Prof. F. Eisenbrand EPFL - DISOPT

Optimisation Discrète

Semestre de printemps 2013

Série 10 **16 mai 2013**

Remarque générale :
Si vous voulez obtenir un bonus pour l'évaluation finale, vous pouvez rendre des solutions écrites aux exercices notés, au plus tard le **lundi 27 mai**, avant 12h dans la boîte au bureau MA B1 533.
Le rendu peut être fait en groupe de trois personnes au plus.

Exercice 1 $(*)$
Deux joueurs A et B conviennent de participer au jeu suivant :
De façon alternée, ils lancent en l'air une pièce équilibrée ; le côté face rapporte un point à A, le coté pile à B. Le premier à n points remporte la victoire.
Considérer le problème suivant : Donné deux nombres entiers $a, b \in \mathbb{N}$, quelle est la probabilité que le joueur A gagne si A et B ont déjà a et b points ?
Trouver un algorithme qui résout le problème en temps $O(n^2)$. La déscription de l'algorithme doit comporter :

(a) Une déscription en mots d'une entrée du tableau du programme dynamique

(b) Une déscription (en langage mathématique ou quelques phrases compréhensibles) comment on peut calculer une telle entrée

(c) Un argument de correction

Solution
Une entrée du tableau $T(x, y)$ correspond à la probabilité que le joueur A gagne si A et B ont déjà a et b points pour $0 \leq x, y \leq n$.
D'abord, on pose $T(n, y) = 1$ pour tout $y \in \{0, \ldots, n-1\}$ et $T(x, n) = 0$ pour tout $x \in \{0, \ldots, n-1\}$. Ensuite, toutes les autres entrées peuvent être calculées comme suit :

$$T(x, y) = \frac{1}{2} \cdot T(x+1, y) + \frac{1}{2} \cdot T(x, y+1)$$

Voici un algorithme complet :
`For` $x = n-1$ `downto` 0 `do`
    `For` $y = n-1$ `downto` 0 `do`
        $T(x, y) = \frac{1}{2} \cdot T(x+1, y) + \frac{1}{2} \cdot T(x, y+1)$

En utilisant cet ordre, le calcul de chaque nouvelle entrée ne considère clairement que des entrées calculées précédemment.

Correction : Avec probabilité $\frac{1}{2}$ le joueur A marque un point de plus et avec probabilité $\frac{1}{2}$ le joueur B. En termes de probabilités, on a

$$Pr[A \text{ gagne} \mid \text{déjà } (a,b) \text{ points}] = Pr[A \text{ gagne} \mid \text{déjà } (a+1,b) \text{ points}] \cdot Pr[A \text{ marque le point}] + Pr[A \text{ gagne} \mid \text{déjà } (a,b+1) \text{ points}] \cdot Pr[B \text{ marque le point}]$$

Le temps d'exécution est clairement $O(n^2)$ car on peut calculer chaque entrée en temps $O(1)$.

Exercice 2

Soit $D = (V, A)$ un graphe orienté avec poids $w : A \to \mathbb{R}_{\geq 0}$.

(i) Soient $k \in \mathbb{N}$ et $s, w \in V$ deux sommets. De plus, soit P le chemin le plus court de s à w en utilisant $\leq k$ arcs et soit $a = (v, w)$ l'arc final de ce chemin.
Démontrer que $P \setminus \{a\}$ est un plus court chemin de s à v en utilisant $\leq k-1$ arcs.

(ii) Décrire un algorithme qui trouve les chemins les plus courts depuis un sommet $s \in V$ à l'aide de la programmation dynamique.

Solution

(i) Supposons que Q est un chemin de s à v en utilisant $\leq k-1$ arcs et Q est plus court que $P \setminus \{a\}$. Si Q ne contient pas w, $Q + a$ est un chemin de s à w plus court que P en utilisant $\leq k$ arcs. Sinon, on peut raccourcir $Q + a$ (qui contient un cycle) pour trouver un chemin de s à w plus court que P (à noter que les poids sont non-négatifs). En tout cas, on obtient une contradiction.

(ii) Une entrée du tableau $T[k, j]$ signifie la longueur du plus court chemin entre s et j en utilisant $\leq k$ arcs. La formule par récurrence est

$$T[k, j] = \min\{T[k-1, j], \min_{(i,j) \in A}\{T[k-1, i] + w(i, j)\}\}.$$

On pose $T[0, s] = 0$ pour initialiser le tableau.

