TD 07 – Corrigé du partiel

Exercice 1. *Stations d'essence*

Bob roule sur l'autoroute en voiture. Il part de Lyon (kilomètre 0) avec le plein d'essence et va à Tombouctou. La i-ème station-service est au kilomètre k_i. Son autonomie (plein d'essence) est de d kilomètres.

1. Quel est son algorithme pour minimiser le nombre d'arrêts pour faire le plein ?

Exercice 2. *Complexité*

On considère un ensemble S de $n \geq 2$ entiers distincts (S n'est pas supposé trié) :

1. Proposer un algorithme en $O(n)$ pour trouver deux éléments x et y de S tels que $|x-y| \geq |u-v|$ pour tout $u, v \in S$.
2. Proposer un algorithme en $O(nlogn)$ pour trouver deux éléments x et y de S tels que $x \neq y$ et $|x-y| \leq |u-v|$ pour tout $u, v \in S$, $u \neq v$.
3. Soit m un entier arbitraire (non nécessairement élément de S). Proposer un algorithme en $O(nlogn)$ pour déterminer s'il existe deux éléments x et y de S tels que $m = x + y$.
4. **(difficile)** Proposer un algorithme en $O(n)$ pour trouver deux éléments distincts x et y de S tels que $|x-y| \leq \frac{1}{n-1}(\max(S) - \min(S))$.

Exercice 3. *Analyse du cours d'actions*

On regarde le cours passé d'une action pendant n jours consécutifs et on se demande quel profit on aurait pu faire. Autrement dit, étant donnés n nombres $p_1, p_2, \ldots, p_n$ ou p_i est le cours de l'action au i-ème jour, on veut trouver $M = \max_{1 \leq i < j \leq n} (p_j - p_i)$ (acheter le jour i, revendre le jour j ultérieur), ou renvoyer 0 si $M < 0$ (pas de profit possible).

1. Donner un algorithme diviser-pour-régner de coût $O(n \log n)$.
2. Donner un algorithme de coût $O(n)$.

Exercice 4. *Plus grand palindrome*

On a une chaîne de caractères $s_1 s_2 \ldots s_n$ et on veut trouver la longueur de la plus longue sous-chaîne (un segment $s_i s_{i+1} \ldots s_j$ de caractère consécutifs) qui soit un palindrome (un palindrome est identique quand on le lit de gauche à droite et de droite à gauche). Par exemple la chaîne *pokalomomolaemon* a une sous-chaîne palindrome de longueur 9, *alomomola*.

1. Donner un algorithme de coût au plus $O(n^3)$.
2. Donner un algorithme de coût $O(n^2)$

Exercice 5. *Recherche d'un élément dans une matrice*

On a une matrice carrée $A = (a_{ij})$ d'ordre n. Les a_{ij} sont des entiers positifs. On suppose que toutes les lignes et toutes les colonnes de A sont croissantes (au sens large). On a enfin un entier x positif dont on veut savoir s'il existe dans la matrice.

1. Donner un algorithme de coût $O(n^2)$
2. Donner un algorithme de coût $O(n \log n)$
3. **(difficile)** Donner un algorithme de coût $O(n)$

Exercice 6. *Problème des cageots de fraises*

On s'intéresse (mais si !) à la distribution de n cageots de fraises dans p magasins. Le bénéfice que l'on peut retirer de chaque magasin est fonction du nombre de cageots fourni. Ainsi, $b_j(i)$ représente le bénéfice que tire le magasin j de la vente de i cageots ($1 \leq i \leq n$ et $1 \leq j \leq p$). On appele *gains marginaux* les gains supplémentaires obtenus par cageot :
$g_j(i) = b_j(i) - b_j(i-1)$, avec $b_j(0) = 0$ (aucun bénéfice pour la vente de 0 cageot).

