Parcial Llenguatges de Programació

Grau en Enginyeria Informàtica
Temps estimat: 1h i 45m

15 maig 2017

Només podeu entrar al compte de la forma `lpXX` que se us ha assignat. Entrar en qualsevol altre compte o que algú altre usi el vostre compte invalidarà el vostre examen i es considerarà còpia.
Per accedir al racó aneu a `https://examens.fib.upc.edu`

Cal que lliureu el codi via racó amb els comentaris que considereu necessaris en un arxiu "examen.hs" executable en l'entorn ghci sense activar cap opció addicional (només fent `ghci examen.hs`) i que solucioni els problemes que es llisten a continuació.

A la vostra solució heu de mantenir tots els noms que indiqui l'enunciat.

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

```
-- Problema 3.1
```

Es valorarà l'ús de funcions d'ordre superior predefinides i la simplicitat de les solucions. Ara bé, només es poden usar funcions predefinides de l'entorn Prelude: no podeu fer cap `import`.

A l'arxiu proves.txt trobareu exemples de crides i la seves solucions per a tenir alguns exemples de prova en format text.

Problema 1 (2 punts): *Remenar.*

Apartat 1.1: Feu la funció `shuffleOnce :: [a] -> [a]` que barreja una llista partint-la per la meitat i posant consecutivament un element de cada part començant per la segona part (si la llargada és imparell la primera meitat té un element menys). Per exemple, `shuffleOnce [8,18,6,11,13,9,2,6,10,12,16]` és
`[9,8,2,18,6,6,10,11,12,13,16]`.
i `shuffleOnce [3,5,46,7,9,10,20,11,1,12,8,6,2,13,27,31]` és
`[1,3,12,5,8,46,6,7,2,9,13,10,27,20,31,11]`.

Apartat 1.2: Feu la funció `shuffleBack :: Eq a => [a] -> Int` que donada una llista `l` retorna un enter que indica quants cops s'ha de remenar amb el `shuffleOnce` (mínim una) perquè torni a ser la llista inicial. Per exemple,
`shuffleBack [3,5,46,7,9,10,20,11,1,12,8,6,2,13,27,31]` és 8.

S'ha de fer usant el `shuffleOnce`.

Problema 2 (3.5 punt): *Mergesort amb segments.*
Apartat 2.1: Feu la funció `segments :: Ord a => [a] -> [[a]]` que, donada una llista, la parteix en segments creixents de mida màxima (sense canviar l'ordre original). Per exemple, `segments [3,5,46,7,9,10,20,11,1,12,8,6,2,13,27,31]` és
`[[3,5,46],[7,9,10,20],[11],[1,12],[8],[6],[2,13,27,31]]`.

Apartat 2.3: Feu la funció `mergeSegments :: Ord a => [[a]] -> [[a]]` que, donada una llista de llistes ordenades creixentment, retorna la mescla ordenada dos a dos de les llistes. És a dir, retorna la llista que conté la mescla ordenada de la primera i la segona, la mescla ordenada de la tercera i la quarta, ... Si n'hi ha un nombre senar, l'última es queda igual. Per exemple,
`mergeSegments [[3,5,46],[7,9,10,20],[11],[1,12],[8],[6],[2,13,27,31]]` és
`[[3,5,7,9,10,20,46],[1,11,12],[6,8],[2,13,27,31]]`.

Apartat 2.3: Feu la funció `mergeSegmentsort :: Ord a => [a] -> [a]` que ordena creixentment una llista usant les funcions `segments` i `mergeSegments` (la idea és una variant del *merge sort*).

Problema 3 (4.5 punts): *Aplanant expressions.* Cal que feu el següent:

Apartat 3.1: Definiu un data polimòrfic `FExpr a`, que permeti representar expressions construïdes amb constants de tipus `a` (amb el constructor `Const`) i amb operadors (que són strings) i una llista d'arguments que seran expressions també del mateix tipus `a` (amb el constructor `Func`). Com a exemples tenim:

```
let expr1 = Func "concat" [Const 'g', Func "concat" [Const 'h', Const 'c']]
let expr2 = Func "Plus" [Const 3, Func "Plus" [Const 6, Func "func1" [Const 2,Const 14,Const 8]]
```

Feu que `FExpr a` sigui de la classe `Show` mostrant-se tal com és l'expressió Haskell.

Apartat 3.2: Definiu la funció `flatten :: FExpr a -> FExpr a` que donada una expressió l'aplana de la següent manera: si x és un dels arguments d'una funció (sota el constructor `Func`) i x també està construït amb el constructor `Func`, amb el mateix operador (string) i una llista d'arguments l, llavors x es reemplaça (concatenant) per l. D'aquesta manera després d'aplanar tots els operadors iguals s'agrupen i no pot aparèixer mai un operador sota d'un altre d'igual. Per exemple, `flatten expr2` és
`Func "Plus" [Const 3,Const 6,Func "func1" [Const 2,Const 14,Const 8]]`
Apartat 3.3: Feu que `FExpr a` siguin `instance` de la class `Eq`, fent que dues expressions són iguals si després d'aplanar són idèntiques permutant els arguments de les funcions (és a dir, que els arguments són iguals encara que potser no estan en el mateix ordre). Per exemple, `expr2` i `Func "Plus" [Func "Plus" [Const 6, Func "func1" [Const 2,Const 8, Func "func1" [Const 14]]],Const 3]`
són iguals.

A l'arxiu `proves.txt` trobareu més exemples.