Algoritmos 1

Primer cuatrimestre 2012

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

TP Funcional

Grupo 10

Integrante	LU	Correo electrónico
Gastón de Orta	244/11	gaston.deorta@hotmail.com
Leandro Lovisolo	645/11	leandro@leandro.me
María Candela Capra Coarasa	234/11	canduh_27@hotmail.com
Lautaro Jose Petaccio	443/11	lausuper@gmail.com

Módulo TPF

```haskell
module TPF where

import Tipos
import Atleta
import Competencia
import JJOO
```

Módulo Tipos

```haskell
module Tipos where

type Deporte = String
type Pais = String
type Categoria = (Deporte, Sexo)

data Sexo = Femenino | Masculino deriving (Show, Eq)
```

Módulo Atleta

```haskell
module Atleta (Atleta, nuevoA, nombreA, sexoA, anioNacimientoA,
               nacionalidadA, ciaNumberA, deportesA, capacidadA,
               entrenarDeporteA)
where

import Tipos

data Atleta = A String Sexo Int Pais Int [(Deporte, Int)]

nuevoA :: String -> Sexo -> Int -> Pais -> Int -> Atleta
nuevoA nom s a nac cia = (A nom s a nac cia [])

nombreA :: Atleta -> String
nombreA (A nombre _ _ _ _ _) = nombre

sexoA :: Atleta -> Sexo
sexoA (A _ sexo _ _ _ _) = sexo

anioNacimientoA :: Atleta -> Int
anioNacimientoA (A _ _ nacimiento _ _ _) = nacimiento

nacionalidadA :: Atleta -> Pais
nacionalidadA (A _ _ _ pais _ _) = pais

ciaNumberA :: Atleta -> Int
ciaNumberA (A _ _ _ _ ciaNumber _) = ciaNumber

deportesA :: Atleta -> [Deporte]
deportesA (A _ _ _ _ _ []) = []
deportesA (A a b c d e (x:xs)) = fst x : deportesA (A a b c d e xs)


capacidadA :: Atleta -> Deporte -> Int
capacidadA (A _ _ _ _ _ [x]) _ = snd x
capacidadA (A a b c d e (x:xs)) deporte
    | fst x == deporte = snd x
    | otherwise = capacidadA (A a b c d e xs) deporte

auxExisteDeporte :: [(Deporte, Int)] -> Deporte -> Bool
auxExisteDeporte [] _ = False
auxExisteDeporte (x:xs) dep
    | (fst x) == dep = True
    | otherwise = auxExisteDeporte xs dep

auxModificaCapacidad :: [(Deporte, Int)] -> Deporte -> Int -> [(Deporte, Int)]
auxModificaCapacidad [] _ _ = []
auxModificaCapacidad (deporte:ldeportes) depAModificar capAModificar
    | (fst deporte) == depAModificar =
        ((fst deporte), capAModificar) : (auxModificaCapacidad ldeportes depAModificar capAModificar)
    | otherwise = deporte : (auxModificaCapacidad ldeportes depAModificar capAModificar)

```

```haskell
auxAgregarDeporte :: [(Deporte, Int)] -> Deporte -> Int -> [(Deporte, Int)]
auxAgregarDeporte [] a b = [(a,b)]
auxAgregarDeporte (x:xs) dep cap
    | dep <= fst x = (dep,cap): (x:xs)
    | dep > fst x = x: auxAgregarDeporte xs dep cap

entrenarDeporteA :: Atleta -> Deporte -> Int -> Atleta
entrenarDeporteA (A nombre sexo anio pais cia deportes) depPorAgregar capPorAgregar
    | auxExisteDeporte deportes depPorAgregar =
        (A nombre sexo anio pais cia (auxModificaCapacidad deportes depPorAgregar capPorAgregar) )
    | otherwise = (A nombre sexo anio pais cia (auxAgregarDeporte deportes depPorAgregar capPorAgregar))

instance Show Atleta where
    show (A nombre sexo edad pais ciaNumber capacidades) = nombre ++ " (#" ++ show ciaNumber ++ ")"
```

Módulo Competencia

```haskell
module Competencia (Competencia, nuevaC, categoriaC, participantesC,
                    finalizadaC, rankingC, lesTocoControlAntiDopingC,
                    leDioPositivoC, finalizarC, linfordChristieC,
                    gananLosMasCapacesC, sancionarTrampososC)

where

import Tipos
import Atleta

data Competencia = C Categoria
                 | Participar Atleta Competencia
                 | Finalizar [Int] [(Int, Bool)] Competencia

nuevaC :: Deporte -> Sexo -> [Atleta] -> Competencia
nuevaC dep sex [] = C (dep, sex)
nuevaC dep sex (atle:atletas) = Participar atle (nuevaC dep sex atletas)

categoriaC :: Competencia -> Categoria
categoriaC (C categoria)  = categoria
categoriaC (Participar _ compe) = categoriaC compe
categoriaC (Finalizar _ _ compe) = categoriaC compe