Remarque : L'algorithme à la base de ce programm dynamique est celui de Dijkstra pour trouver les chemins les plus courts.

Exercice 3

Le problème du rendu de monnaie est le suivant : étant donné un système de monnaie avec n pièces différentes, comment rendre une somme donnée M de façon optimale, c'est-à-dire avec le nombre minimal de pièces et billets ?
Donner un algorithme qui résout ce problème avec l'aide de la programmation dynamique.

Solution

Un moyen conceptuellement simple de trouver le nombre $N_S(v)$ de pièces permettant de rendre la valeur v dans le système de pièces $S = (c_1, \ldots, c_n)$, est la programmation dynamique. En effet, supposons qu'on sache rendre de façon optimale toutes les valeurs strictement inférieure à v. Pour rendre v, il faut au moins une pièce, à prendre parmi n possibles. Une fois choisie cette pièce, la somme restante est inférieure strictement à v, donc on sait la rendre de façon optimale. Il suffit donc d'essayer les n possibilités :

$$N_S(v) = 1 + \min_{\substack{1 \le i \le n \\ v - c_i \ge 0}} N_S(v - c_i).$$

On pose $N_S(0) = 0$ pour initialiser le tableau.
Pour calculer la façon optimale de rendre une somme M, il faut considérer un tableau de (dimension 1 et de) taille M. Le calcul à effectuer pour chaque entrée est proportionnel à n, alors le temps total est $O(n \cdot M)$ et donc exponentiel en la taille de M.
Ci-contre, un exemple montrant le calcul de $N_{(1,7,23)}(28)$ par programmation dynamique. Un arbre est construit. Sa racine est la somme à rendre. Un nœud représente une somme intermédiaire. Les fils d'un nœud correspondent aux sommes restant à rendre selon la dernière pièce rendue (1, 7, ou 23, chaque pièce correspondant à une couleur d'arête attitrée). La construction de l'arbre se fait en largeur d'abord, c'est-à-dire qu'on calcule les fils de tous les nœuds d'un niveau avant de passer au niveau suivant, et on s'arrête dès qu'un nœud 0 est trouvé : le chemin allant de la racine à ce nœud permet de rendre la monnaie avec un minimum de pièces. Ici, on atteint 0 en rendant quatre pièces de 7 (chemin vert de longueur quatre), donc $N_{(1,7,23)}(28) = 4$.

Exercice 4
Considérer le problème du sac à dos.
Étant donné n objets avec poids $w_1, \ldots, w_n \in \mathbb{N}$ et valeurs $c_1, \ldots, c_n \in N$, il s'agit de choisir un sous-ensemble des objets qui maximise la valeur totale sans dépasser le poids maximum (ou la capacité) W du sac à dos.
Trouver un algorithme qui le résout en temps $O(n \cdot W)$.

Solution

L'idée pour cet algorithme est d'échanger les rôles des poids et des valeurs pour l'algorithme vu en cours.
Au lieu de construire un graphe comme en cours, on présente ici l'algorithme à l'aide d'un tableau.
L'entrée du tableau $T[i, x]$ correspond à la valeur maximale que l'on peut obtenir parmi les i premiers objets avec capacité x ($1 \le i \le n$ et $0 \le x \le W$). La formule par récurrence est

$$T[i, x] = \max\{T[i-1, x], T[i-1, x - w_i] + c_i\}$$

(si $x - w_i \ge 0$, bien sûr) et on initialise $T[0, x] = 0$ pour tout $0 \le x \le W$. Observer que le premier terme du maximum correspond à ne pas prendre l'objet i tandis que le deuxième terme correspond à prendre l'objet i dans la solution (alors on

obtient sa valeur c_i, mais il faut considérer aussi son poids w_i). Pour trouver la solution optimale, il suffit d'afficher l'entrée $T[n, W]$.
Le temps d'exécution est clairement $O(n \cdot W)$ car la taille du tableau est $n \cdot W$ et on peut calculer chaque entrée en temps $O(1)$.

Remarque : Le graphe se compose d'un sommet pour chaque étiquette (i, x) pour tout $1 \leq i \leq n$ et $0 \leq x \leq W$. On a un arc de (i, x) à $(i+1, x)$ avec valeur 0 pour tout $1 \leq i \leq n-1$ et $0 \leq x \leq W$ et un arc de (i, x) à $(i+1, x+w_{i+1})$ avec valeur c_{i+1} pour tout $1 \leq i \leq n-1$ et $0 \leq x \leq W - w_{i+1}$.