Option Informatique

TD $n°02$

Problème 1 : Manipulation de matrices et nombres de Fibonacci

Dans ce problème on implémente les matrices d'entiers par des tableaux de tableaux, par exemple la matrice $2 \times 3 \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}$ par le tableau `[|[|a;b;c|];[|d;e;f|]|]`.

Q1. Expliquer comment on peut obtenir la matrice nulle 2×3 avec la fonction `Array.make_matrix`.

Q2. Écrire une fonction `modifier_matrice : int array array → int → int → int → unit` telle que `modifier_matrice m i j v` qui ne renvoie rien mais modifie $m_{i,j}$ en lui affectant la valeur v.

Expliquer pourquoi il est **indispensable** d'avoir utilisé `Array.make_matrix` pour créer la matrice `m` initiale et il aurait été **catastrophique** de la créer en emboitant des `Array.make` (faites le test).

Q3. Écrire une fonction `taille : int array array → int*int` telle que si m est une matrice de taille $n \times p$, l'appel `taille m` retourne le couple `(n,p)`.

Q4. Écrire une fonction `produit : int array array → int array array → int array array` telle que `produit a b` renvoie la matrice `a×b`.

En supposant que a est de taille $n \times p$ et b de taille $p \times q$, combien d'additions et combien de multiplications un tel appel effectue-t-il ? Et dans le cas de matrices carrées ?
Remarque culturelle : l'algorithme de Strassen utilise moins de multiplications (pour de grandes matrices).

Bonus (très facultatif) : Écrire une version sans boucle, en utilisant à la place des fonctions auxiliaires récursives.

Q5. Écrire une fonction récursive `puissance : int array array → int → int array array` telle que `puissance m n` renvoie m^n, calculé à l'aide de l'algorithme d'exponentiation rapide.

Justifier que, pour des matrices $p \times p$, cette fonction effectue $O(p^3 \lg(n))$ calculs d'additions et de multiplications d'entiers.

Quelle est la complexité spatiale ? Attention, il faut tenir compte de la pile d'appel récursifs. On pourra considérer le pire des cas où l'exposant est de la forme $n = 2^k - 1$ avec $k \in \mathbb{N}$.

Bonus (facultatif) : Proposer une nouvelle version avec une complexité spatiale ne dépendant pas de n.

Q6. Exprimer $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n$ en fonction des nombres de Fibonacci.

En déduire une fonction `fibo : int → int` qui calcule le $n^{\text{ième}}$ nombre de Fibonacci F_n en effectuant $O(\lg(n))$ calculs d'additions et de multiplications d'entiers.

Q7. Soit $\phi = \frac{1+\sqrt{5}}{2}$ le nombre d'or. On **admet** le résultat de maths suivant : $F_n = \frac{1}{\sqrt{5}}\left(\phi^n - \left(-\frac{1}{\phi}\right)^n\right)$.
Cette formule permet-elle de calculer F_n avec une complexité meilleure qu'à la question précédente ? Justifier.

Problème 2 : répartition dans une liste

Étant donné un entier m, le but de ce problème est de tester si un entier de $[\![0, m-1]\!]$ appartient ou pas à une liste d'entiers, et, le cas échéant, de déterminer combien de fois il y apparaît.

Précisément : on veut écrire une fonction `repartition : int → int list → (int*int) list` telle que `repartition m l` donne la liste de tous les couples `(a,k)` pour lesquels `a` est un entier strictement inférieur à `m` apparaissant dans `l` et `k` le nombre de fois que `a` apparaît dans `l`.

On propose trois manières différentes de le faire et on compare leurs complexités en temps et en espace.

Par recherche exhaustive

Q1. Écrire une fonction `frequence : int → int list → int` telle que `frequence a l` retourne le nombre d'apparitions (éventuellement nul) de l'entier `a` dans la liste `l`. Justifier que votre fonction est de complexité linéaire en la longueur `n` de la liste `l` (cela nécessite qu'elle le soit).

Q2. En déduire une fonction `repartition1 : int → int list → (int*int) list` qui répond à la question posée. Justifier que votre fonction est de complexité quadratique en la longueur `n` de la liste `l` (cela nécessite qu'elle le soit).

Avec un tableau de comptage

Q3. On définit une fonction `fonction1` par le code suivant :

```
let fonction1 (m:int) (l:int list) =
        let theta = Array.make m 0 in
        let aux l = match l with
                | [] -> theta
                | h::t -> let theta = aux t in
                                theta.(u) <- theta.(u)+1;
                                theta;;
```

Inférer le type de la fonction `fonction1`.
Expliquer ce que calcule `fonction1 m l` et le démontrer par récurrence sur la longueur n de l.

Q4. Écrire une fonction `repartition2 : texte → repartition` qui renvoie une répartition de l'alphabet dans un texte en utilisant la fonction `fonction1` définie à la question 1. Déterminer sa complexité en temps en fonction de `m` et de la longueur `n` de la liste donnée en argument. Quel est par contre l'inconvénient par rapport à la solution de la partie précédente ?

Avec une table modulaire

Soit $p \in \mathbb{N}^*$. Une *table modulaire de comptage d'ordre p d'une liste ℓ* est un tableau de longueur p dont la $i^{\text{ième}}$ case contient la liste de tous les couples (a, k) avec pour lesquels `a` est un entier congru à i modulo p dans ℓ et `k` le nombre de fois que a apparaît dans ℓ.

Par exemple, avec $p = 3$, une table modulaire de comptage d'ordre 3 de la liste `[0;2;0;3;6]` est le tableau de trois listes suivant : `[| [(0,2);(2,1);(6,1)] ; [] ; [(2,1)] |]`.

Q5. Écrire une fonction `incremente_table : (int*int) array → int → (int*int) list array` telle que `incremente_table t a` modifie la table modulaire de comptage `t` en incrémentant le nombre associé à `a` (si `a` ne figure pas dans la table, il faut le rajouter avec la valeur 1).

Q6. Écrire une fonction `repartition_modulaire : int list → int → (int*int) list array`, qui utilise la fonction définie à la question précédente, telle que la valeur de retour de `repartition_modulaire t m` est une table modulaire de comptage d'ordre m du texte t.