Parcial Llenguatges de Programació

Grau en Enginyeria Informàtica

10 Maig 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 4.a
```

Es valorarà l'us 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*
Definiu una funció `esDob` que donats dos nombres indiqui si el segon és el doble del primer. La definició s'ha de fer usant lambda expressions i funcions del Haskell i ha de ser de la forma

```
esDob = ...
```

Problema 2: *Divisibles*
Definiu una funció `elimDiv`, que donada una llista d'enters retorna una llista que preserva les posicions relatives de la llista donada, però on s'han eliminat aquells elements que eren divisibles per un d'anterior (és a dir, a l'esquerra). Per exemple,
`elimDiv [3,6,7,8,9,14,17,32]` retorna `[3,7,8,17]`

Problema 3: *Generació infinita*
Definiu una funció `infGen` que, donada una funció f d'enters a enters estrictament creixent, genera la llista infinita ordenada creixentment que inclou sempre l'1 i que, a més, inclou $f(x)$ si i només si x pertany a la llista.

Per exemple, `infGen (*2)` i `infGen (+3)` retornen respectivament

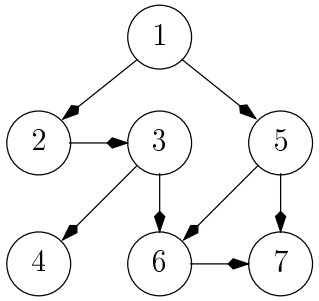
```
[1,2,4,8,16,32,64,...
[1,4,7,10,13,16,19,...
```

Feu a continuació una **nova** funció que donat un enter no negatiu n i una funció f calculi $f^n(1)$ usant `infGen`.

Problema 4: *Grafs dirigits*
Volem representar els grafs dirigits amb una llista de vèrtexs de tipus `v` i una funció que ens dona la llista d'adjacència i que té tipus `v -> [v]`.

Per exemple, usant com a constructor `G`, podem representar el graf de la dreta amb la (constant) gr1:

```
gr1 = G [1..7] fgr
    where fgr 1 = [2,5]
          fgr 2 = [3]
          fgr 3 = [4,6]
          fgr 4 = []
          fgr 5 = [6,7]
          fgr 6 = [7]
          fgr 7 = []
          fgr _ = []
```



Apartat 4.a: definiu un tipus de dades `Graf` amb `data` que segueixi aquesta descripció i definiu la igualtat de grafs com a igualtat de la llista de vèrtexs i la igualtat del resultat de la funció d'adjacència per a tots els vèrtexs de la llista. Per simplicitat, considereu igualtat sintàctica de les llistes. Indiqueu que Graf és una `instance` de la classe `Eq` on (==) és la igualtat que heu definit (recordeu que cal que el tipus dels vèrtexs també sigui de la classe `Eq`).

Apartat 4.b: Assumint que el graf és acíclic. Feu una funció `isReach` que donat un `Graf` i dos vèrtexs ens diu si es pot arribar al segon vèrtex des del primer.

Apartat 4.c: Definiu ara un nou tipus `PGraf` que sigui com l'anterior però on les arestes tenen pesos. Assumint que el graf és acíclic. Feu una funció `isPReach` que donat un `PGraf` i dos vèrtexs retorna un parell que ens diu si es pot arribar al segon vèrtex des del primer i, en cas de que es pugui, el pes total del camí trobat (no cal que sigui el millor).