1. Si, pour chaque magasin j, la fonction de gains marginaux g_j est décroissante, proposer un algorithme pour répartir les cageots de fraises dans les magasins avec bénéfice maximal. Donner un contre-exemple si les gains marginaux ne sont pas décroissants.
2. **(difficile)** Proposer un algorithme qui calcule le bénéfice optimal dans le cas général. Quelle est sa complexité ?

Exercice 7. *Plus longue coupe équilibrée*

Soit F un tableau unidimensionnel d'entiers de taille $n \geq 1$. On suppose que tous les éléments de F valent 0 ou 1. Une coupe $[i..j[$ de F, avec $0 \leq i < n$ et $i \leq j \leq n$, est dite équilibrée si elle comporte autant de 0 que de 1 :

$$\text{card}\{k/F[k] = 0,\ i \leq k < j\} = \text{card}\{k/F[k] = 1,\ i \leq k < j\}.$$

La longueur de la coupe $[i..j[$ est son nombre d'éléments $j - i$. **Le but de l'exercice est de déterminer la longueur de la plus longue coupe équilibré du tableau** F. On définit la classe

```
class Tableau {
  final static int n = 100 ; // définition d'une constante
  static int[] F = new int[n] ;

  ... méthodes (fonctions) à écrire ...
  ... }
```

On supposera que F a été initialisé par une méthode extérieure. Cet exercice est représentatif d'une large classe de problèmes traitant de parcours de tableaux : nous allons construire des solutions de plus en plus efficaces.

1. Quel serait l'ordre de grandeur $O(n^x)$ du coût de la méthode suivante, trop naïve pour que nous l'écrivions en Java : examiner toutes les coupes $[i..j[$ du tableau F et prendre le maximum des longueurs de celles qui sont équilibrées ?
2. Ecrire une fonction `static int equi1(int i)` qui renvoie la longueur de la plus longue coupe équilibrée du tableau F qui commence en i. On cherche donc

   $$\max_j\{j - i,\ i \leq j \leq n,\ [i..j[\text{ équilibrée}\}.$$

   On veut que le coût de cette fonction `equi1` soit en $O(n)$.
3. Faire appel à la fonction précédente `equi1` pour écrire une fonction `static int longueur1()` qui retourne la longueur de la plus longue coupe équilibrée de F. Quel est l'ordre de grandeur $O(n^x)$ du coût de cette fonction ?

On cherche maintenant une fonction `static int longueur2()` qui retourne la longueur de la plus longue coupe équilibrée de F, et dont le coût soit linéaire en n. Pour cela, on introduit le déséquilibre $\text{deseq}(i,j)$ d'une coupe $[i..j[$, i.e. la différence entre son nombre de 1 et son nombre de 0 :

$$\text{deseq}(i,j) = \text{card}\{k/F[k] = 1,\ i \leq k < j\} - \text{card}\{k/F[k] = 0,\ i \leq k < j\}.$$

Clairement, $\text{deseq}(i,j)$ est un entier compris entre $-n$ et n.
L'idée est de parcourir une fois le tableau F et de calculer, pour chaque indice i, $0 \leq i \leq n$, le déséquilibre $\text{deseq}(0,i)$. Plus précisément, on va remplir un tableau auxiliaire `aux` de taille $2n+1$, dont les éléments auront auparavant été initialisés à -1, de la façon suivante :

$$aux(d+n) = i \text{ ssi } i \text{ est le plus petit indice tel que } \text{deseq}(0,i) = d.$$

4. Ecrire une fonction static void prepareLongueur2() qui effectue ce travail (ne pas oublier que le déséquilibre 0 est obtenu pour une coupe vide : ainsi $aux(n) = 0$).

5. On remarque alors que si $\text{deseq}(0, i) = \text{deseq}(0, j)$ pour deux indices i et j avec $i \leq j$, alors la coupe $[i..j[$ est équilibrée. Expliquer comment modifier la fonction prepareLongueur2() pour obtenir la fonction longueur2() qui calcule la longueur de la plus longue coupe équilibrée de F avec un coût linéaire en n.