participantesC :: Competencia -> [Atleta]
participantesC (C _) = []
participantesC (Participar atle compe) = atle : (participantesC compe)
participantesC (Finalizar _  _ compe) = participantesC compe

finalizadaC :: Competencia -> Bool
finalizadaC (Finalizar _ _ _) = True
finalizadaC _ = False

rankingC :: Competencia -> [Atleta]
rankingC (Finalizar ciaNumbers _ c) = atletasPorCiaNumber ciaNumbers c

atletasPorCiaNumber :: [Int] -> Competencia -> [Atleta]
atletasPorCiaNumber [] c = []
atletasPorCiaNumber (x:xs) c = (buscar x (participantesC c)) : atletasPorCiaNumber xs c
    where buscar ciaNumber (x:xs)
            | ciaNumber == ciaNumberA(x) = x
            | otherwise                  = buscar ciaNumber xs

lesTocoControlAntiDopingC :: Competencia -> [Atleta]
lesTocoControlAntiDopingC (Finalizar _ dopping c) = atletasPorCiaNumber (ciaNumbers dopping) c
    where ciaNumbers [] = []
          ciaNumbers ((n,_):xs) = n:(ciaNumbers xs)

leDioPositivoC :: Competencia -> Atleta -> Bool
leDioPositivoC (Finalizar _ dopping _) a = buscar dopping (ciaNumberA a)
    where buscar (x:xs) ciaNumber
            | fst x == ciaNumber = snd x
            | otherwise          = buscar xs ciaNumber

finalizarC :: Competencia -> [Int] -> [(Int, Bool)] -> Competencia
finalizarC compe posiciones dopping = Finalizar posiciones dopping compe

linfordChristieC :: Competencia -> Atleta -> Competencia
linfordChristieC (C cat) _ = C cat
linfordChristieC (Participar atle compe) atletaASacar
```

```haskell
    | (ciaNumberA atle) /= (ciaNumberA atletaASacar) =
        Participar atle (linfordChristieC compe atletaASacar)
    | otherwise = (linfordChristieC compe atletaASacar)

sancionarTrampososC :: Competencia -> Competencia
sancionarTrampososC (Finalizar ranking dopping compe) =
    Finalizar (auxSinTramposos ranking dopping) dopping compe

auxSinTramposos :: [Int] -> [(Int, Bool)] -> [Int]
auxSinTramposos ranking [] = ranking
auxSinTramposos [] _ = []
auxSinTramposos (rank:ranking) dopping
    | elem rank (auxCiaDoppingVerdadero dopping) = auxSinTramposos ranking dopping
    | otherwise                                  = rank : (auxSinTramposos ranking dopping)

auxCiaDoppingVerdadero :: [(Int, Bool)] -> [Int]
auxCiaDoppingVerdadero [] = []
auxCiaDoppingVerdadero (x:xs) | (snd x) == True = (fst x) : auxCiaDoppingVerdadero xs
                              | otherwise = auxCiaDoppingVerdadero xs

gananLosMasCapacesC :: Competencia -> Bool
gananLosMasCapacesC (Finalizar [] dopping compe) = True
gananLosMasCapacesC (Finalizar [x] dopping compe) = True
gananLosMasCapacesC (Finalizar (frank:srank:ranking) dopping compe) =
    (capacidadA (auxAtletaConCia frank (participantesC compe))
                (fst (categoriaC compe))) >=
    (capacidadA (auxAtletaConCia srank (participantesC compe))
                (fst (categoriaC compe))) &&
    gananLosMasCapacesC (Finalizar (srank:ranking) dopping compe)

auxAtletaConCia :: Int -> [Atleta] -> Atleta
auxAtletaConCia cia (atle:atletas) | (ciaNumberA atle) == cia = atle
                                   | otherwise = auxAtletaConCia cia atletas

instance Show Competencia where
    show c = "Competencia " ++ show (categoriaC c) ++ (participantes c) ++ (ranking c)
        where participantes c = if length (participantesC c) > 0
                                then ": " ++ show (participantesC c)
                                else "";
              ranking c = if finalizadaC c
                          then ", ranking: [" ++ ciaNumbers (rankingC c) ++ "]"
                          else "";
              ciaNumbers [x] = (show (ciaNumberA x));
              ciaNumbers (x:xs) = (show (ciaNumberA x)) ++ "," ++ ciaNumbers xs;
```

Módulo JJOO

```haskell
module JJOO (JJOO, nuevoJ, anioJ, atletasJ, cantDiasJ, cronogramaJ,
             jornadaActualJ, dePaseoJ, medalleroJ,
