1. TAD Mago 🧙

Este TAD representa a un mago individual, definido por su nombre y el conjunto de hechizos que ha aprendido.

Definición e Implementación

Haskell

```
-- Tipos base
type Hechizo = String
type Nombre = String

-- Estructura de datos
data Mago = M Nombre (Set Hechizo)
```

Funciones e Implementaciones

- **crearM** :: Nombre -> Mago -- Costo: O(1)

Haskell

```
crearM n = M n emptyS
```

- **nombre** :: Mago -> Nombre -- Costo: O(1)

Haskell

```
nombre (M n _) = n
```

- **aprender** :: Hechizo -> Mago -> Mago -- Costo: O(logH)

Haskell

```
aprender h (M n hs) = M n (addS h hs)
```

- **hechizos** :: Mago -> Set Hechizo -- Costo: O(1)

Haskell

```
hechizos (M n hs) = hs
```

- **Instancias de Eq y Ord** -- Costo: O(1)

Haskell

```
instance Eq Mago where
 (M n1 _) == (M n2 _) = n1 == n2

instance Ord Mago where
 -- Un mago es "menor o igual" si sabe más o la misma cantidad de hechizos.
 (M _ hs1) <= (M _ hs2) = sizeS hs1 >= sizeS hs2
```

---

2. TAD Escuela de Magia 🏰

Este TAD gestiona una colección de magos y los hechizos disponibles.

Definición e Invariantes de Representación

La escuela se implementa con un conjunto para los hechizos enseñados, un mapa para buscar magos por nombre y una cola de prioridad para encontrar al mago más poderoso rápidamente.

Haskell

```
data EscuelaDeMagia = EDM (Set Hechizo) (Map Nombre Mago) (PriorityQueue Mago)
```

Invariantes de Representación (Reglas Clave):

- Todos los magos del Map deben estar también en la PriorityQueue y viceversa.
- Todos los hechizos que un mago conoce deben estar en el Set de hechizos de la escuela.
- No puede haber dos magos con el mismo nombre en la PriorityQueue.

Funciones e Implementaciones

- **fundarEscuela** :: EscuelaDeMagia -- Costo: O(1)

Haskell

```
fundarEscuela = EDM emptyS emptyM emptyPQ
```

- **estaVacia** :: EscuelaDeMagia -> Bool -- Costo: O(1)

Haskell

estaVacia (EDM _ _ pqm) = isEmptyPQ pqm

- **magos** :: EscuelaDeMagia -> [Nombre] -- Costo: O(M)

Haskell
magos (EDM _ mm _) = domM mm

- **registrar** :: Nombre -> EscuelaDeMagia -> EscuelaDeMagia -- Costo: O(logM)

Haskell
registrar n (EDM sh mm pqm) =
 case lookupM n mm of
  Nothing -> let m = crearM n
        in EDM sh (assocM n m mm) (insertPQ m pqm)
  Just m -> EDM sh mm pqm

- **hechizosDe** :: Nombre -> EscuelaDeMagia -> Set Hechizo -- Costo: O(logM)

Haskell
hechizosDe n (EDM _ mm _) =
 case lookupM n mm of
  Nothing -> error "Esa persona no es alumne de la escuela"
  Just m -> hechizos m

- **leFaltaAprender** :: Nombre -> EscuelaDeMagia -> Int -- Costo: O(logM)

Haskell
leFaltaAprender n (EDM sh mm _) =
 case lookupM n mm of
  Nothing -> error "No es alumne de la escuela"
  Just m -> sizeS sh - sizeS (hechizos m)

- **egresarUno** :: EscuelaDeMagia -> (Mago, EscuelaDeMagia) -- Costo: O(logM)

Haskell
egresarUno (EDM sh mm pqm) =
 let m = maxPQ pqm
 in (m, EDM sh (deleteM (nombre m) mm) (deleteMaxPQ pqm))

- **enseñar** :: Hechizo -> Nombre -> EscuelaDeMagia -> EscuelaDeMagia -- Costo: O(MlogM+logH)

Haskell
enseñar h n (EDM sh mm pqm) =
 case lookupM n mm of
  Nothing -> error "No es alumne de la escuela"
  Just m -> let newM = aprender h m
        in EDM (addS h sh) (assocM n newM mm) (modificarPQ newM pqm)

- **Función Auxiliar modificarPQ** -- Costo: O(MlogM)

Haskell
-- PRECOND: Hay un mago con el mismo nombre que el mago dado.
modificarPQ :: Mago -> PriorityQueue Mago -> PriorityQueue Mago
modificarPQ m pqm =
 let maxM = maxPQ pqm
 in if m == maxM
   then insertPQ m (deleteMaxPQ pqm)
   else insertPQ maxM (modificarPQ m (deleteMaxPQ pqm))

---

3. Funciones de Usuario 🧑‍💻

Estas funciones se crean utilizando únicamente la interfaz pública del TAD EscuelaDeMagia.

- **hechizosAprendidos** :: EscuelaDeMagia -> Set Hechizo -- Costo: O(M·(logM+HlogH))

Haskell

```
hechizosAprendidos :: EscuelaDeMagia -> Set Hechizo
hechizosAprendidos escuela = hechizosDeEn (magos escuela) escuela

hechizosDeEn :: [Nombre] -> EscuelaDeMagia -> Set Hechizo
hechizosDeEn [] _ = emptyS
hechizosDeEn (n:ns) escuela =
 unionS (hechizosDe n escuela) (hechizosDeEn ns escuela)
```

- **hayUnExperto** :: EscuelaDeMagia -> Bool -- Costo: O(logM)

```
Haskell
hayUnExperto :: EscuelaDeMagia -> Bool
hayUnExperto escuela =
 let (m, _) = egresarUno escuela
 in leFaltaAprender (nombre m) escuela == 0
```

- **egresarExpertos** :: EscuelaDeMagia -> ([Mago], EscuelaDeMagia) -- Costo: O(MlogM)

```
Haskell
egresarExpertos :: EscuelaDeMagia -> ([Mago], EscuelaDeMagia)
egresarExpertos escuela =
 if not (hayUnExperto escuela)
 then ([], escuela)
 else let (m, escuelaSinM) = egresarUno escuela
        (ms, escuelaSinMs) = egresarExpertos escuelaSinM
    in (m:ms, escuelaSinMs)
```