TAD-Roulette-Resuelto.pdf

jmp__0807

Estructuras de Datos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

Que no te escriban poemas de amor cuando terminen la carrera ►►►►►►

(a nosotros por suerte nos pasa) ☺

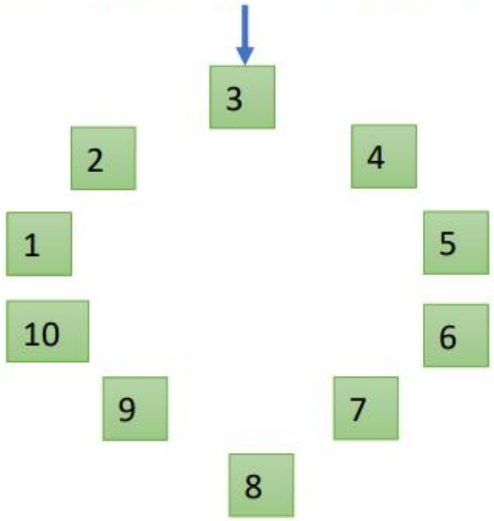
WUOLAH

TAD Roulette

WUOLAH

Especificación informal

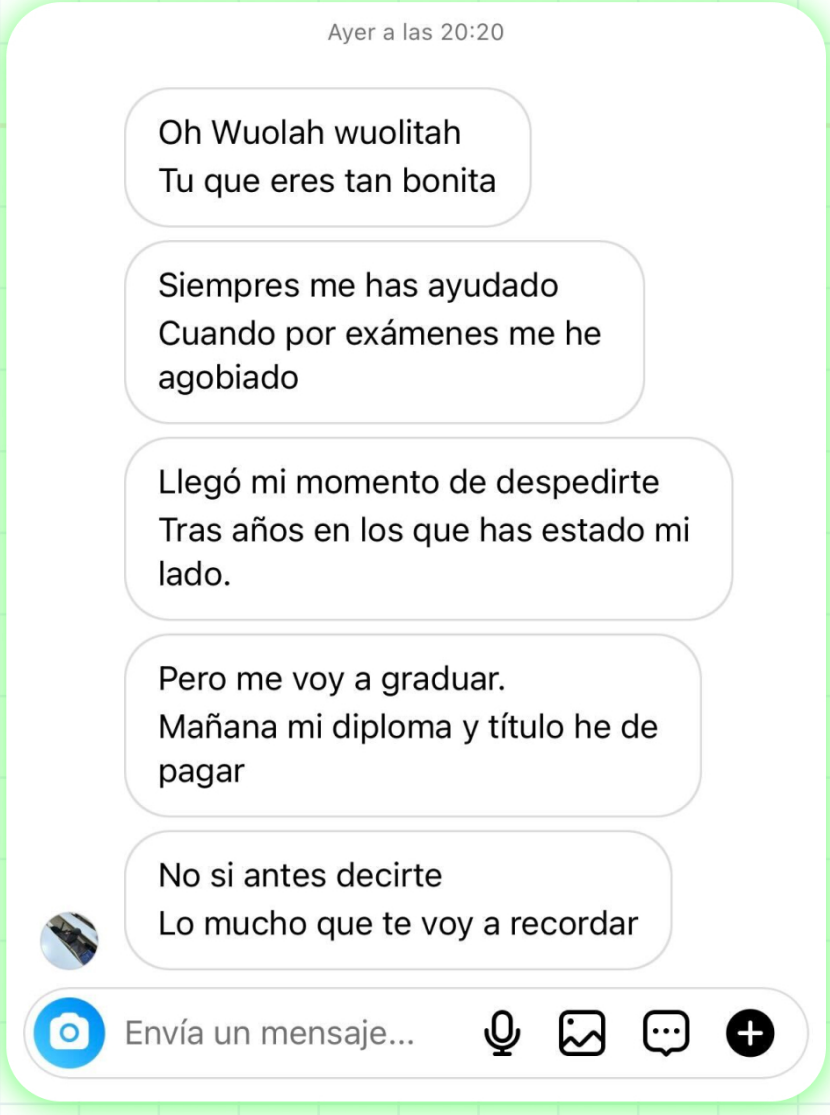
- Una ruleta es una estructura circular de elementos donde hay uno destacado señalado con un puntero.



En este caso, la ruleta contiene los elementos del 1 al 10 en sentido horario y el elemento destacado es el 3.

Que no te escriban poemas de amor cuando terminen la carrera 🚩🚩🚩🚩🚩🚩

(a nosotros por suerte nos pasa)



WUOLAH

Especificación informal de una Roulette

Las operaciones disponibles para una ruleta son:

- Saber si está vacía o no.
- Devolver el elemento destacado.
- Girar n posiciones la ruleta en sentido horario si n es positivo o antihorario si n es negativo.
- Borrar el elemento destacado. EL destacado pasa a ser el siguiente en sentido horario.
- Añadir un nuevo elemento delante del destacado que pasa a ser el destacado.
- Aplicar una función a cada elemento de la ruleta quedando los resultados como elementos de una nueva ruleta.
- Pasar de una lista a una ruleta.
- Pasar de una ruleta a una lista.
- Crea una ruleta vacía.

