Estructuras de datos

Clases teóricas

por Pablo E. “Fidel” Martínez López

8. Repaso 2

Repaso

Tipos abstractos de datos

- Los tipos abstractos de datos (TADs)
  - Quedan definidos por su ***interfaz***
  - Induce roles en la manera de usarlo
    - Diseñador, usuario, implementador
    - Cada rol tiene diferentes obligaciones y responsabilidades
  - Requieren ciertas herramientas para implementarlos
    - Invariantes de representación, eficiencia

Eficiencia

- Para medir eficiencia se usan modelos especiales
  - Modelo de ***peor caso*** para medir operaciones
    - En base al comportamiento del peor caso
    - En función de la cantidad de elementos de la estructura
  - Clasificación
    - Constante, siempre el mismo costo, O(1)
    - Logarítmico, búsqueda en un árbol, O(log n)
    - Lineal, solo operaciones constantes por elemento, O(n)
    - “Eneloguene”, hasta operaciones logarítmicas por elemento, O(n log n)
    - Cuadrática, hasta operaciones lineales por elemento, O(n^2)

Tipos abstractos de datos

- Existen TADs clásicos que hay que conocer
  - Stacks, Queues, Sets, PriorityQueues, Maps, Multisets
  - Implementaciones: BSTs, AVLs, Heaps
- Al estudiarlos, aprendemos las herramientas necesarias
  - Como usuario
    - Usar la intefaz sin conocer implementaciones
  - Como implementador
    - Elección de representaciones eficaces
    - Formas de invariantes útiles para mejorar eficiencia
    - Mediciones de eficiencia para tener alternativas

Uso de TADs

Tipos abstractos de datos: ejemplo

- TAD Mago (H es el máximo número de hechizos de un mago)

```
type Hechizo = String
type Nombre  = String

data Mago  -- TAD

crearM      :: Nombre -> Mago                  -- O(1)
nombre      :: Mago -> Nombre                  -- O(1)
aprender    :: Hechizo -> Mago -> Mago         -- O(log H)
hechizos    :: Mago -> Set Hechizo             -- O(1)
(==), (<=)  :: Mago -> Mago -> Bool            -- O(1)
    -- Iguales, mismo nombre; menor, más hechizos
```

Tipos abstractos de datos: ejemplo

- TAD Escuela de Magia

  (H total de hechizos, M cantidad de magos)

```
data EscuelaDeMagia -- TAD

fundarEscuela    :: EscuelaDeMagia                                -- O(1)
estaVacia        :: EscuelaDeMagia -> Bool                        -- O(1)
registrar        :: Nombre -> EscuelaDeMagia -> EscuelaDeMagia    -- O(log M)
magos            :: EscuelaDeMagia -> [Nombre]                    -- O(M)
hechizosDe       :: Nombre -> EscuelaDeMagia -> Set Hechizo       -- O(log M)
leFaltaAprender  :: Nombre -> EscuelaDeMagia -> Int               -- O(log M)
egresarUno       :: EscuelaDeMagia -> (Mago, EscuelaDeMagia)      -- O(log M)
enseñar          :: Hechizo -> Nombre                             -- O(M log M
                    -> EscuelaDeMagia -> EscuelaDeMagia           --   + log H)
```

Tipos abstractos de datos

- TAD Escuela de Magia
  - Representación (dada)

```
data EscuelaDeMagia =
  EDM (Set Hechizo)           -- Todos los enseñados
      (Map Nombre Mago)       -- Magos por nombre
      (PriorityQueue Mago)    -- Magos por poder
  {- INV.REP.:
      * ??
  -}
```

  - ¿Cuáles deben ser los invariantes?

Tipos abstractos de datos

- Ejemplo: escuela de magia
  - Representación para la escuela de magia

```
data EscuelaDeMagia =
  EDM (Set Hechizo)          -- Todos los enseñados
      (Map Nombre Mago)      -- Magos por nombre
      (PriorityQueue Mago)   -- Magos por poder
  {- INV.REP.:
      * todos los magos del map están en la PQ y viceversa
      * todos los hechizos de cada mago están en el set
      * en la PQ no hay dos magos con el mismo nombre
  -}
```

