Estructuras de datos

Clases teóricas

por Pablo E. “Fidel” Martínez López

6. Tipos Abstractos de Datos II

Multisets, Maps y Priority Queues

Repaso

Tipos abstractos de datos

- Los tipos abstractos de datos (TADs)
  - Quedan definidos por su ***interfaz***
  - Inducen 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
    - Con una medición gruesa (sin detalles)
  - Clasificación
    - Constante, siempre el mismo costo, O(1)
    - Lineal, solo operaciones constantes por elemento, O(n)
    - Cuadrática, hasta operaciones lineales por elemento, O(n^2)

Tipos abstractos de datos

- Existen TADs clásicos que hay que conocer
  - Stacks, Queues, Sets
- 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

TADs clásicos: Stacks

- Ejemplo: TAD Stack

```
module Stack (Stack, emptyS, isEmptyS, push
                   , top, pop, lenS) where

data Stack a

emptyS   :: Stack a
isEmptyS :: Stack a -> Bool
push     :: a -> Stack a -> Stack a
top      :: Stack a -> a
pop      :: Stack a -> Stack a
lenS     :: Stack a -> Int
```

TADs clásicos: Stacks

```
data Stack a

emptyS   :: Stack a
isEmptyS :: Stack a -> Bool
push     :: a -> Stack a -> Stack a
top      :: Stack a -> a
pop      :: Stack a -> Stack a
lenS     :: Stack a -> Int
```

- Ejemplo: TAD Stack
- Implementaciones posibles
  - Lista
    - Operaciones constantes, O(1) (salvo **`lenS`**)
- Ejemplos de uso
  - Balanceo de paréntesis
  - Ejecución de código con subtareas

TADs clásicos: Queues

- Ejemplo: TAD Queue

```
module Queue (Queue, emptyQ, isEmptyQ, enqueue
                   , firstQ, dequeue) where

data Queue a

emptyQ   :: Queue a
isEmptyQ :: Queue a -> Bool
enqueue  :: a -> Queue a -> Queue a
firstQ   :: Queue a -> a
dequeue  :: Queue a -> Queue a
```

TADs clásicos: Queues

```
data Queue a

emptyQ   :: Queue a
isEmptyQ :: Queue a -> Bool
enqueue  :: a -> Queue a -> Queue a
firstQ   :: Queue a -> a
dequeue  :: Queue a -> Queue a
```

- Ejemplo: TAD Queue
- Implementaciones posibles
  - Lista simple (agregando adelante)
  - Lista simple (agregando atrás)
  - Dos listas (frente y fondo)
    (con invariante de representación)
- Es interesante observar cómo varía la eficiencia de las operaciones de usuario al cambiar de implementación

TADs clásicos: Sets

- Ejemplo: TAD Set

```
module Set (Set, emptyS, addS, belongs, sizeS
                , removeS, unionS, set2list) where

data Set a

emptyS   :: Set a
addS     :: Eq a => a -> Set a -> Set a
belongs  :: Eq a => a -> Set a -> Bool
sizeS    :: Eq a => Set a -> Int
removeS  :: Eq a => a -> Set a -> Set a
unionS   :: Eq a => Set a -> Set a -> Set a
set2list :: Eq a => Set a -> [a]
```

TADs clásicos: Sets

```
data Set a

emptyS   :: Set a
addS     :: Eq a => a -> Set a -> Set a
belongs  :: Eq a => a -> Set a -> Bool
sizeS    :: Eq a => Set a -> Int
removeS  :: Eq a => a -> Set a -> Set a
unionS   :: Eq a => Set a -> Set a -> Set a
set2list :: Eq a => Set a -> [a]
```

- Ejemplo: TAD Set
- Implementaciones posibles
  - Lista sin repetidos (requiere invariante)
  - Lista con repetidos
- Es interesante observar cómo varía la eficiencia de las operaciones de usuario al cambiar de implementación

