DS1 d'informatique MPSI
17 février 2012

Devoir surveillé

Durée : 2 heures

Aucun document n'est autorisé. **On pourra définir des fonctions non demandées explicitement, si cela facilite la programmation. Inutile de prouver la correction et la terminaison des fonctions écrites, sauf si on le demande explicitement. Les fautes de syntaxe seront sanctionnées.**

Sur les permutations

Une *permutation* est une bijection d'un ensemble E dans lui-même. Pour tout $n \in \mathbb{N}^*$, on considère l'ensemble $E_n = [\![0, n-1]\!]$, de cardinal n. Une permutation φ de E_n sera représentée en Caml par un vecteur v de taille n, tel que $\varphi(k) = v.(k)$ pour tout $k \in [\![0, n-1]\!]$. Par exemple, `[|2; 1; 0; 5; 3; 4|]` représente la permutation φ_0 de E_6 donnée par $\varphi_0(0) = 2$, $\varphi_0(1) = 1$, $\varphi_0(2) = 0$, $\varphi_0(3) = 5$, $\varphi_0(4) = 3$ et $\varphi_0(5) = 4$.

On notera qu'un vecteur v de taille n représentera une permutation de E_n si et seulement si les contenus des cases de v sont les éléments de E_n.

Sauf à la question C.7, on supposera sans avoir à le vérifier que les vecteurs donnés en argument aux fonctions à écrire représenteront bien des permutations (sur E_n). Dans cet esprit, on confond par la suite les permutations et les vecteurs d'entiers qui les représentent en machine. Plus généralement, si l'énoncé contraint les arguments des fonctions à écrire, le code de ces fonctions sera écrit en supposant que ces contraintes sont satisfaites.

Partie A – Ordre d'une permutation

A.1 Écrire une fonction composer de type `int vect -> int vect -> int vect` qui prend en argument des permutations t et u sur E_n, et renvoie la permutation $t \circ u$.

A.2 Écrire une fonction inverser de type `int vect -> int vect` qui, à une permutation t sur E_n, associe sa réciproque t^{-1}.

La notation t^k désigne t composée k fois. On définit l'*ordre* d'une permutation t comme le plus petit entier naturel non nul k tel que $t^k = \mathrm{Id}_{E_n}$. Par exemple, φ_0 est d'ordre 6.

A.3 Donner un exemple de permutation (de E_n) d'ordre 1 et de permutation (de E_n) d'ordre n.

A.4 En utilisant la fonction composer, écrire une fonction ordre de type `int vect -> int`, associant à une permutation sur E_n son ordre. Un bonus sera attribué en cas de programmation récursive terminale (pour une fonction auxiliaire).

Partie B – Manipuler les permutations

La *période* d'un indice $i \in E_n$ pour la permutation t est le plus petit $k \in \mathbb{N}^*$ tel que $t^k(i) = i$. Pour φ_0, les périodes respectives de $0, 1, 2, 3, 4, 5$ sont $2, 1, 2, 3, 3, 3$.

B.1 Écrire une fonction periode de type `int vect -> int -> int` qui, à une permutation t et un indice i, associe la période de i pour t.

L'*orbite* de i pour la permutation t est l'ensemble des indixes j tels qu'il existe k pour lequel $j = t^k(i)$. Il est facile de montrer que la relation « être dans l'orbite de » pour t est une relation d'équivalence (*i.e.* réflexive, symétrique, transitive). Pour φ_0, les orbites sont $\{0, 2\}, \{1\}, \{3, 4, 5\}$.

On peut observer que la période de i pour t est le cardinal de son orbite.

B.2 Écrire une fonction estDansOrbite de type `int vect -> int -> int -> bool` telle que `estDansOrbite t i j` soit vrai si j est dans l'orbite de i pour t et faux sinon.

Une *transposition* est une permutation qui échange deux éléments *distincts* et qui laisse les autres inchangés.

B.3 Écrire une fonction estTransposition de type `int vect -> bool` qui détermine si une permutation est une transposition.

Un *cycle* est une permutation dont exactement une orbite est de taille strictement supérieure à un. Toutes les autres orbites, s'il y en a, sont réduites à des singletons.

B.4 Écrire une fonction estCycle de type `int vect -> bool` qui détermine si une permutation est un cycle.

Partie C – Opérations efficaces sur les permutations

C.1 Écrire une fonction periodes de type int vect -> int vect, qui renvoie le tableau p des périodes. C'est-à-dire que $p.(i)$ est la période de l'indice i pour la permutation t. Un bonus sera attribué si le nombre d'appel à la fonction periode est inférieur ou égal au nombre d'orbites pour t.

On envisage ensuite le calcul efficace de l'itérée t^k. On remarque en effet que $t^k.(i)$ est égal à $t^r.(i)$, où r est le reste de la division euclidienne de k par la periode de i pour t.

C.2 Écrire une fonction itererEfficace de type int vect -> int -> int vect, qui calcule l'itérée t^k en utilisant le tableau des périodes. On rappelle que t^0 est l'identité.

La fonction ordre n'est pas très efficace. En effet, elle procède à de l'ordre de o compositions de permutations, où o est l'ordre de la permutation passée en argument. Or o est de l'ordre de $n^{\sqrt{n}}$ dans le cas le pire. On peut améliorer considérablement le calcul de l'ordre en constatant que l'ordre d'une permutation est le plus petit commun multiple des périodes des éléments.

C.3 Donner un exemple de permutation dont l'ordre excède strictement la taille.

C.4 Écrire une fonction pgcd, de type int -> int -> int, qui prend en arguments deux entiers naturels a et b, et renvoie le plus grand diviseur commun de a et b. On impose un calcul récursif selon l'algorithme d'Euclide qui repose sur l'identité $pgcd(a, b) = pgcd(b, r)$, où r est le reste de la division euclidienne de a par b (lorsque $b \neq 0$). On rappelle aussi que $pgcd(a, 0) = a$.

C.5 Écrire une fonction ppcm, de type int -> int -> int, qui prend en arguments deux entiers strictement positifs a et b, et renvoie le plus petit commun multiple de a et b. On utilisera l'identité

$$ppcm(a, b) = \frac{ab}{pgcd(a, b)}.$$

C.6 Écrire une fonction ordreEfficace, de type int vect -> int, qui calcule l'ordre de la permutation t selon la méthode efficace. On cherchera à minimiser le nombre d'appels à ppcm effectués.

C.7 Écrire une fonction estPermutation qui prend une application (représentée par un tableau d'entiers t) en argument et vérifie que t représente bien une permutation. Autrement dit, la fonction renvoie true si t représente une permutation et false sinon.