Tipos abstractos

```
data EscuelaDeMagia =
  EDM (Set Hechizo)          -- Todos los enseñados
      (Map Nombre Mago)      -- Magos por nombre
      (PriorityQueue Mago)   -- Magos por poder
 {- INV.REP.:
    * todos los magos del map están en la cola y viceversa
    * todos los hechizos de cada mago están en el set
    * en la PQ no hay dos magos con el mismo nombre      -}
```

- TAD Escuela de Magia

```
fundarEscuela :: EscuelaDeMagia                    -- O(1)
fundarEscuela …

estaVacia :: EscuelaDeMagia -> Bool                -- O(1)
estaVacia …

magos :: EscuelaDeMagia -> [Nombre]                -- O(M)
magos …
```

Tipos abstractos

```
data EscuelaDeMagia =
  EDM (Set Hechizo)          -- Todos los enseñados
      (Map Nombre Mago)      -- Magos por nombre
      (PriorityQueue Mago)   -- Magos por poder
 {- INV.REP.:
    * todos los magos del map están en la cola y viceversa
    * todos los hechizos de cada mago están en el set
    * en la PQ no hay dos magos con el mismo nombre      -}
```

- TAD Escuela de Magia

```
fundarEscuela :: EscuelaDeMagia                    -- O(1)
fundarEscuela = EDM emptyS emptyM emptyPQ

estaVacia :: EscuelaDeMagia -> Bool                -- O(1)
estaVacia (EDM _ _ pqm) = isEmptyPQ pqm

magos :: EscuelaDeMagia -> [Nombre]                -- O(M)
magos (EDM _ mm _) = domM mm
```

  - ¿Podría usarse el map para ver si está vacía?
  - ¿Podría usarse la cola para obtener los magos?

Tipos abstractos

```
data EscuelaDeMagia =
  EDM (Set Hechizo)          -- Todos los enseñados
      (Map Nombre Mago)      -- Magos por nombre
      (PriorityQueue Mago)   -- Magos por poder
 {- INV.REP.:
    * todos los magos del map están en la cola y viceversa
    * todos los hechizos de cada mago están en el set
    * en la PQ no hay dos magos con el mismo nombre      -}
```

- TAD Escuela de Magia

```
registrar :: Nombre -> EscuelaDeMagia -> EscuelaDeMagia
registrar …                                          -- O(log M)
```

```
hechizosDe :: Nombre -> EscuelaDeMagia -> Set Hechizo
hechizosDe …                                         -- O(log M)
```

Tipos abstractos

```
data EscuelaDeMagia =
  EDM (Set Hechizo)          -- Todos los enseñados
      (Map Nombre Mago)      -- Magos por nombre
      (PriorityQueue Mago)   -- Magos por poder
 {- INV.REP.:
    * todos los magos del map están en la cola y viceversa
    * todos los hechizos de cada mago están en el set
    * en la PQ no hay dos magos con el mismo nombre      -}
```

- TAD Escuela de Magia

```
registrar :: Nombre -> EscuelaDeMagia -> EscuelaDeMagia
registrar n (EDM sh mm pqm) =                          -- O(log M)
    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
```

  - ¿Por qué buscarlo a ver si existe?
  - ¿Por qué insertarlo en el map y la cola?

Tipos abstractos

```
data EscuelaDeMagia =
  EDM (Set Hechizo)          -- Todos los enseñados
      (Map Nombre Mago)      -- Magos por nombre
      (PriorityQueue Mago)   -- Magos por poder
 {- INV.REP.:
    * todos los magos del map están en la cola y viceversa
    * todos los hechizos de cada mago están en el set
    * en la PQ no hay dos magos con el mismo nombre      -}
```

- TAD Escuela de Magia

```
hechizosDe :: Nombre -> EscuelaDeMagia -> Set Hechizo
hechizosDe n (EDM _ mm _) =                              -- O(log M)
      case (lookupM n mm) of
        Nothing -> error "Esa persona no es alumne de la escuela"
        Just m  -> hechizos m
```

Tipos abstractos

```
data EscuelaDeMagia =
  EDM (Set Hechizo)          -- Todos los enseñados
      (Map Nombre Mago)      -- Magos por nombre
      (PriorityQueue Mago)   -- Magos por poder
 {- INV.REP.:
    * todos los magos del map están en la cola y viceversa
    * todos los hechizos de cada mago están en el set
    * en la PQ no hay dos magos con el mismo nombre      -}
```

