IFT2015 automne 2013 — Devoir 5

Miklós Csűrös

19 novembre 2013 — remise vendredi le 29 novembre à 20 :15

Remettez un rapport écrit par email (à csuros@iro. . .) en format PDF. La taille de votre message ne doit pas dépasser 2^{20} octets. Travaillez seul sur tous les problèmes. Le devoir vaut 50 points, avec 10 points boni possibles.

5.1 Tri bébé (10 points)

a. $n = 3$ **(5 points)** ► Quel est le minimum de comparaisons pour trier trois éléments ? Donner un algorithme pour trier trois éléments a, b, c avec ce nombre minimum de comparaisons.

b. $n = 4$ **(5 points)** ► Quel est le minimum de comparaisons pour trier quatre éléments ? Donner un algorithme pour trier quatre éléments a, b, c, d avec ce nombre minimum de comparaisons.

5.2 Vis-écrou (10+10 points)

On a un ensemble de n vis et n écrous. On sait que les vis sont de tailles différentes, et qu'il existe exactement un écrou pour chaque vis. On ne peut comparer qu'un écrou E à une vis V à la fois (donc pas de comparaisons écrou-écrou ou vis-vis), pour vérifier que soit $E < V$, soit $E = V$, soit $E > V$.

a. $n = 2$ **(5 points)** ► Donner un algorithme pour trier $n = 2$ paires de vis-écrou.

b. $n = 3$ **(5 points)** ► Donner un algorithme pour trier $n = 3$ paires de vis-écrou.

c. $n > 3$ **(10 points boni)** ▶ Quel est le nombre minimal de comparaisons vis-écrou qu'un algorithme de tri déterministe doit faire ?

REMARQUE. On peut trier les paires par la technique de tri rapide (choisir un vis, l'utiliser pour pivot sur tous les écrous, après utiliser l'écrou de même taille comme pivot sur les vis — voir Démonstration). Mais cela peut prendre $\Theta(n^2)$. Il est très difficile par contre de trouver un algorithme de tri déterministe pour ce problème avec $O(n \log n)$ temps de calcul. En fait, pour un tel algorithme (avec preuve de temps de calcul), vous aurez ∞ points boni :)

5.3 Permutations (30 points)

Supposons qu'on a un tableau $A[0..n-1]$ qui contient des éléments avec clés comparables et on veut les mettre dans l'ordre croissant. On utilise un tri pour *les indices* : l'algorithme nous donne la permutation des indices $\{0, 1, 2, \ldots, n-1\}$ qui correspond à l'ordre croissant. On représente une permutation sur n éléments par la vecteur $\pi = \pi[0..n-1]$ telle que pour tout $j = \{0, 1, \ldots, n-1\}$ il existe exactement un indice i avec $\pi[i] = j$. (On écrit $\pi = \begin{bmatrix} i_0 & i_1 & \ldots & i_{n-1} \end{bmatrix}$ pour dénoter $\forall j \colon \pi[j] = i_j$). La sortie du tri des indices est donc une permutation π pour laquelle

$$A\big[\pi[0]\big] \leq A\big[\pi[1]\big] \leq A\big[\pi[2]\big] \leq \cdots \leq A\big[\pi[n-1]\big].$$

Exemple : si $A = \begin{bmatrix} \texttt{C} & \texttt{A} & \texttt{B} & \texttt{D} \end{bmatrix}$, le tri donne la permutation $\pi = \begin{bmatrix} 1 & 2 & 0 & 3 \end{bmatrix}$. Avec une notation plus formelle,

$$\pi \cdot A = A\begin{bmatrix} \pi[0] & \pi[1] & \cdots & \pi[n-1] \end{bmatrix}.$$

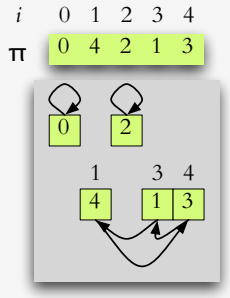
Par exemple, si $\pi = \begin{bmatrix} 0 & 4 & 2 & 1 & 3 \end{bmatrix}$,

$$\pi \cdot \begin{bmatrix} A & B & C & D & E \end{bmatrix} = \begin{bmatrix} A & E & C & B & D \end{bmatrix}$$

Il n'est pas difficile de réarranger les éléments de A selon l'ordre π à l'aide d'un tableau auxiliare :

```
SHUFFLE(A[0..n-1], π[0..n-1])        // réarranger le tableau A selon la permutation π
P1  initialiser tableau auxiliaire aux[0..n-1]
P2  for i ← 0, 1, ..., n-1 do j ← π[i] ; aux[i] ← A[j]
P3  for i ← 0, 1, ..., n-1 do A[i] ← aux[i]                    // copier dans A
```

Le but de cet exercice est de développer un algorithme pour faire le même réarrangement *en place*, sans utiliser un tableau auxiliaire. L'idée est de considérer la décomposition de la permutation en cycles.



Un *cycle* de la permutation est une suite maximale d'indices différents $i, \pi[i], \pi[\pi[i]], \pi[\pi[\pi[i]]], \ldots$ On peut décomposer chaque permutation uniquement en un ensemble de cycles. Par exemple, la permutation $\pi = \begin{bmatrix} 0 & 4 & 2 & 1 & 3 \end{bmatrix}$ comprend trois cycles : (0), $\begin{pmatrix} 1 & 4 & 3 \end{pmatrix}$ et (2).

a. Nombre de cycles. (15 points) ▶ Donnez un algorithme NUMCYCLES(π) qui compte le nombre de cycles dans une permutation fournie comme une tableau $\pi[0..n-1]$. L'algorithme doit prendre $O(n)$ temps. (L'algorithme a le droit de détruire le contenu de π si nécessaire.)

Indice: *On peut détecter un cycle qui contient i facilement : boucler sur i, $\pi[i]$, $\pi[\pi[i]], \ldots$ jusqu'à ce qu'on retombe sur i. Faites un parcours du tableau, en lançant des parcours de cycles mais marquez les cellules visités pour arriver à $O(n)$.*

b. Permutation d'un tableau (15 points) ▶ Donnez un algorithme SHUFFLE(A, π) pour calculer $\pi \cdot A$ *en place*. Les arguments π, A sont des tableaux de taille (connue) n. L'algorithme doit prendre $O(n)$ temps et utiliser $O(1)$ mémoire de travail à part de π et A. (L'algorithme a le droit de changer les éléments de π pendant son exécution.)

Indice: *Détectez les cycles comme en a., et décalez les éléments au long de chaque cycle $i, \pi[i], \pi[\pi[i]] \ldots,$.*