             boicotPorDisciplinaJ, losMasFracasadosJ, liuSongJ,
             stevenBradburyJ, uyOrdenadoAsiHayUnPatronJ, sequiaOlimpicaJ,
             transcurrirDiaJ)
where

import Tipos
import Atleta
import Competencia

data JJOO = J Int [Atleta] Int
          | NuevoDia [Competencia] JJOO
            deriving (Show)

nuevoJ :: Int -> [Atleta] -> [[Competencia]] -> JJOO
nuevoJ anio atletas [] = (J anio atletas 1)
nuevoJ anio atletas xs = NuevoDia (last xs) (nuevoJ anio atletas (init xs))

anioJ :: JJOO -> Int
anioJ (J anio _ _) = anio
anioJ (NuevoDia _ juegos) = anioJ juegos

atletasJ :: JJOO -> [Atleta]
atletasJ (J _ atletas _) = atletas
atletasJ (NuevoDia _ juegos) = atletasJ juegos

cantDiasJ :: JJOO -> Int
cantDiasJ (J _ _ _) = 0
```

```haskell
cantDiasJ (NuevoDia _ juegos) = 1 + cantDiasJ juegos

cronogramaJ :: JJOO -> Int -> [Competencia]
cronogramaJ (J _ _ _) _ = []
cronogramaJ (NuevoDia competencias juegos) dias
    | dias == (cantDiasJ juegos) + 1 = competencias
    | otherwise                      = cronogramaJ juegos dias

jornadaActualJ :: JJOO -> Int
jornadaActualJ (J _ _ jornadaActual) = jornadaActual
jornadaActualJ (NuevoDia _ juegos) = jornadaActualJ juegos


-------------------------------------------------------------------------------
-- dePaseoJ -------------------------------------------------------------------
-------------------------------------------------------------------------------

dePaseoJ :: JJOO -> [Atleta]
dePaseoJ (J _ atletas _)          = atletas
dePaseoJ (NuevoDia [] juegos)     = dePaseoJ juegos
dePaseoJ (NuevoDia (x:xs) juegos) = dePaseoJ (juegoSinAtletas (NuevoDia xs juegos) (participantesC x))

juegoSinAtletas :: JJOO -> [Atleta] -> JJOO
juegoSinAtletas (J anio atletas d) atletasARemover = (J anio (removerAtletas atletas atletasARemover) d)
juegoSinAtletas (NuevoDia c j) atletasARemover     = (NuevoDia c (juegoSinAtletas j atletasARemover))

removerAtletas :: [Atleta] -> [Atleta] -> [Atleta]
removerAtletas atletas [] = atletas
removerAtletas [] _       = []
removerAtletas (x:xs) atletasARemover
    | elem (ciaNumberA x) (auxAtletasACias atletasARemover) = removerAtletas xs atletasARemover
    | otherwise = x : (removerAtletas xs atletasARemover)

auxAtletasACias :: [Atleta] -> [Int]
auxAtletasACias [] = []
auxAtletasACias (atle:atletas) = (ciaNumberA atle) : (auxAtletasACias atletas)

-------------------------------------------------------------------------------
-- Fin de dePaseoJ ------------------------------------------------------------
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- medalleroJ -----------------------------------------------------------------
-------------------------------------------------------------------------------

medalleroJ :: JJOO -> [(Pais, [Int])]
medalleroJ j = medallero (paisesGanadoresOrdenados j)
    where medallero [] = []
          medallero (x:xs) = (medalleroPorPais x j) : medallero xs

paisesGanadoresOrdenados :: JJOO -> [Pais]
paisesGanadoresOrdenados j = ordenar (paisesGanadores j)
    where ordenar [] = []
          ordenar [p] = [p]
          ordenar (p1:p2:ps) = bubbleSort (p1:p2:ps) [] False
          bubbleSort (p1:p2:ps) acc flag
            | (tieneMasMedallas p1 p2 j) = bubbleSort (p2:ps) (acc ++ [p1]) flag
            | otherwise                  = bubbleSort (p1:ps) (acc ++ [p2]) True
          bubbleSort [p] acc flag
            | flag == True = bubbleSort (acc ++ [p]) [] False
            | otherwise    = (acc ++ [p])

paisesGanadores :: JJOO -> [Pais]
paisesGanadores j = sinRepetidos (obtenerPaises ((medallistas 0 j) ++
                                                 (medallistas 1 j) ++
                                                 (medallistas 2 j)))
    where obtenerPaises [] = []
          obtenerPaises (x:xs) = (nacionalidadA x) : obtenerPaises xs

-- Devuelve los elementos de una lista sin repetir.
