L'algorithme de Robinson, Schensted et Knuth

Option informatique
lycée Louis-le-Grand

vacances de la Toussaint 1997

Résumé

Nous considérons ici les relations qui apparaissent entre des permutations involutives et des tableaux de nombres, appelés tableaux de Young (ou de Ferrer) et qui possèdent diverses propriétés de monotonie. On implémente l'algorithme qui associe un couple de tableaux de Young à une permutation donnée, ainsi que l'algorithme réciproque, puis on montre que l'on obtient deux fois le même tableau dans le cas d'une involution.

1 Permutations

1.1 Rappels

Soit $n \geqslant 1$. Une permutation σ de S_n est une bijection de l'ensemble $\{0, 1, \ldots, n-1\}$ dans lui-même. On la représentera par la liste des images $\sigma(0)$, $\sigma(1)$, …, $\sigma(n-1)$. Une permutation σ est dite **involutive** si $\sigma \circ \sigma$ est l'application identique. Une **inversion** d'une permutation σ est un couple (i, j) d'indices tels que $\sigma(i) > \sigma(j)$ et pourtant $i < j$.

1.2 Représentation en Caml

On choisit de représenter les permutations par des listes d'entiers.

◇ Écrire une fonction `est_permutation : int list -> bool` qui dit si la liste argument représente ou non une permutation.
◇ Écrire la fonction `compose : int list -> int list -> int list`.
◇ Écrire alors la fonction `involutive : int list -> bool` et la fonction `inverse : int list -> int list` qui calcule la bijection réciproque.

1.3 Recherche des inversions

◇ Écrire la fonction `est_inversion : int list -> int -> int -> bool` telle que `inversion sigma i j` est vraie si et seulement si (i, j) est une inversion de σ (et donc en particulier si $i < j$).
◇ Écrire la fonction `nb_inversions : int list -> int` qui calcule le nombre d'inversions d'une permutation, et une autre fonction `liste_inversions : int list -> (int * int) list` qui renvoie la liste de ces inversions.

2 Tableaux de Young

2.1 Définitions

Soit $n \geqslant 1$, $p \geqslant 1$ et n_1, n_2, …, n_p des entiers naturels non nuls de somme n.
On suppose que $n_1 \geqslant n_2 \geqslant \cdots \geqslant n_p$.
Un tableau de Young de forme $(n_1, n_2, \ldots, n_p)$ est un tableau de cases alignées sur la gauche, organisées en p lignes, la k-ième comportant n_k cases, et remplies par des entiers de telle sorte que chaque ligne de cases est en ordre croissant de gauche à droite, et chaque colonne de haut en bas.

Nécessairement, l'entier le plus petit sera dans la case le plus en haut à gauche, et l'entier le plus grand au bout d'une des lignes du tableau.
Par exemple, la figure 1 montre un tableau de Young de taille 15, de forme $(6, 4, 4, 1)$, qui possède 5 lignes, où on a rangé les entiers de 0 à 14.

0	1	4	8	9	14
2	5	6	12		
3	7	11	13		
10					

Figure 1: un premier exemple de tableau de Young

◇ Dessiner tous les tableaux de taille 4 contenant les entiers de 0 à 3. Combien y en a-t-il ? Vérifier que c'est aussi le nombres d'involutions de S_4. Le résultat est général, comme nous allons le voir.

On représentera en Caml un tableau de Young par une liste de liste d'entiers : une liste par ligne, de haut en bas. On définit donc `type young == int list list`.

◇ Écrire les fonctions `taille : young -> int` et `forme : young -> int list`.

2.2 Insertion dans un tableau de Young

Voici, sur l'exemple du tableau de la figure 2, la description d'un algorithme d'insertion d'un nouvel élément — ici le nombre 8 — dans un tableau de Young.

1	3	5	9	12	16
2	6	10	15		
4	13	14			
11					
17					

Figure 2: l'exemple support de l'algorithme d'insertion

L'algorithme d'insertion sur un exemple. Plaçant 8 en première ligne, sa place est celle qu'occupe actuellement l'entier 9. Je dépose donc 8 à sa place, et m'apprête à placer l'entier 9 à sa place dans la deuxième ligne : on dira que j'ai *bousculé* 9.
Sa place est celle qu'occupe actuellement l'entier 10. Je dépose donc 9 à sa place, et m'apprête à placer l'entier 10 à sa place dans la troisième ligne : j'ai *bousculé* 10.
Sa place est celle qu'occupe actuellement l'entier 13. Je dépose donc 10 à sa place, et m'apprête à placer l'entier 13 à sa place dans la quatrième ligne : j'ai *bousculé* 13.
Sa place n'est pas encore occupée, puisqu'il s'agit du bout de ligne… J'ai fini.

◇ À l'aide de cette définition (très informelle), écrire la fonction `insertion : young -> int -> young` correspondante.

2.3 L'algorithme inverse : suppression d'un élément d'un tableau de Young

Décrire, de la même façon informelle qui a été utilisée ci-dessus, et sur l'exemple du tableau de Young qu'on a obtenu par l'insertion de l'entier 8 qu'on dessinera, l'algorithme inverse : il s'agit, en détruisant l'entier 8, de retrouver le tableau de Young initial.

◇ Ensuite, écrire la fonction Caml correspondante `suppression : young -> int -> young`.