Algunas consideraciones sobre eficiencia

Eficiencia

- La eficiencia se mide en base al número de elementos de la estructura
- Ese número habitualmente se lo denomina “n”, y así O(n) es el orden lineal
- Pero, ¿qué mide exactamente ese número?
  - Debe tenerse esto en cuenta al calcular costos
  - Si no, los resultados pueden ser engañosos
  - Ej: n no es lo mismo en Set con y sin repetidos…

Más sobre implementación de TADs

TADs clásicos: Queues

```
data Queue a

emptyQ   :: Queue a
isEmptyQ :: Queue a -> Bool
enqueue  :: a -> Queue a -> Queue a
firstQ   :: Queue a -> a
dequeue  :: Queue a -> Queue a
```

- Ejemplo: TAD Queue
- ¿Cómo calcular la operación **`lenQ`** como usuario?

```
lenQ :: Queue a -> Int
lenQ q = if isEmptyQ q
           then 0
           else 1 + lenQ (dequeue q)
```

  - ¿Qué eficiencia tiene?
    - Si **`dequeue`** es O(1)...

TADs clásicos: Queues

```
data Queue a

emptyQ   :: Queue a
isEmptyQ :: Queue a -> Bool
enqueue  :: a -> Queue a -> Queue a
firstQ   :: Queue a -> a
dequeue  :: Queue a -> Queue a
```

- Ejemplo: TAD Queue
- ¿Cómo calcular la operación **`lenQ`** como usuario?

```
lenQ :: Queue a -> Int
lenQ q = if isEmptyQ q
           then 0
           else 1 + lenQ (dequeue q)
```

  - ¿Qué eficiencia tiene?
    - Si **`dequeue`** es O(1)... Es O(n)
  - ¿Se puede mejorar?

TADs clásicos: Queues

```
data Queue a

emptyQ   :: Queue a
isEmptyQ :: Queue a -> Bool
enqueue  :: a -> Queue a -> Queue a
firstQ   :: Queue a -> a
dequeue  :: Queue a -> Queue a
lenQ     :: Queue a -> Int
```

- Ejemplo: TAD Queue extendido
  - ¿Y si **`lenQ`** estuviese en la interfaz, podría ser mejor?
    - No, sin cambiar la representación…
    - Usando esta representación, sí

```
data Queue a = Q [a] Int
  {- INV.REP.: en (Q xs n), n es la longitud de xs -}
```

  - ¿Qué eficiencia tiene?
    - Puede conseguirse **`lenQ`** en O(1), sin alterar el resto
    - También puede usarse para otras implementaciones

Más TADs clásicos: Priority Queues

TADs clásicos: Priority Queues

- TAD Priorty Queue

```
module PriorityQueue(PriorityQueue, emptyPQ, isEmptyPQ
                     , insertPQ, findMinPQ, deleteMinPQ) where

data PriorityQueue a

emptyPQ     :: PriorityQueue a
isEmptyPQ   :: PriorityQueue a -> Bool
insertPQ    :: Ord a => a -> PriorityQueue a -> PriorityQueue a
findMinPQ   :: Ord a => PriorityQueue a -> a
deleteMinPQ :: Ord a => PriorityQueue a -> PriorityQueue a
```

  - Siempre sale el mínimo elemento (el de máxima prioridad)

TADs clásicos: Priority Q

- TAD Priorty Queue
- Ejemplos de uso
  - Sala de espera de hospital
    - El paciente más grave tiene más prioridad
  - *Schedulers* de sistemas operativos
    - Los procesos cambian su prioridad según varios factores (e.g. tiempo sin ser atendidos, duración, urgencia)

```
module PriorityQueue
  (PioirityQueue, emptyPQ, isEmptyPQ
 , insertPQ, findMinPQ, deleteMinPQ) where

data PriorityQueue a

emptyPQ     :: PriorityQueue a
isEmptyPQ   :: PriorityQueue a -> Bool
insertPQ    :: Ord a => a -> PriorityQueue a
                        -> PriorityQueue a
findMinPQ   :: Ord a => PriorityQueue a -> a
deleteMinPQ :: Ord a => PriorityQueue a
                     -> PriorityQueue a
```