sinRepetidos :: Eq a => [a] -> [a]
sinRepetidos [] = []
sinRepetidos (x:xs) = if (elem (last (x:xs)) (init (x:xs)))
                        then sinRepetidos (init (x:xs))
                        else (sinRepetidos (init (x:xs))) ++ [last (x:xs)]

-- Dados unos JJOO y una posiciÃ³n, devuelve todos los atletas
-- que finalizaron alguna competencia en esa posiciÃ³n.
```

```haskell
medallistas :: Int -> JJOO -> [Atleta]
medallistas m j = obtenerMedallistas (competenciasFinalizadas j) m
    where obtenerMedallistas [] m = []
          obtenerMedallistas (x:xs) m
              | (length (rankingC x)) <= m = obtenerMedallistas xs m
              | otherwise                  = (rankingC x !! m) : obtenerMedallistas xs m

-- Devuelve las competencias finalizadas hasta la jornada actual inclusive.
competenciasFinalizadas :: JJOO -> [Competencia]
competenciasFinalizadas j = competencias (jornadaActualJ j) j
    where competencias 1 j = soloFinalizadas (cronogramaJ j 1)
          competencias d j = soloFinalizadas (cronogramaJ j d) ++ (competencias (d - 1) j)
          soloFinalizadas [] = []
          soloFinalizadas (c:cs) = if finalizadaC c
                                   then c : soloFinalizadas cs
                                   else soloFinalizadas cs

tieneMasMedallas :: Pais -> Pais -> JJOO -> Bool
tieneMasMedallas p1 p2 j =
         (m1 !! 0 >  m2 !! 0) ||
        ((m1 !! 0 == m2 !! 0) && (m1 !! 1 >  m2 !! 1)) ||
        ((m1 !! 0 == m2 !! 0) && (m1 !! 1 == m2 !! 1) && (m1 !! 2 >= m2 !! 2))
    where m1 = snd (medalleroPorPais p1 j)
          m2 = snd (medalleroPorPais p2 j)

medalleroPorPais :: Pais -> JJOO -> (Pais, [Int])
medalleroPorPais p j = (p, (medallero p j))
    where medallero p j = [length (medallas p 0 j),
                           length (medallas p 1 j),
                           length (medallas p 2 j)]
          medallas p m j = filtrarPorPais p (medallistas m j)
          filtrarPorPais p [] = []
          filtrarPorPais p (x:xs) = if nacionalidadA x == p
                                    then x : (filtrarPorPais p xs)
                                    else filtrarPorPais p xs

-------------------------------------------------------------------------------
-- Fin de medalleroJ ----------------------------------------------------------
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- transcurrirDiaJ ------------------------------------------------------------
-------------------------------------------------------------------------------

transcurrirDiaJ :: JJOO -> JJOO
transcurrirDiaJ j = transcurrir j ((cantDiasJ j) - (jornadaActualJ j))
    where transcurrir (J anio atletas diaActual) _ = (J anio atletas (diaActual + 1))
          transcurrir (NuevoDia cs j) 0 = (NuevoDia (auxFinalizarCompetencias cs) (transcurrir j (-1)))
          transcurrir (NuevoDia cs j) i = (NuevoDia cs (transcurrir j (i - 1)))

auxFinalizarCompetencias :: [Competencia] -> [Competencia]
auxFinalizarCompetencias [] = []
auxFinalizarCompetencias (compe:competencias)
    | finalizadaC compe = compe : (auxFinalizarCompetencias competencias)
    | otherwise         = (finalizarC compe
                                      (auxCrearRanking (participantesC compe) (categoriaC compe))
                                      (auxCrearDopping compe))
                              : (auxFinalizarCompetencias competencias)

auxCrearRanking :: [Atleta] -> Categoria -> [Int]
auxCrearRanking [] _ = []
auxCrearRanking atletas cate =
    (ciaNumberA (auxMayorCapacidad atletas cate (head atletas))) :
        (auxCrearRanking (auxSacaUnaVez atletas (auxMayorCapacidad
                                                 atletas cate (head atletas))) cate)

auxSacaUnaVez :: [Atleta]-> Atleta -> [Atleta]
auxSacaUnaVez [] at = []
auxSacaUnaVez (atle:atletas) at
    | (ciaNumberA atle) == (ciaNumberA at) = auxSacaUnaVez atletas at
    | otherwise                            = atle : (auxSacaUnaVez atletas at)

auxMayorCapacidad :: [Atleta] -> Categoria -> Atleta -> Atleta
auxMayorCapacidad [] _ atleMax = atleMax