- TAD Escuela de Magia

```
leFaltaAprender :: Nombre -> EscuelaDeMagia -> Int
leFaltaAprender …                                  -- O(log M)
```

```
egresarUno :: EscuelaDeMagia -> (Mago, EscuelaDeMagia)
egresarUno …                                       -- O(log M)
```

Tipos abstractos

```
data EscuelaDeMagia =
  EDM (Set Hechizo)          -- Todos los enseñados
      (Map Nombre Mago)      -- Magos por nombre
      (PriorityQueue Mago)   -- Magos por poder
 {- INV.REP.:
    * todos los magos del map están en la cola y viceversa
    * todos los hechizos de cada mago están en el set
    * en la PQ no hay dos magos con el mismo nombre      -}
```

- TAD Escuela de Magia

```
leFaltaAprender :: Nombre -> EscuelaDeMagia -> Int
leFaltaAprender n (EDM sh mm _) =                  -- O(log M)
    case (lookupM n mm) of
      Nothing -> error "No es alumne de la escuela"
      Just m  -> sizeS sh - sizeS (hechizos m)

egresarUno :: EscuelaDeMagia -> (Mago, EscuelaDeMagia)
egresarUno (EDM sh mm pqm) =                       -- O(log M)
    let m = maxPQ pqm
     in (m, EDM sh (deleteM (nombre m) mm) (deleteMaxPQ pqm))
```

- ¿Cómo sabemos que son O(log M)?

Tipos abstractos

```
data EscuelaDeMagia =
  EDM (Set Hechizo)          -- Todos los enseñados
      (Map Nombre Mago)      -- Magos por nombre
      (PriorityQueue Mago)   -- Magos por poder
 {- INV.REP.:
    * todos los magos del map están en la cola y viceversa
    * todos los hechizos de cada mago están en el set
    * en la PQ no hay dos magos con el mismo nombre      -}
```

- TAD Escuela de Magia

```
enseñar :: Hechizo -> Nombre                          -- O(M log M
           -> EscuelaDeMagia -> EscuelaDeMagia        --   + log H)
enseñar …
```

Tipos abstractos

```
data EscuelaDeMagia =
  EDM (Set Hechizo)          -- Todos los enseñados
      (Map Nombre Mago)      -- Magos por nombre
      (PriorityQueue Mago)   -- Magos por poder
 {- INV.REP.:
    * todos los magos del map están en la cola y viceversa
    * todos los hechizos de cada mago están en el set
    * en la PQ no hay dos magos con el mismo nombre      -}
```

- TAD Escuela de Magia

```
enseñar :: Hechizo -> Nombre                          -- O(M log M
           -> EscuelaDeMagia -> EscuelaDeMagia        --    + log H)
enseñar h n (EDM sh mm pqm) =
  case (lookup 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)

modificarPQ :: Mago -> PriorityQueue Mago
                    -> PriorityQueue Mago          -- O(M log M)
```

Tipos abstractos

```
data EscuelaDeMagia =
  EDM (Set Hechizo)          -- Todos los enseñados
      (Map Nombre Mago)      -- Magos por nombre
      (PriorityQueue Mago)   -- Magos por poder
 {- INV.REP.:
    * todos los magos del map están en la cola y viceversa
    * todos los hechizos de cada mago están en el set
    * en la PQ no hay dos magos con el mismo nombre      -}
```

- TAD Escuela de Magia

```
modificarPQ :: Mago -> PriorityQueue Mago
                    -> PriorityQueue Mago          -- O(M log M)
-- PRECOND: hay un mago con el mismo nombre que el mago dado
modificarPQ m pqm =
  let maxM = maxPQ pqm
   in if m==maxM
       then insertPQ m (deleteMaxPQ pqm)
       else insertPQ maxM (modificarPQ m (deleteMaxPQ pqm))
```

- No hay otra forma de modificar un elemento en una cola

Tipos abstractos de

```
data EscuelaDeMagia

fundarEscuela   :: EscuelaDeMagia                                -- O(1)
estaVacia       :: EscuelaDeMagia -> Bool                        -- O(1)
registrar       :: Nombre -> EscuelaDeMagia -> EscuelaDeMagia    -- O(log M)
magos           :: EscuelaDeMagia -> [Nombre]                    -- O(M)
hechizosDe      :: Nombre -> EscuelaDeMagia -> Set Hechizo       -- O(log M)
leFaltaAprender :: Nombre -> EscuelaDeMagia -> Int               -- O(log M)
egresarUno      :: EscuelaDeMagia -> (Mago, EscuelaDeMagia)      -- O(log M)
enseñar         :: Hechizo -> Nombre                             -- O(M log M
                    -> EscuelaDeMagia -> EscuelaDeMagia          --   + log H)
```

