Devoir sur table
Décembre 2020

Documents autorisés: poly et notes de cours, notes de TD

L'épreuve durera 1 heure. Le sujet vaut 20 points plus 2 points de bonus.

Exercice I : Suites numériques

Q1[2pts] – On définit la suite u de paramètres c, t, m de la manière suivante:

$$\begin{cases} u_0 & = & c \\ u_{n+1} & = & (1+t)u_n - m \end{cases}$$

Définir la fonction

`u (n:int) (c:float) (t:float) (m:float) : float`

qui donne le `n`-ième terme de la suite u_n.

Exercice II : Distance de Hamming

La *distance de Hamming* donne une mesure de la différence entre deux séquences de *bits*. C'est le nombre de positions où les deux séquences diffèrent. On l'utilise en transmission réseau pour des codes correcteurs.

Q1[2pts] – Définir la fonction

`nbtrue (bs:bool list) : int`

qui donne le nombre de fois où `bs` contient la valeur `true`.

Bonus: +1pt pour une définition récursive terminale ou l'utilisation de `List.fold_left`.

Q2[2pts] – Définir la fonction

`sumb (xs:'a list) (ys:'a list) : bool list`

qui donne la liste des booléens qui contient `true` à chaque position où `xs` et `ys` diffèrent et `false` à toutes les autres positions. De plus, la fonction déclenche l'exception `Invalid_argument "sumb"` si les deux listes n'ont pas la même longueur.

Par exemple:
`(sumb [0;0;1;0;1;0] [0;1;1;0;0;1])` vaut `[false; true; false; false; true; true]`
`(sumb [] [0;0])` donne `Invalid_argument "sumb"`

Malus: -1pt si vous utilisez la fonction `List.length`.

Q3[1pts] – Déduire des deux questions précédentes la définition de la fonction

`disth (xs:'a list) (ys:'a list) : int`

qui donne la distance de Hamming entre `xs` et `ys`.

tourner la page

Exercice III : Tri rapide

Le principe de l'algorithme de tri dit *rapide* appliqué à une liste `xs` est le suivant:

- si `xs` est vide, le résultat est `[]`
- si `xs = x::xs'` alors le résultat est la liste `(xs1 @ [x] @ xs2)` où `xs1` est le résultat du tri des éléments de `xs'` inférieurs ou égaux à `x` et `xs2` est le résultat du tri des éléments de `xs'` strictement supérieurs à `x`

On va utiliser cette méthode pour trier des listes.

Q1[2pts] – Définir les fonctions

```
list_le (x:'a) (xs:'a list) : 'a list
list_gt (x:'a) (xs:'a list) : 'a list
```

telles que `(le x xs)` donne la liste des éléments de `xs` inférieurs ou égaux à `x` et `(gt x xs)` donne la liste des éléments de `xs` strictement supérieurs à `x`

Bonus: +1pt si vous utilisez l'itérateur `List.filter`

Q2[3pts] – Définir la fonction

```
qsort (xs:'a list) : 'a list
```

qui donne la liste triée des éléments de `xs` en utilisant la méthode du tri rapide.

Exercice IV : Différences dans une liste

Dans toutes les questions de cet exercice, utilisez la fonction `List.mem`.

Q1[2pts] – Définir la fonction

```
all_diff (xs:'a list) : bool
```

qui donne `true` si tous les éléments de `xs` sont différents entre eux.
Par convention, `(all_diff [])`=`true`.

Q2[2pts] – Définir la fonction

```
nb_diff (xs:'a list) : int
```

qui donne le nombre d'éléments différents entre eux de `xs`.

Par exemple:
`(nb_diff [])` vaut `0`
`(nb_diff [42; 42; 42])` vaut `1`
`(nb_diff [42; 54; 42; 42; 54])` vaut `2`

Q3[3pts] – Pour savoir qu'une liste contient au moins `n` valeurs différentes entre elles il n'est pas efficace d'utiliser la fonction `nb_diff`. En effet, on peut définir directement la fonction `at_least_n_diff` telle que:

`(at_least_n_diff 0 xs)`	=	`true`	
`(at_least_n_diff n [])`	=	`false`	si `n` $\neq 0$
`(at_least_n_diff n (x::xs))`	=	`(at_least_n_diff n xs)`	si `n` $\neq 0$ et `x` est dans `xs`
`(at_least_n_diff n (x::xs))`	=	`(at_least_n_diff (n-1) xs)`	si `n` $\neq 0$ et `x` n'est pas dans `xs`

Définir la fonction

```
at_least_n_diff (n:int) (xs:'a list) : bool
```

en OCaml.

Q4[1pts] – Si `n` < 0, l'application `(at_least_n_diff n xs)` a-t-elle une valeur ? Si oui, laquelle; sinon expliquez pourquoi.