la probabilité d'échec total, trois scientifiques supplémentaires de haut niveau ont été recrutés et sont affectés au projet. Il reste à les répartir dans les équipes. Le tableau suivant donne les probabilités d'échec de chaque équipe si 0, 1, 2 ou 3 de ces nouveaux scientifiques lui sont affectés.

	Probabilité d'échec $P(i,j)$			
Nb scientifiques	Équipe			
	I	II	III	IV
0	0.40	0.60	0.80	1.00
1	0.20	0.40	0.50	0.90
2	0.15	0.20	0.30	0.10
3	0.10	0.19	0.20	0.10

On note ces probabilités $P(i,j)$, où $P(i,j)$ correspond à la probabilité d'échec de l'équipe numéro i si on lui affecte j nouveaux scientifiques.

(1) Proposez un algorithme naïf (éventuellement non optimal) pour déterminer la répartition conduisant à une probabilité d'échec minimale ?

(2) On veut construire un algorithme de programmation dynamique qui trouve une solution optimale. On s'intéresse à calculer le taux d'erreur optimal si on considère les k premières équipes de recherches ($1 \leq k \leq 4$) et j scientifiques ($0 \leq j \leq 3$). On note ce taux $f(k,j)$. Pouvez vous exprimer une récurrence sur les quantités *f(k,j)*, ainsi que des valeurs de quantités initiales (Les $f(k,j)$ pour *j=0* par exemple) ? Justifiez vos réponses.

(3) Déduire de la réponse à la question précédente un algorithme (itératif) de programmation dynamique pour résoudre ce problème. Utilisez cet algorithme sur l'exemple.

(4) Quelles sont les complexités algorithmiques des 2 algorithmes de la question 1 et de la question 3 ? Justifiez vos réponses.

(5) On souhaite, en plus de trouver la solution qui minimise la probabilité d'échec, savoir où affecter les scientifiques pour implémenter cette solution optimale. Comment faire ?

(6) L'équipe 2 a été retirée du projet. Peut-on déduire de la réponse à la question 3 la nouvelle probabilité d'échec du projet ? Sinon, expliquez succinctement comment procéder ?

V. **Distance de Levenstein.**

On s'intéresse au calcul de la distance de Levenstein entre deux mots m_1 et m_2 de longueurs respectives n et p. On utilise l'algorithme de programmation dynamique tel que décrit en cours qui calcule itérativement les distances entre les préfixes $m1[0 : i]$ et $m2[0 : j]$ pour tout couple (i, j). On rappelle la récursion :

$$D(i,j) = min \begin{cases} D(i-1, j-1) + cout_{sub} \\ D(i-1, j) + cout_{del} \\ D(i, j-1) + cout_{ins} \end{cases} \quad (1)$$

(1) En considérant tous les coûts égaux à 1, quelle est la distance entre les mots *niche* et *chiens* ?

(2) Dans le cas général les valeurs $D(i, j)$ de D étant dépendantes les unes des autres, dans quel ordre peut-on les calculer ?

(3) Ecrire le pseudo code / l'algorithme du calcul de la distance entre deux mots m_1 et m_2

(4) La distance obtenus correspond à un alignement optimal des 2 séquences. Peut on le retrouver à la fin de l'algorithme que vous avez écrit ? Si oui expliquez comment, si non modifiez votre algorithme.