auxMayorCapacidad (atleta:atletas) cate atleMax
    | (capacidadA atleta (fst cate)) >= (capacidadA atleMax (fst cate)) =
      auxMayorCapacidad atletas cate atleta
    | otherwise = auxMayorCapacidad atletas cate atleMax
```

```haskell

auxCrearDopping :: Competencia -> [(Int, Bool)]
auxCrearDopping c
    | length (participantesC c) >= 1 = [(ciaNumberA (head (participantesC c)), False)]
    | otherwise                      = []

-------------------------------------------------------------------------------
-- Fin de transcurrirDiaJ -----------------------------------------------------
-------------------------------------------------------------------------------



-------------------------------------------------------------------------------
-- boicotPorDisciplinaJ -------------------------------------------------------
-------------------------------------------------------------------------------

boicotPorDisciplinaJ :: JJOO -> (Deporte, Sexo) -> Pais -> (Int, JJOO)
boicotPorDisciplinaJ j cat p = (cantAtletasBoicoteados, boicotearJornada j)
    where boicotearJornada (J x y z) = (J x y z)
          boicotearJornada (NuevoDia cs j) = (NuevoDia (boicotearCategorias cs)
                                                       (boicotearJornada j))
          boicotearCategorias [] = []
          boicotearCategorias (x:xs)
            | (categoriaC x == cat) && (not (finalizadaC x)) =
                (nuevaC (fst (categoriaC x))
                        (snd (categoriaC x))
                        (boicotearAtletas (participantesC x))) : (boicotearCategorias xs)
            | (categoriaC x == cat) && (finalizadaC x) =
                (finalizarC (nuevaC (fst cat) (snd cat)
                                    (boicotearAtletas (participantesC x)))
                            (boicotearRanking (rankingC x))
                            (boicotearDoping x)) : (boicotearCategorias xs)
            | otherwise = x:(boicotearCategorias xs)
          boicotearAtletas [] = []
          boicotearAtletas (x:xs)
            | nacionalidadA x == p = boicotearAtletas xs
            | otherwise            = x:(boicotearAtletas xs)
          boicotearRanking xs = ciaNumbers (boicotearAtletas xs)
          ciaNumbers [] = []
          ciaNumbers (x:xs) = (ciaNumberA x):(ciaNumbers xs)
          boicotearDoping c = tuplasDoping (boicotearAtletas (lesTocoControlAntiDopingC c)) c
          tuplasDoping [] _ = []
          tuplasDoping (x:xs) c = (ciaNumberA x, leDioPositivoC c x):(tuplasDoping xs c)
          cantAtletasBoicoteados = contarBoicoteados (categoriaBoicoteada (categorias 1))
          categorias d
            | d < cantDiasJ j = (cronogramaJ j d) ++ (categorias (d + 1))
            | otherwise       =  cronogramaJ j d
          categoriaBoicoteada (x:xs)
            | categoriaC x == cat = x
            | otherwise           = categoriaBoicoteada xs
          contarBoicoteados x = (length (participantesC x)) -
                                (length (boicotearAtletas (participantesC x)))

-------------------------------------------------------------------------------
-- Fin de boicotPorDisciplinaJ ------------------------------------------------
-------------------------------------------------------------------------------



-------------------------------------------------------------------------------
-- losMasFracasadosJ-----------------------------------------------------------
-------------------------------------------------------------------------------

losMasFracasadosJ :: JJOO -> Pais -> [Atleta]
losMasFracasadosJ j p = noGanaronMedallas (losMasParticipantes atletasDelPais)
    where noGanaronMedallas [] = []
          noGanaronMedallas (x:xs)
            | ganoMedallas x = noGanaronMedallas xs
            | otherwise      = x:(noGanaronMedallas xs)
          ganoMedallas a = elem (ciaNumberA a) (ciaNumbers ((medallistas 0 j) ++
                                                            (medallistas 1 j) ++
                                                            (medallistas 2 j)))
