

# Algorithmique TD5

## L3 Informatique – ENS Paris-Saclay

14 octobre 2020

### Exercice 1

On suppose qu'on a un système monétaire constitué de pièces de valeurs  $1, p, p^2, \dots, p^k$  pour  $p \geq 2$  et  $k \geq 0$ .

1. Démontrer que l'algorithme glouton qui fait l'appoint en prenant en priorité les pièces les plus grosses est optimal, i.e., qu'il minimise le nombre de pièces utilisées.
2. Donner un exemple de système monétaire où l'algorithme glouton n'est pas optimal.
3. Pour un système monétaire quelconque, montrer comment résoudre le problème en temps  $O(Mk)$  où  $M$  est le montant à payer et  $k$  est le nombre de pièce de monnaies.

### Exercice 2

On considère le problème d'ordonnement des tâches suivant. On dispose de  $n$  tâches  $a_1, \dots, a_n$  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 où 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 et calculer sa complexité.

### Exercice 3

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

J.L. Bentley 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 fait 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 des 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  $x$  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.