Algorithmique TD5
L3 Informatique – ENS Cachan

15 octobre 2018

Exercice 1

On considère le problème d'ordonnancement des tâches suivant. On dispose de n tâches $a_1, \ldots, a_n$ dites unitaires (i.e. chaque tâche prend exactement une unité de temps à effectuer). À chaque tâche sont associées une date d'échéance $1 \leq d_i \leq n$, et une pénalité w_i qui survient si et seulement si la tâche a_i est exécutée à une date postérieure à d_i. On cherche à trouver un ordonnancement des tâches E (c'est à dire une permutation de la liste des tâches) qui minimise le total des pénalités.

1. On dit qu'une tâche est en retard si elle se termine après sa date d'échéance, et en avance dans le cas contraire. Montrer qu'on peut toujours mettre un ordonnancement sous forme en avance d'abord, i.e. ou toutes les tâches en avance précèdent les tâches en retard.
2. Montrer que tout ordonnancement sous forme en avance d'abord, peut se mettre sous une forme canonique ou les tâches en avance sont triées par ordre croissant de date d'échéance.

On dit qu'un ensemble de tâches est indépendant s'il existe un ordonnancement de ses tâches tel qu'aucune ne soit en retard. On note $\mathcal{I}$ l'ensemble de tous les ensembles indépendants de tâches.

Étant donné un ensemble F de tâches, pour $1 \leq t \leq n$, on note $N_t(F)$ le nombre de tâches de F dont la date d'échéance est inférieure à t.

3. Montrer l'équivalence des propositions suivantes :
   (a) F est indépendant
   (b) $\forall 1 \leq t \leq n, N_t(F) \leq t$
   (c) Si les tâches de F sont ordonnancées par ordre monotone croissant de dates d'échéance, alors aucune des tâches n'est en retard.
4. Montrer que $(E, \mathcal{I})$ est un matroïde
5. En déduire un algorithme permettant de trouver l'ordonnancement optimal d'un ensemble de tâches.

Exercice 2

On s'intéresse à l'opération de jointure sur les arbres rouge-noir. Elle prend en entrée deux arbres rouge-noir T_1 et T_2, et un enregistrement x tels que pour tout nœuds x_1 de T_1 et x_2 de T_2, $x_1.clé \leq x.clé \leq x_2.clé$. Elle retourne un arbre rouge-noir T correspondant à l'union de T_1, x, et T_2.

1. On décide de conserver la hauteur noire d'un arbre rouge-noir T dans un champ $T.hb$. Montrer que ce champ peut être mis à jour par les opérations d'insertion et de suppression sans utiliser d'espace supplémentaire dans les nœuds de l'arbre ni augmenter la complexité asymptotique.

On souhaite implémenter l'opération Jointure(T_1, x, T_2). On note n le nombre total de nœuds dans T_1 et T_2.

2. On suppose que $T_1.hb \geq T_2.hb$. Décrire un algorithme en temps $O(\log(n))$ qui trouve un nœud noir y dans T_1 ayant une clé maximale parmi tous les nœuds de hauteur noire $T_2.hb$.

3. Décrire un algorithme en temps $O(\log(n))$ qui réalise la jointure de T_1, x et T_2, en distinguant les cas $T_1.hb \geq T_2.hb$ et $T_1.hb < T_2.hb$.

Exercice 3

Le problème du voyageur de commerce euclidien consiste à déterminer une tournée optimale permettant de relier un ensemble donné $\{V_1, \ldots, V_n\}$ de points du plan, i.e., un circuit $V_{i_1}, \ldots, V_{i_n}, V_{i_{n+1}} = V_{i_1}$ avec $V_i \neq V_j$ pour $1 \leq i < j \leq n$, et tel que la somme des longeurs des segments $[V_{i_j}, V_{i_{j+1}}]$ soit minimale. C'est un problème NP-complet.

J.L. Bently a suggéré de simplifier le problème en se restreignant aux tournées *bitoniques*, autrement dit, celles qui partent du point le plus à gauche, continuent de gauche à droite jusqu'au point le plus à droite, puis retournent vers le point de départ en se déplaçant de droite à gauche.

On suppose que deux points n'ont jamais la même abscisse. Donner un algorithme en temps $O(n^2)$ permettant de déterminer une tournée bitonique optimale.

Exercice 4

On se propose d'étudier les arbres splay (« Splay trees »), qui sont des arbres binaires de recherche qui "s'autoéquilibrent", en ramenant les éléments fréquemment accédés près de la racine de l'arbre. Afin de faire cela on introduit une opération, appelée « splay », qui prend un arbre A et un nœud x, et fais remonter x pour qu'il devienne la racine de l'arbre. L'opération applique récursivement des transformations locales qui ont pour but de faire remonter x petit à petit.

1. Distinguer trois cas différents (modulo symétries), et proposer une transformation locale qui fait remonter x pour chacun de ces cas, en utilisant des rotations simples ou doubles.
2. Expliquer comment implémenter les opérations suivantes en utilisant l'opération de « splay » :
   - Insérer un élément (à la fin, le nouvel élément doit être à la racine de l'arbre)
   - Supprimer un élément
   - Fusionner deux arbres splay
   - Séparer un arbre splay en deux : étant donné un nœud x de l'arbre, il faut rendre un arbre qui contient les éléments (strictement) plus petits que x, et un autre qui contient les éléments (strictement) plus grand que x.
3. On veut maintenant analyser la complexité amortie de opérations ci-dessus. On définit pour cela les notions suivantes :
   - La taille d'un nœud x : $\text{size}(x)$ est le nombre de nœuds du sous arbre qui a r pour racine.
   - Le rang d'un nœud x : $\text{rank}(x) = \log_2(\text{size}(x))$

   Trouver une fonction de potentiel telle que l'opération de « splay » au nœud x ait un coût amorti inférieur à $3(\text{rank}(r) - \text{rank}(x)) + 1$, où r est la racine de l'arbre.
4. En déduire la complexité amortie d'une suite d'opérations quelconques.