- TAD Escuela de Magia
  - Como usuario

```
hechizosAprendidos :: EscuelaDeMagia -> Set Hechizo
 -- Propósito: Retorna todos los hechizos aprendidos por los magos.
 -- Eficiencia: O(M * (logM + H logH))
hechizosAprendidos …
```

Tipos abstractos de

```
data EscuelaDeMagia

fundarEscuela   :: EscuelaDeMagia                              -- O(1)
estaVacia       :: EscuelaDeMagia -> Bool                      -- O(1)
registrar       :: Nombre -> EscuelaDeMagia -> EscuelaDeMagia -- O(log M)
magos           :: EscuelaDeMagia -> [Nombre]                  -- O(M)
hechizosDe      :: Nombre -> EscuelaDeMagia -> Set Hechizo     -- O(log M)
leFaltaAprender :: Nombre -> EscuelaDeMagia -> Int             -- O(log M)
egresarUno      :: EscuelaDeMagia -> (Mago, EscuelaDeMagia)    -- O(log M)
enseñar         :: Hechizo -> Nombre                           -- O(M log M
                    -> EscuelaDeMagia -> EscuelaDeMagia        --   + log H)
```

- TAD Escuela de Magia
  - Como usuario

```
hechizosAprendidos :: EscuelaDeMagia -> Set Hechizo
 -- Propósito: Retorna todos los hechizos aprendidos por los magos.
 -- Eficiencia: O(M * (log M + H log H))
hechizosAprendidos escuela = hechizosDeEn (magos escuela) escuela

hechizosDeEn []     _       = emptyS
hechizosDeEn (n:ns) escuela = unionS (hechizosDe n escuela)
                                     (hechizosDeEn ns escuela)
```

  - Observar que hay DOS números para medir eficiencia

Tipos abstractos de

```
data EscuelaDeMagia

fundarEscuela   :: EscuelaDeMagia                                -- O(1)
estaVacia       :: EscuelaDeMagia -> Bool                        -- O(1)
registrar       :: Nombre -> EscuelaDeMagia -> EscuelaDeMagia    -- O(log M)
magos           :: EscuelaDeMagia -> [Nombre]                    -- O(M)
hechizosDe      :: Nombre -> EscuelaDeMagia -> Set Hechizo       -- O(log M)
leFaltaAprender :: Nombre -> EscuelaDeMagia -> Int               -- O(log M)
egresarUno      :: EscuelaDeMagia -> (Mago, EscuelaDeMagia)      -- O(log M)
enseñar         :: Hechizo -> Nombre                             -- O(M log M
                    -> EscuelaDeMagia -> EscuelaDeMagia          --   + log H)
```

- TAD Escuela de Magia
  - Como usuario

```
hayUnExperto :: EscuelaDeMagia -> Bool
 -- Propósito: Indica si existe un mago que sabe todos los hechizos
 --            enseñados por la escuela.
 -- Eficiencia: O(log M)
hayUnExperto …
```

Tipos abstractos de

```
data EscuelaDeMagia

fundarEscuela   :: EscuelaDeMagia                              -- O(1)
estaVacia       :: EscuelaDeMagia -> Bool                      -- O(1)
registrar       :: Nombre -> EscuelaDeMagia -> EscuelaDeMagia  -- O(log M)
magos           :: EscuelaDeMagia -> [Nombre]                  -- O(M)
hechizosDe      :: Nombre -> EscuelaDeMagia -> Set Hechizo     -- O(log M)
leFaltaAprender :: Nombre -> EscuelaDeMagia -> Int             -- O(log M)
egresarUno      :: EscuelaDeMagia -> (Mago, EscuelaDeMagia)    -- O(log M)
enseñar         :: Hechizo -> Nombre                           -- O(M log M
                    -> EscuelaDeMagia -> EscuelaDeMagia        --   + log H)
```

