Parcial Llenguatges de Programació

Grau en Enginyeria Informàtica

23 Novembre 2012

Per accedir al racó aneu a `https://examens.fib.upc.es`
Cal que lliureu via racó el codi amb els comentaris que considereu necessaris en un arxiu ".hs" executable en l'entorn ghci i que satisfaci els diferents enunciats que es llisten a continuació.

Cal que al començar la solució de cada problema afegiu una línia comentada indicant el problema (i subapartat, si en té) que ve a continuació. Per exemple,

```
-- Problema 1
-- Problema 5.a
```

Es valorarà l'ús que es faci de funcions d'ordre superior predefinides. Ara bé, en principi només s'han d'usar les de l'entorn Prelude, és a dir no hauria de caldre cap import. Si voleu usar una funció que requereixi import consulteu-ho abans amb el professor.

Excepte en el problema 1 podeu usar les funcions auxiliars que us calguin.

Problema 1: *Lambdes i fold*
Definiu una funció `sum2` que donada una llista de llistes de números ens retorni la suma total. Per exemple,
`sum2 [[3,-2,6],[8],[2,5,6],[]]` retorna `28`

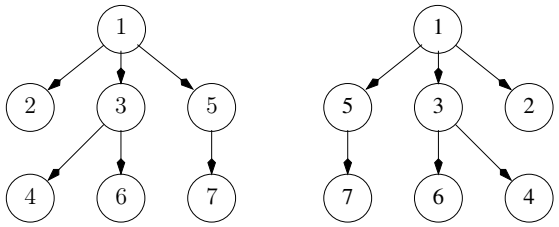
La definició s'ha de fer en una sola línia usant funcions predefinides de Haskell i lambda expressions. Es valorarà més si ho feu usant el fold (r o l).

```
sum2 ll = ...
```

Problema 2: *Map extra*
Definiu una funció `fsmap` que, donats un element x de tipus a i una llista lf de funcions de tipus $a \to a$, retorna l'aplicació de totes les funcions de lf (d'esquerra a dreta) a x. Per exemple:
`fsmap 3 [(+2),(*3),(+4)]` retorna `19`
`fsmap "o" [(++"la"),(:)'h',(++"!")]` retorna `"hola!"`

Problema 3: *No divisible*
Definiu una funció `knoDiv`, que donada una llista l d'enters més grans que 0 i un natural k, retorna el k-èssim (on 0 és el primer) enter positiu que no és divisible per cap nombre de la llista l. Per exemple,
`knoDiv 4 [2,3,5]` retorna `17`

Problema 4: *Arbres generals simètrics*
Definiu el tipus arbre general i feu una operació `simetric` que, donat un arbre general, ens retorna el seu simètric. Com a exemple, aquests dos arbres són simètrics:



Problema 5: *Cues eficients*
Volem representar les Cues de forma que millorem l'eficiència conjunta de les operacions d'afegir i avançar. Per això implementem la cua amb dos llistes tals que si concatenem la primera amb la inversa de la segona tenim els elements de la cua en l'ordre de sortida. Usant com a constructor `Cua`, un exemple de cua seria

```
c1 = Cua [2,8,5] [4,7]
```

que representa la cua on el primer és 2 i segueix amb 8, 5, 7 i 4.

D'aquesta manera afegir és fa posant l'element per davant de la segona llista (que és menys costós). D'altra banda si hem d'avançar traurem el primer de la primera llista, si en té i sinó passarem els de la segona llista a la primer (en l'ordre correcte) i agafarem el primer i deixarem la resta.

Apartat 5.a: Doneu la definició del tipus `Cua` i implementeu les operacions `afegir` i `avancar` (seguint les indicacions de l'enunciat). Noteu que `avancar` torna un parell que conté el primer element i una cua amb la resta d'elements.

```
cBuida :: Cua a
cBuida = Cua [] []

afegir :: a -> Cua a -> Cua a

avancar :: Cua a -> (a, Cua a)

esBuida :: Cua t -> Bool
esBuida (Cua [] []) = True
esBuida (Cua _ _) = False
```

Apartat 5.b: Definiu la igualtat de cues de manera que dues cues són igual si tenen els mateixos elements i el mateix ordre de sortida. Indiqueu que `Cua` és una `instance` de la classe `Eq` on (==) és la igualtat que heu definit (recordeu que cal que el tipus dels elements de la cua també sigui de la classe `Eq`). Com a exemple tenim que

```
snd (avancar (afegir 6 (afegir 3 cBuida))) == afegir 6 ( snd (avancar (afegir 3 cBuida)))
```

ha de retornar `True`