3 L'algorithme RSK

3.1 L'algorithme direct

On considère une matrice à coefficients entiers, à deux lignes et r colonnes $\begin{pmatrix} q_1 & q_2 & \cdots & q_r \\ p_1 & p_2 & \cdots & p_r \end{pmatrix}$ vérifiant $q_1 < q_2 < \cdots < q_r$. L'algorithme de Robinson, Schensted et Knuth lui associe un couple (P, Q) de tableaux de Young de la façon suivante. On commence par deux tableaux vides P et Q, puis, pour i variant, dans cet ordre, de 1 à r, on insère dans P l'entier p_i : on aura créé une nouvelle case dans le tableau P (ce n'est pas forcément là que se trouvera p_i, cependant), on crée une nouvelle case au même endroit dans le tableau Q, où on écrit q_i. De cette manière, P et Q ont toujours la même forme.

Par exemple, on obtient les tableaux de la figure 3 à partir de la matrice $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$.

	P	Q
étape 1	7	1
étape 2	2 7	1 3
étape 3	2 9 7	1 5 3
étape 4	2 5 7 9	1 5 3 6
étape finale	2 3 5 9 7	1 5 3 6 8

Figure 3: les étapes successives de l'algorithme RSK sur un exemple

On aura compris que l'algorithme d'insertion que nous avons décrit précédemment n'est utilisé que dans la construction de P, pas dans celle de Q.

◇ Montrer néanmoins que le tableau Q obtenu est bien un tableau de Young.

◇ Écrire la fonction `RSK : int list -> int list -> young * young` qui prend en argument les deux listes $[q_1; \ldots; q_r]$ et $[p_1; \ldots; p_r]$ et qui renvoie le couple (P, Q).

3.2 L'algorithme inverse

◇ Écrire la fonction `RSK_inverse : young * young -> int list * int list` qui renvoie le couple de listes $([q_1; \ldots; q_r], [p_1; \ldots; p_r])$ pour un argument égal au couple (P, Q) de tableaux de Young.

4 RSK et permutations

4.1 Minima à gauche

Étant donnée une liste $[a_1; \ldots; a_r]$ d'entiers on appelle *minima à gauche* ceux des a_k qui vérifient la relation :

$$a_k = \min_{1 \leqslant i \leqslant k} a_i.$$

Par exemple, dans le cas de la liste $[7; 3; 6; 4; 5; 1; 2]$, les minima à gauche forment la liste $[7; 3; 1]$.

◇ Écrire une fonction `minima_à_gauche : int list -> int list` qui renvoie cette liste des minima à gauche.

4.2 Classification

Reprenons l'exemple de la matrice $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$.

Les minima à gauches de la liste p sont 7 et 2 : on dira que les couples (q, p) correspondants, c'est-à-dire ici $(1, 7)$ et $(3, 2)$ sont de classe 1. Reste la matrice $\begin{pmatrix} 5 & 6 & 8 \\ 9 & 5 & 3 \end{pmatrix}$. Cette fois tous les éléments sont minima à gauche, et les couples $(5, 9)$, $(6, 5)$ et $(8, 3)$ seront de classe 2.

On vérifiera que sur le nouvel exemple de la matrice $\begin{pmatrix} 1 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 7 & 1 & 4 & 9 & 3 & 5 \end{pmatrix}$, les couples de classe 1 sont $(1, 2)$ et $(4, 1)$; les couples de classe 2 sont $(3, 7)$, $(5, 4)$ et $(7, 3)$; enfin, les couples de classe 3 sont $(6, 9)$ et $(8, 5)$.

◇ Écrire une fonction `classification : int list -> int list -> (int * int) list list` qui, à partir des deux listes $[q_1; \ldots; q_r]$ et $[p_1; \ldots; p_r]$ calcule la liste des classes, chaque classe successive étant représentée par une liste de couples (p, q).

Remarquons au passage que les minima à gauche d'une liste forment toujours une liste décroissante d'entiers.

Reprenons le problème de l'algorithme RSK, et partons d'une matrice $\begin{pmatrix} q_1 & \cdots & q_r \\ p_1 & \cdots & p_r \end{pmatrix}$.

Supposons que les couples d'une même classe s'écrivent (q_{i_1}, p_{i_1}), …, (q_{i_k}, p_{i_k}) avec $q_{i_1} < \cdots < q_{i_k}$ et $p_{i_1} > \cdots > p_{i_k}$. On vérifie alors que p_{i_k} et q_{i_1} sont les entiers qui sont écrits dans la première ligne de P et de Q en colonne c, où c est le numéro de la catégorie considérée. Les lignes suivantes sont produites — du moins pour cette classe — par les couples de la matrice $\begin{pmatrix} q_{i_2} & q_{i_3} & \cdots & q_{i_k} \\ p_{i_1} & p_{i_2} & \cdots & p_{i_{k-1}} \end{pmatrix}$.

Avec l'exemple de la figure 3 page précédente, on avait trouvé les couples $(1, 7)$ et $(3, 2)$ de classe 1, et $(5, 9)$, $(6, 5)$, $(8, 3)$ de classe 2. On en déduit que la première ligne de P est composée de 2 et 3, et la première ligne de Q de 1 et 5.

Les lignes suivantes sont produites par la matrice $\begin{pmatrix} 3 & 6 & 8 \\ 7 & 9 & 5 \end{pmatrix}$.

◇ Déduire de l'étude précédente une nouvelle façon d'écrire la fonction `RSK`.

4.3 Pour terminer…

◇ Montrer que si l'algorithme RSK associe à la matrice $\begin{pmatrix} 0 & 1 & \ldots & n-1 \\ \sigma(0) & \sigma(1) & \ldots & \sigma(n-1) \end{pmatrix}$ d'une permutation σ le couple (P, Q) de tableaux, il associe le couple (Q, P) à la permutation réciproque.

◇ En déduire que les involutions sont caractérisées par $P = Q$, et donc qu'il y a autant d'involutions dans S_n que de tableaux de Young contenant les entiers de 0 à $n - 1$.