          losMasParticipantes [] = []
          losMasParticipantes xs = obtenerLosMasParticipantes xs xs
          obtenerLosMasParticipantes [] _      = []
          obtenerLosMasParticipantes (x:xs) ys
            | esMasParticipante x ys = x:(obtenerLosMasParticipantes xs ys)
            | otherwise              = obtenerLosMasParticipantes xs ys
          esMasParticipante _ [] = True
          esMasParticipante w (x:xs) =
                ((participacion w competencias) >= (participacion x competencias)) &&
```

```
                (esMasParticipante w xs)
          competencias = obtenerCompetencias 1
          obtenerCompetencias d
            | d <= cantDiasJ j = (cronogramaJ j d) ++ (obtenerCompetencias (d + 1))
            | otherwise        = []
          participacion _ [] = 0
          participacion a (x:xs)
            | elem (ciaNumberA a) (ciaNumbers (participantesC x)) = 1 + (participacion a xs)
            | otherwise                                          = participacion a xs
          atletasDelPais = obtenerAtletasDelPais (atletasJ j)
          obtenerAtletasDelPais [] = []
          obtenerAtletasDelPais (x:xs)
            | nacionalidadA x == p = x:(obtenerAtletasDelPais xs)
            | otherwise            = obtenerAtletasDelPais xs
          ciaNumbers [] = []
          ciaNumbers (x:xs) = (ciaNumberA x):(ciaNumbers xs)

-- ---------------------------------------------------------------------------
-- Fin de losMasFracasadosJ --------------------------------------------------
-- ---------------------------------------------------------------------------


-- ---------------------------------------------------------------------------
-- liuSongJ ------------------------------------------------------------------
-- ---------------------------------------------------------------------------

liuSongJ :: JJOO -> Atleta -> Pais -> JJOO
liuSongJ (J anio atletas d) liu p = (J anio (cambiarAtletas atletas liu p) d)
liuSongJ (NuevoDia cs j) liu p    = (NuevoDia (cambiarCompetencias cs liu p) (liuSongJ j liu p))

cambiarAtletas :: [Atleta] -> Atleta -> Pais -> [Atleta]
cambiarAtletas [] _ _ = []
cambiarAtletas (x:xs) liu p
        | ciaNumberA x == ciaNumberA liu = nacionalizarLiu:(cambiarAtletas xs liu p)
        | otherwise                      = x:(cambiarAtletas xs liu p)
    where nacionalizarLiu                = entrenarLiu nuevoLiu (deportesA liu)
          nuevoLiu                       = (nuevoA (nombreA liu) (sexoA liu) (anioNacimientoA liu)
                                                   p (ciaNumberA liu))
          entrenarLiu liu' []            = liu'
          entrenarLiu liu' (x:xs)        = entrenarLiu (entrenarDeporteA liu' x (capacidadA liu x)) xs

cambiarCompetencias :: [Competencia] -> Atleta -> Pais -> [Competencia]
cambiarCompetencias [] _ _= []
cambiarCompetencias (x:xs) liu p = (cambiarCompetencia x):(cambiarCompetencias xs liu p)
    where cambiarCompetencia c
            | finalizadaC c            = finalizarC (nuevaCompetencia c)
                                                    (ciaNumbers (rankingC c))
                                                    (tuplasDoping c)
            | otherwise                = nuevaCompetencia c
          nuevaCompetencia c           = (nuevaC (fst (categoriaC c))
                                                 (snd (categoriaC c))
                                                 (cambiarAtletas (participantesC c) liu p))
          ciaNumbers []                = []
          ciaNumbers (x:xs)            = (ciaNumberA x):(ciaNumbers xs)
          tuplasDoping c               = obtenerTuplasDoping c (lesTocoControlAntiDopingC c)
          obtenerTuplasDoping c []     = []
          obtenerTuplasDoping c (x:xs) = (ciaNumberA x, leDioPositivoC c x):(obtenerTuplasDoping c xs)

-- ---------------------------------------------------------------------------
-- Fin de liuSongJ -----------------------------------------------------------
-- ---------------------------------------------------------------------------


