Agrégation de mathématiques, option informatique
Algorithmique
Programmation dynamique

1er décembre 2014

1 Introduction

`Exercice 1` `Plus longue sous-suite croissante`

Dasgupta et al. Algorithms, p. 157

Soit $a_1, \ldots, a_n$ une suite finie de nombres. Une sous-suite de $a_1, \ldots, a_n$ est une suite $a_{i_1}, \ldots, a_{i_k}$ où $1 \leq i_1 < \cdots < i_k \geq n$. On cherche ici à calculer la plus longue sous-suite croissante de $a_1, \ldots, a_n$. Par exemple, la plus longue sous-suite croissante de 5, 2, 8, 6, 3, 6, 9, 7 est 2, 3, 6, 9.

1. Écrire un algorithme qui calcule la longueur de la plus longue sous-suite croissante de $a_1, \ldots, a_n$.

(Considérer le calcul de la plus longue sous-suite croissante qui termine par a_k comme sous-problème.)

2. Ecrire un algorithme qui retourne la plus longue sous-suite croissante de $a_1, \ldots, a_n$.

2 Sous-séquence commune

Cormen, et al. Introduction à l'algorithmique, p. 308

Soit deux chaînes de caractères $x[1, \ldots, n]$ et $y[1, \ldots, m]$ de longueurs respectives n et m. On cherche une plus longue sous-séquence (comprendre séquence extraite) commune à x et y. Par exemple, une plus longue sous-séquence commune à `CRAPULE` et `PERCEUSE` est `RUE` (`C R AP U L E` et `PE R CE U S E`).

3. Donner une plus longue sous-séquence commune aux chaînes `ABCBDBC` et `ADCDAB`.

4. Caractériser la structure d'une solution optimale (étude de cas sur la dernière lettre).

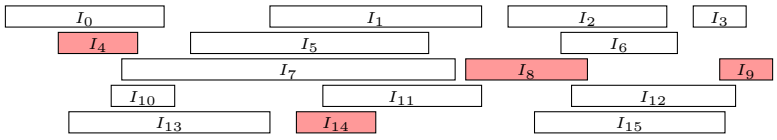
5. Proposer un algorithme de programmation dynamique pour calculer la longueur d'une plus longue sous-séquence commune.
6. Adapter l'algorithme pour construire une plus longue sous-séquence commune.
7. Quel est l'espace mémoire utilisé ?
8. Proposer une version n'utilisant que $\min(n, m) + 2$ emplacements mémoire si on ne souhaite connaître que la longueur d'une plus longue sous-séquence commune.

3 Quand glouton ne fonctionne plus

`Exercice 2` `Ordonnancement d'intervalles`

Kleinberg, Tardos, Algorithm design, p. 12

On considère ici le problème d'*ordonnancement d'intervalles*. En entrée, nous avons deux séquences $s = (s_1, \ldots, s_n)$ et $f = (f_1, \ldots, f_n)$ avec $s_i \leq f_i$ pour tout $i \in \{1, \ldots, n\}$. On dira que $J \subseteq \{1, \ldots, n\}$ est compatible avec (s, f) si la famille $([s_i, f_i])_{i \in J}$ est constituée d'intervalles disjoints deux à deux. Le but est de trouver J compatible de cardinalité maximale.



1. On peut représenter ce problème avec un graphe : chaque intervalle est un nœud et on relie deux intervalles s'ils ont une intersection non vide. A quel problème se réduit le problème d'ordonnancement d'intervalles ? Est-ce une bonne idée ?
2. Écrire un algorithme glouton qui répond au problème.

(par f_i croissant)

On s'intéresse maintenant à la version avec poids. En entrée, nous avons toujours nos deux séquences $s = (s_1, \ldots, s_n)$ et $f = (f_1, \ldots, f_n)$ mais aussi une troisième séquences $w = (w_1, \ldots, w_n)$ de poids. Le but est de trouver J compatible avec $w(J) = \sum_{j \in J} w_j$ maximal. $w(J)$ s'appelle le poids de J. On suppose que $f_1 \leq f_2 \leq \ldots f_n$.