TADs clásicos: Priority Q

```
module PriorityQueue
  (PioritytQueue, emptyPQ, isEmptyPQ
 , insertPQ, findMinPQ, deleteMinPQ) where
data PriorityQueue a
emptyPQ      :: PriorityQueue a
isEmptyPQ    :: PriorityQueue a -> Bool
insertPQ     :: Ord a => a -> PriorityQueue a
                         -> PriorityQueue a
findMinPQ    :: Ord a => PriorityQueue a -> a
deleteMinPQ :: Ord a => PriorityQueue a
                      -> PriorityQueue a
```

- Ejemplo: TAD Priorty Queue
- Implementaciones básicas
  - Con listas arbitrarias
    - el costo de **`findMinPQ`** y **`deleteMinPQ`** es O(n)
  - Con listas ordenadas
    - el costo de **`insertPQ`** es O(n)
  - ¿Se puede mejorar?
    - Se requieren otras estructuras… (árboles)

Más TADs clásicos: Maps (o diccionarios)

TADs clásicos: Maps

- TAD Map (abreviatura de Mapping, *asociación*)

```
module Map(Map, emptyM, assocM, lookupM
                , deleteM, keys) where
data Map key value   -- clave, valor
emptyM  :: Map k v
assocM  :: Ord k => k -> v -> Map k v -> Map k v
lookupM :: Ord k => k -> Map k v -> Maybe v
deleteM :: Ord k => k -> Map k v -> Map k v
domM    :: Ord k => Map k v -> [k]
```

  - Cada clave se asocia a un valor

TADs clásicos: Maps

```
module Map(Map, emptyM, assocM, lookupM
                , deleteM, keys) where

data Map k v

emptyM  :: Map k v
assocM  :: Eq k => k -> v -> Map k v -> Map k v
lookupM :: Eq k => k -> Map k v -> Maybe v
deleteM :: Eq k => k -> Map k v -> Map k v
keys    :: Eq k => Map k v -> [k]
```

- TAD Map
- Ejemplos de uso
  - Agenda
  - Diccionario
  - Diccionario de sinónimos (*theasurus*)
  - Memorias asociativas
  - En general, toda aplicación donde se requiere obtener información a partir de una clave

TADs clásicos: Maps

```
module Map(Map, emptyM, assocM, lookupM
                , deleteM, keys) where

data Map k v

emptyM  :: Map k v
assocM  :: Ord k => k -> v -> Map k v -> Map k v
lookupM :: Ord k => k -> Map k v -> Maybe v
deleteM :: Ord k => k -> Map k v -> Map k v
domM    :: Ord k => Map k v -> [k]
```

- TAD Map
- Implementaciones básicas
  - Con lista de pares (clave, valor)
    - Operaciones de O(n) y otras O(n^2) (OJO al n)
  - Con lista de pares (clave, valor), sin claves repetidas
    - Operaciones de O(n)
    - Alternativa: 2 listas, de claves y de valores, de igual longitud
  - ¿Se puede mejorar?
    - Se requieren otras estructuras… (árboles)

Más TADs clásicos: Multisets

TADs clásicos: Multisets

- TAD Multiset (o *bag*)

```
module Multiset(Multiset, emptyMS, addMS, ocurrencesMS
                , unionMS, intersectionMS, multiSet2List) where
data Multiset a
emptyMS       :: MultiSet a
addMS         :: Ord a => a -> MultiSet a -> MultiSet a
occurencesMS  :: Ord a => a -> MultiSet a -> Int
unionMS       :: Ord a => MultiSet a -> MultiSet a -> MultiSet a
intersectMS   :: Ord a => MultiSet a -> MultiSet a -> MultiSet a
ms2list       :: Multiset a -> [(a,Int)]
```

  - Cada elemento puede aparecer varias veces

TADs clásicos: Multisets

- TAD Multiset
- Implementaciones
  - Con lista de elementos
  - Con un map de elementos a Ints

Resumen

Resumen

- Más TADs clásicos
  - Priority Queues
  - Maps (o diccionarios)
  - Multisets
- Variaciones de implementaciones lineales
- Uso de un TAD para implementar otro
- Práctica de cálculo de costos
- Algunos consejos útiles para implementar mejor