-- ---------------------------------------------------------------------------
-- uyOrdenadoAsiHayUnPatronJ -------------------------------------------------
-- ---------------------------------------------------------------------------

uyOrdenadoAsiHayUnPatronJ :: JJOO -> Bool
uyOrdenadoAsiHayUnPatronJ juegos = auxUyOrdenadoAsiHayUnPatronJ juegos []

auxUyOrdenadoAsiHayUnPatronJ :: JJOO -> [Pais] -> Bool
auxUyOrdenadoAsiHayUnPatronJ (J _ _ _) lista = (auxExistePatron lista ((length lista) - 1)
                                                                    ((length lista) - 1))
auxUyOrdenadoAsiHayUnPatronJ (NuevoDia competencias juegos) paises
    | auxExisteAlgunoConRanking competencias =
        auxUyOrdenadoAsiHayUnPatronJ juegos
                                     ((auxMejorPaisEnElDia (auxPaisesGanadoresEnElDia competencias [])
                                      (head(auxPaisesGanadoresEnElDia competencias [])))):paises)
    | otherwise = auxUyOrdenadoAsiHayUnPatronJ juegos paises
```

```haskell

auxExisteAlgunoConRanking :: [Competencia] -> Bool
auxExisteAlgunoConRanking [] = False
auxExisteAlgunoConRanking (compe:competencias)
    | (finalizadaC compe) = ((length (rankingC compe)) > 0)
    | otherwise           = auxExisteAlgunoConRanking competencias

auxMejorPaisEnElDia :: [(Pais, Int)] -> (Pais, Int) -> Pais
auxMejorPaisEnElDia [] pais = (fst pais)
auxMejorPaisEnElDia (pais:paises) paisMax
    |  ((snd paisMax) > (snd pais)) ||
      (((snd paisMax) == (snd pais)) &&
       ((fst paisMax) < (fst pais))) = auxMejorPaisEnElDia paises paisMax
    | otherwise                      = auxMejorPaisEnElDia paises pais

auxMeterPais :: [(Pais, Int)] -> Pais -> [(Pais, Int)]
auxMeterPais [] pais = [(pais,1)]
auxMeterPais (pais:paises) paisAMeter
    | ((fst pais) == paisAMeter) = (fst pais, (snd(pais)+1)) : paises
    | otherwise                  = pais : (auxMeterPais paises paisAMeter)

auxPaisesGanadoresEnElDia :: [Competencia] -> [(Pais, Int)] -> [(Pais, Int)]
auxPaisesGanadoresEnElDia [] paises = paises
auxPaisesGanadoresEnElDia (compe:competencias) paises
    | (finalizadaC compe) && (length (rankingC compe) > 0) =
          auxPaisesGanadoresEnElDia competencias
                                    (auxMeterPais paises (nacionalidadA ((rankingC compe) !! 0)))
    | otherwise = auxPaisesGanadoresEnElDia competencias paises

auxRecorreYCompara :: [Pais] -> Pais -> Pais -> Int -> Int -> Bool
auxRecorreYCompara paises paisBuscado paisSiguiente 0 maximo
    | ((paises!!0) == paisBuscado) && ((paises!!1) == paisSiguiente) = True
    | otherwise                                                      = True

auxRecorreYCompara paises paisBuscado paisSiguiente indice maximo
    |  (indice < maximo) &&
      ((paises!!indice) == paisBuscado) = ((paises!!(indice+1)) == paisSiguiente) &&
                                          (auxRecorreYCompara paises paisBuscado
                                                              paisSiguiente (indice-1) maximo)
    |  (indice == maximo) &&
      ((paises!!indice) == paisBuscado) = True && (auxRecorreYCompara paises paisBuscado
                                                                      paisSiguiente (indice-1) maximo)
    | otherwise = (auxRecorreYCompara paises paisBuscado paisSiguiente (indice-1) maximo)

auxExistePatron :: [Pais] -> Int -> Int -> Bool
auxExistePatron [] _ _ = True
auxExistePatron [x] _ _ = True
auxExistePatron [x,y] _ _ = True
auxExistePatron paises 0 maximo = auxRecorreYCompara paises (paises!!0) (paises!!1) maximo maximo
auxExistePatron paises indice maximo
    | (indice == maximo) = True && (auxExistePatron paises (indice-1) maximo)
    | otherwise          = (auxRecorreYCompara paises (paises !! indice)
                                               (paises !! (indice + 1))
                                                maximo maximo) &&
                           (auxExistePatron paises (indice-1) maximo)

-------------------------------------------------------------------------------
