DS1 d'informatique MPSI
18 mars 2011

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.**

Exercice 1 : Une preuve de Cauchy de l'inégalité entre moyennes arithmétique et géométrique

Soit $n \in \mathbb{N}^*$, $(x_1, \dots, x_n) \in (\mathbb{R}_+^*)^n$. On appelle *moyenne arithmétique* (resp. *géométrique*) de $(x_1, \dots, x_n)$ le réel $a = \frac{x_1+\dots+x_n}{n}$ (resp. $g = \sqrt[n]{x_1 \dots x_n}$). On peut montrer que $g \leqslant a$.

Pour tout $n \in \mathbb{N}^*$, notons $\mathcal{H}_n$ l'hypothèse : on a montré cette inégalité pour tout $(a_1, \dots, a_n) \in (\mathbb{R}_+^*)^n$.

Une jolie preuve de Cauchy consiste à montrer, outre le fait que $\mathcal{H}_1$ est trivialement vérifié, que

$$\forall\, n \in \mathbb{N} \setminus \{0, 1\}, \mathcal{H}_n \Rightarrow \mathcal{H}_{n-1},$$

et

$$\forall\, n \in \mathbb{N}^*, \mathcal{H}_n \Rightarrow \mathcal{H}_{2n}.$$

On ne demande aucunement de montrer cette inégalité entre moyennes, que ce soit par la preuve de Cauchy ou autrement.

1 La preuve de Cauchy utilise quelle définition inductive de $\mathbb{N}^*$ dans lui-même ? On ne demande pas de prouver que l'on définit bien $\mathbb{N}^*$ inductivement ainsi.

2 Proposer une fonction cauchy : int -> bool permettant de vérifier que la preuve de Cauchy fonctionne pour tout entier naturel non nul n préalablement fixé, et prouver sa terminaison.

Remarque : cette fonction cauchy doit renvoyer true pour chaque valeur strictement positive de son argument, n'a rien à voir avec les différentes moyennes (celles-ci ne doivent pas figurer dans votre programme), seulement avec l'idée astucieuse de Cauchy, et elle ne suit pas nécessairement la définition inductive donnée à la question précédente.

Exercice 2 : Programmation sur les listes

Dans cet exercice, il est interdit d'utiliser des fonctions sur les listes telles que @ ou mem, ou même hd et tl. On a en revanche le droit d'utiliser les fonctions définies au cours de l'exercice, ainsi bien sûr que le conse « : : », [] et [a] (qui peut d'ailleurs s'écrire a :: []). On ne se posera pas de problème de complexité.

1 Réécrire les fonctions tête et queue pour les listes.

2 Réécrire la concaténation des listes en une fonction conc : 'a list -> 'a list -> 'a list.

3 Écrire une fonction membre : 'a -> 'a list -> bool telle que membre elt liste teste l'appartenance de elt à liste .

4 Écrire une fonction tous_distincts : 'a list -> bool qui renvoie true si et seulement si les termes de la liste en argument sont distincts deux à deux. On notera Ω l'ensemble des listes dont les termes sont distincts deux à deux.

5 Écrire une fonction tourne : 'a list -> 'a list qui envoie une liste [a0; ... ; a_n] sur [a1; ... ; an; a0].

6 Étant donné deux éléments l et l' de Ω, nous dirons que l' est une *permutée circulaire* de l si, à la suite d'un nombre fini d'applications de tourne, on obtient l' à partir de l. On définit bien sûr ainsi une relation d'équivalence sur les listes de termes distincts deux à deux.

a Écrire une fonction place_en_tete : 'a -> 'a list -> 'a list telle que place_en_tete elt l envoie la permutée circulaire de l commençant par elt si elt se trouve dans l, renvoie une erreur sinon.

b Écrire une fonction permutee_circulaire : 'a list -> 'a list -> bool testant si deux éléments de Ω sont des permutées circulaires l'une de l'autre.

c Deux éléments de Ω sont dits *permutés* l'une de l'autre si elles sont égales à permutation (pas nécessairement circulaire) près.

Par exemple, [1; 2; 3; 4; 5] et [1; 3; 2; 5; 4] sont des permutées, mais pas des permutées circulaires l'une de l'autre.

Proposer une fonction permutee : 'a list -> 'a list -> bool testant si deux éléments de Ω sont des permutées l'une de l'autre.

Exercice 3 : Parcours en escalier dans un tableau

Soit $n \in \mathbb{N}^*$, M_n la matrice de taille $(n+1)\times(n+1)$ telle que le coefficient en position (i,j) soit $m_{i,j} = 2^i 3^j$, pour tout $(i,j) \in [\![0,n]\!]^2$. Notez que pour se simplifier la vie, nous avons indexé lignes et colonnes par $[\![0,n]\!]$, et non $[\![1,n+1]\!]$.

Nous utiliserons la fonction prédéfinie make_matrix de type int -> int -> 'a -> 'a vect vect, telle que make_matrix n p a soit la matrice de taille (n,p), dont tous les termes valent a. Par exemple :

```
# make_matrix 2 2 0;;
- : int vect vect = [|[|0; 0|]; [|0; 0|]|]
```

Pour accéder au terme en position (i,j) d'une matrice t, on écrira t.(i).(j) :

```
# let exemple = [|[|1; 2; 3|]; [|4; 5; 6|]|];;
exemple : int vect vect = [|[|1; 2; 3|]; [|4; 5; 6|]|]

# exemple.(0).(1);;
- : int = 2

# exemple.(1).(2);;
- : int = 6
```

En poursuivant cet exemple, on peut facilement obtenir le nombre de lignes et de colonnes d'une matrice :

```
(* nombre de lignes d'une matrice *)

# vect_length exemple;;
- : int = 2

(* Nombre de colonnes d'une matrice *)

# vect_length exemple.(0);;
- : int = 3
```

1 Écrire une fonction init : int -> int vect vect qui à un entier naturel non nul n associe la matrice M_n. Un **bonus** sera attribué si cette fonction effectue moins de $(n+1)^2$ multiplications pour l'argument n (on indiquera d'ailleurs ce nombre de multiplications).

2 On considère $p \in [\![1, 2^n 3^n]\!]$, et on cherche le plus grand des minorants de p de la forme $2^i 3^j$.

On observe (inutile de le prouver) que si sur la ligne i_0 le plus grand des minorants de p se trouve en colonne j_0, alors sur la ligne i_0+1, le plus grand des minorants de p, *s'il existe*, est en position j_0 ou $j_0 - 1$.

Proposer un algorithme plus_grand_minorant : int -> int -> int qui envoie n p sur le plus grand des minorants de p parmi les éléments de M_n.