- TAD Escuela de Magia
  - Como usuario

```
hayUnExperto :: EscuelaDeMagia -> Bool
 -- Propósito: Indica si existe un mago que sabe todos los hechizos
 --            enseñados por la escuela.
 -- Eficiencia: O(logM)
hayUnExperto escuela = let m = fst (egresarUno escuela)
                       in leFaltaAprender (nombre m) escuela == 0
```

Tipos abstractos de

```
data EscuelaDeMagia

fundarEscuela   :: EscuelaDeMagia                              -- O(1)
estaVacia       :: EscuelaDeMagia -> Bool                      -- O(1)
registrar       :: Nombre -> EscuelaDeMagia -> EscuelaDeMagia -- O(log M)
magos           :: EscuelaDeMagia -> [Nombre]                  -- O(M)
hechizosDe      :: Nombre -> EscuelaDeMagia -> Set Hechizo     -- O(log M)
leFaltaAprender :: Nombre -> EscuelaDeMagia -> Int             -- O(log M)
egresarUno      :: EscuelaDeMagia -> (Mago, EscuelaDeMagia)    -- O(log M)
enseñar         :: Hechizo -> Nombre                           -- O(M log M
                    -> EscuelaDeMagia -> EscuelaDeMagia        --   + log H)
```

- TAD Escuela de Magia
  - Como usuario

```
egresarExpertos :: EscuelaDeMagia -> ([Mago], EscuelaDeMagia)
  -- Propósito: Devuelve un par con la lista de magos que saben todos
  --            los hechizos dados por la escuela y la escuela sin ellos.
  -- Eficiencia: O(M log M)
egresarExpertos …
```

Tipos abstractos de

```
data EscuelaDeMagia

fundarEscuela   :: EscuelaDeMagia                                -- O(1)
estaVacia       :: EscuelaDeMagia -> Bool                        -- O(1)
registrar       :: Nombre -> EscuelaDeMagia -> EscuelaDeMagia -- O(log M)
magos           :: EscuelaDeMagia -> [Nombre]                    -- O(M)
hechizosDe      :: Nombre -> EscuelaDeMagia -> Set Hechizo     -- O(log M)
leFaltaAprender :: Nombre -> EscuelaDeMagia -> Int             -- O(log M)
egresarUno      :: EscuelaDeMagia -> (Mago, EscuelaDeMagia)    -- O(log M)
enseñar         :: Hechizo -> Nombre                             -- O(M log M
                    -> EscuelaDeMagia -> EscuelaDeMagia        --   + log H)
```

- TAD Escuela de Magia
  - Como usuario

```
egresarExpertos :: EscuelaDeMagia -> ([Mago], EscuelaDeMagia)
  -- Propósito: Devuelve un par con la lista de magos que saben todos
  --            los hechizos dados por la escuela y la escuela sin ellos.
  -- Eficiencia: O(M logM)
egresarExpertos escuela =
         if not (hayUnExperto escuela)
          then ([], escuela)
          else let (m , escuelaSinM)  = egresarUno escuela
                   (ms, escuelaSinMs) = egresarExpertos escuelaSinM
               in (m:ms, escuelaSinMs)
```

Tipos abstractos de

```
type Hechizo = String
type Nombre = String

data Mago

crearM      :: Nombre -> Mago              -- O(1)
nombre      :: Mago -> Nombre              -- O(1)
aprender    :: Hechizo -> Mago -> Mago     -- O(log H)
hechizos    :: Mago -> Set Hechizo         -- O(1)
(==), (<=)  :: Mago -> Mago -> Bool        -- O(1)
   -- Iguales, mismo nombre; menor, más hechizos
```

- TAD Mago
  - Implementación

```
data Mago = M Nombre (Set Hechizo)

crearM n              = M n emptyS
nombre (M n _)        = n
aprender h (M n hs)   = M n (addS h hs)
hechizos (M n hs)     = hs

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

instance Ord Mago where
  (M _ hs1) <= (M _ hs2) = sizeS hs1 >= sizeS hs2
```

Resumen

Resumen

- Al implementar un TAD específico
  - Pueden usarse otros TADs
    - Estándar (Set, Map, etc.)
    - Específicos (Mago, etc.)
- Distinguir entre ser usuario y ser implementador
  - Prestar atención a los invariantes
- Al medir eficiencia
  - Puede haber más de una medida a considerar
  - Deben combinarse adecuadamente