-- Fin de uyOrdenadoAsiHayUnPatronJ -----------------------------------------
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- stevenBradburyJ ----------------------------------------------------------
-------------------------------------------------------------------------------

stevenBradburyJ :: JJOO -> Atleta
stevenBradburyJ j = buscarElMenosCapaz (tuplasMedallistasCapacidad j)
    where buscarElMenosCapaz [x]       = fst x
          buscarElMenosCapaz (x:xs)
            | esElMenosCapaz x (x:xs)  = fst x
            | otherwise                = buscarElMenosCapaz xs
          esElMenosCapaz _ []          = True
          esElMenosCapaz a (x:xs)      = (snd(a) <= snd(x)) && (esElMenosCapaz a xs)
          tuplasMedallistasCapacidad j = obtenerTuplas (competenciasFinalizadas j)
          obtenerTuplas []             = []
          obtenerTuplas (x:xs)
            | length (rankingC x) == 0 = obtenerTuplas xs
            | otherwise                = obtenerTupla x : obtenerTuplas xs
          obtenerTupla x               = (head (rankingC x), capacidadA (head (rankingC x))
```

```haskell
                                                                    (fst (categoriaC x)))

-------------------------------------------------------------------------------
-- Fin de stevenBradburyJ -----------------------------------------------------
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- sequiaOlimpicaJ ------------------------------------------------------------
-------------------------------------------------------------------------------

sequiaOlimpicaJ :: JJOO -> [Pais]
sequiaOlimpicaJ j = buscarMasSecos (obtenerPaises (atletasJ j)) (obtenerPaises (atletasJ j))
    where buscarMasSecos [] _ = []
          buscarMasSecos [x] _ = [x]
          buscarMasSecos (x:xs) paises
              | esMasSeco x paises = x:(buscarMasSecos xs paises)
              | otherwise          = buscarMasSecos xs paises
          esMasSeco _ [] = True
          esMasSeco w (x:xs) = (maxDiasSinGanar w j >= maxDiasSinGanar x j) &&
                               (esMasSeco w xs)
          obtenerPaises [] = []
          obtenerPaises (x:xs) = sinRepetidos ((nacionalidadA x):(obtenerPaises xs))
          maxDiasSinGanar p j = buscarMax (calcularDiferencias (jornadas p j))
          jornadas p j= 0 : (jornadasEnLasQueGano p j) ++ [jornadaActualJ j]
          calcularDiferencias [x,y] = [y - x]
          calcularDiferencias (x1:x2:xs) = (x2 - x1):(calcularDiferencias (x2:xs))
          buscarMax [x] = x
          buscarMax (x:xs)
            | esMax x xs = x
            | otherwise  = buscarMax xs
          esMax _ [] = True
          esMax w (x:xs) = (w >= x) && (esMax w xs)

jornadasEnLasQueGano :: Pais -> JJOO -> [Int]
jornadasEnLasQueGano p j = acumularJornadasEnLasQueGano p 1 j
    where acumularJornadasEnLasQueGano p d j
            | d > jornadaActualJ j     = []
            | ganoMedallasEseDia p d j = d:(acumularJornadasEnLasQueGano p (d + 1) j)
            | otherwise                =    acumularJornadasEnLasQueGano p (d + 1) j
          ganoMedallasEseDia p d j = ganoAlgunaMedalla p (filtrarFinalizadas (cronogramaJ j d))
          filtrarFinalizadas [] = []
          filtrarFinalizadas (x:xs)
              | finalizadaC x = x:(filtrarFinalizadas xs)
              | otherwise     = filtrarFinalizadas xs
          ganoAlgunaMedalla p [] = False
          ganoAlgunaMedalla p (x:xs) = (salioEnPosicion p 1 x  ||
                                        salioEnPosicion p 2 x  ||
                                        salioEnPosicion p 3 x) || ganoAlgunaMedalla p xs
          salioEnPosicion pais pos c = length (rankingC c) >= pos &&
                                       nacionalidadA (rankingC c !! (pos - 1)) == pais

-------------------------------------------------------------------------------
-- Fin de sequiaOlimpicaJ -----------------------------------------------------
-------------------------------------------------------------------------------
```