Que no te escriban poemas de amor
cuando terminen la carrera ►►►►►►

(a nosotros por
suerte nos pasa)

WUOLAH

Especificación formal de una Roulette

```
empty     :: Roulette a
isEmpty   :: Roulette a -> Bool
sign      :: Roulette a -> a
turn      :: Integer -> Roulette a -> Roulette a
delete    :: Roulette a -> Roulette a
insert    :: a -> Roulette a -> Roulette a
mapRoulette :: (a -> b) -> Roulette a -> Roulette b
listToRoulette :: [a] -> Roulette a
rouletteToList :: Roulette a -> [a]
```

Implementación del TAD Roulette

- Vamos a implementar la Roulette con una cola (Queue) y un entero que guarda el tamaño de la ruleta:
  - Los elementos en sentido horario de la ruleta son los sucesivos elementos de la cola
  - En la cabeza de la cola estará el elemento destacado.
- INVARIANTES:
  - Siempre hay un elemento destacado (salvo en la ruleta vacía)
  - Siempre mantendremos el elemento destacado en la cabeza de la cola.

```
data Roulette a = R (Q.Queue a) Integer deriving Eq
```

Ejemplo

```
data Roulette a = R (Q.Queue a) Integer deriving Eq
```

- Ejemplo.
  - La ruleta anterior se representará por

    `R LinearQueue(3,4,5,6,7,8,9,10,1,2) 10`

Su show será `QueueRoulette:10(3,4,5,6,7,8,9,10,1,2)`

- NOTA: Todas las operaciones con colas estarán cualificadas con Q:

  `Q.Queue, Q.enqueue, Q.first, etc.`

```haskell
-- Javier Montes Perez TAD Roulette

import Data.List(intercalate)
import qualified DataStructures.Queue.LinearQueue as Q
import Test.QuickCheck

data Roulette a = R (Q.Queue a) Integer deriving Eq

sample1 = R (foldl (flip Q.enqueue) Q.empty [3,4,5,6,7,8,9,10,1,2]) 10


-- Ejercicio 1 (0.05 ptos.)
-- Crea una ruleta vacia
empty :: Roulette a
empty = R Q.empty 0 --tamaño 0 --

-- Ejercicio 2 (0.05 ptos.)
-- Determina si una ruleta está vacia
isEmpty :: Roulette a -> Bool
isEmpty (R q size) = size == 0

-- Ejercicio 3 (0.10 ptos)
-- devuelve el dato apuntado
sign :: Roulette a -> a
sign (R q _) = Q.first q

-- Ejercicio 4 (0.20 ptos)
-- turn gira la ruleta un determinado numero de elementos

turn :: Integer -> Roulette a -> Roulette a
turn n (R q size)
    | size == 0 || modu == 0 = (R q size)
    | otherwise = R (aux q modu) size
        where
            modu = (mod n size)
            aux q 0 = q
            aux q s | s > 0 = aux (Q.enqueue (Q.first q) (Q.dequeue q)) (s-1)

-- Ejercicio 5 (0.10 ptos)
-- elimina el elemento situado en la posicion del puntero y coloca el puntero en la siguiente pos
delete :: Roulette a -> Roulette a
delete (R q size) = R (Q.dequeue q) (size-1)

-- Ejercicio 6 (0.15 ptos)
-- inserta el elemento en la posicion del puntero y corre el resto en sentido horario
insert :: a -> Roulette a -> Roulette a
insert x (R q size) = turn size (R (Q.enqueue x q) (size+1))

-- Ejercicio 7 (0.15 ptos)
-- genera una ruleta con los objetos de la lista situados en orden horario y con el puntero apuntado ...
listToRoulette :: [a] -> Roulette a
listToRoulette ls = foldr (\x solResto -> insert x solResto) (R Q.empty 0) ls

-- Ejercicio 8 (0.15 ptos)
```

```haskell
-- genera una lista con los elementos de una ruleta. El primero será el apuntado por el
-- puntero y luego irán los elementos en sentido horario
rouletteToList :: Roulette a -> [a]
rouletteToList (R q size) = mkList q
    where mkList q
            | Q.isEmpty q = []
            | otherwise = [Q.first q] ++ mkList (Q.dequeue q)

-- Ejercicio 9 (0.20 ptos)
-- mapRoulette toma una funcion de a -> b y se la aplica a todos los elementos de la ruleta
-- quedando la ruleta en la misma posicion

mapRoulette :: (a -> b) -> Roulette a -> Roulette b
mapRoulette f r@(R q size) = listToRoulette (map f (rouletteToList r))

-- Ejercicio 9 (0.10 ptos)
-- probar con quickCheck que para cualquier n y cualquier ruleta girar n a la derecha y luego
-- n a la izquierda produce la misma ruleta. Las ruletas son Arbitray por lo que pueden
-- aparecer como argumentos de una propiedad

p1 :: Integer -> Roulette Integer -> Bool
p1 n r = turn n (turn (-n) r) == turn (-n)(turn n r)


-- Showing a roulette
instance (Show a) => Show (Roulette a) where
  show (R q size) = "QueueRoulette:"++show size++"(" ++ (intercalate "," (aux q)) ++ ")"
    where
    aux q1
        | Q.isEmpty q1 = []
        | otherwise = show x : aux q'
            where
                x = Q.first q1
                q' = Q.dequeue q1

-- Arbitrary instance
instance Arbitrary a => Arbitrary (Roulette a) where
    arbitrary = do
        xs <- listOf arbitrary
        return (foldr insert empty xs)
```


si lees esto me debes un besito