3. Écrire un algorithme de programmation dynamique qui répond au problème.

4 Approximation d'un problème NP-complet

`Exercice 3` `Problème du sac-à-dos`

Dasgupta et al. Algorithms, p. 167, p. 283

On dispose de n objets ayant respectivement un poids $p_1, \ldots, p_n \in \mathbb{N}$ et une valeur $v_1, \ldots, v_n \in \mathbb{N}$. On dispose d'un sac-à-dos qui suppose au maximum un poids total de $P_{\max} \in \mathbb{N}$ et on veut le remplir avec ces objets en maximisant la valeur totale des objets emportés dans le sac, sans dépasser le poids $P_{\max}$.

p_1 kg
v_1€

p_2 kg
v_2€

...

p_n kg
v_n€

$P_{\max}$ kg au max

1. Proposer un premier algorithme de programmation dynamique où les sous-problèmes sont

> calculer la valeur maximale que l'on peut atteindre en mettant des objets parmi $\{1, \ldots, k\}$ sans dépasser le poids p.

Donner sa complexité (en temps et en mémoire).

2. Appliquer votre algorithme pour trouver l'optimum avec les données suivantes

i	1	2	3	4	5
p_i	1	2	5	6	7
v_i	1	6	18	22	28

et un poids maximum de $P_{\max} = 11$.

3. Proposer un deuxième algorithme de programmation dynamique où les sous-problèmes sont

> calculer le poids minimal que l'on doit atteindre si on met des objets parmi $\{1, \ldots, k\}$ et que la valeur dépasse v.

Donner sa complexité (en temps et en mémoire).

4. Proposer un algorithme en $O(\frac{n^3}{\epsilon})$ qui calcule une solution où la valeur totale est entre la valeur optimale v^* et $v^*(1-\epsilon)$.

5 Autres exemples

Problème	Référence	Découpage	Dimensions	Remarque
Segmented Least Squares	[Kleinberg et Tardos, 6.3, p. 261]	*	1	
Sac à dos	[Kleinberg et Tardos, 6.4, p. 267] [Papadimitriou et al., 6.4, p. 181]	2	2	illustre la NP-complétude faible, algorithme d'approximation aussi précis que l'on souhaite
Seconde structure ARN	[Kleinberg et Tardos, 6.5, p. 272]	*	2	illustre la difficulté pour trouver des sous-problèmes
Distance d'édition	[Kleinberg et Tardos, 6.6, p. 278] [Papadimitriou et al., 6.3, p. 174]	3	2	
Bellman-Ford	[Kleinberg et Tardos, 6.8, p. 290] [Papadimitriou et al., 6.6, p. 186]	*	2	Montrer comment économiser de l'espace
Floyd-Warshall	[Papadimitriou et al., 6.6, p. 186]	*	2	Aplatissement de l'espace des problèmes
Voyageur de commerce	[Papadimitriou et al., 6.6, p. 186]	*	2 (une est exponentielle)	Problème NP-complet
Plus court chemin dans un DAG	[Papadimitriou et al., 6.1, p. 169]	*	1	
Plus longue sous-suite croissante	[Papadimitriou et al., 6.2, p. 171]	*	1	
Plus longue sous-suite commune à deux chaînes	[Cormen et al., 15.4, p. 362]	3	2	Montrer comment économiser de l'espace
Multiplication de matrices en chaîne	[Papadimitriou et al., 6.5, p. 184]	*	2	
Ensemble indépendant sur des arbres	[Papadimitriou et al., 6.7, p. 189]	*	1	Restriction sur les entrées d'un problème NP-complet
Arbres binaires de recherche optimaux	[Cormen et al., 15.5, p.368]	*	2	Montrer comment économiser de l'espace