Estructura de Datos (2021-22, Grupo A)

Enunciado

Importante Al final del examen deberás subir solamente como ficheros **no comprimidos** los siguientes: `Seq.hs`, `LinkedSeq.java`, `BinaryTree.hs`, `BinaryTree.java`, `DiGraphDftTimer.hs` y `DiGraphDftTimer.java`. Si renuncias a la evaluación continua, también debes subir `WBinTree.hs` y `WBinTree.java`. Asegúrate de que todos los ficheros subidos realmente corresponden con tus soluciones, incluyen tu nombre y DNI en el comentario inicial y compilan sin error. Solo se corregirán ficheros sin errores de compilación.

Ejercicio 1

Sumar un número de un único dígito a una secuencia que representa un número.

Haskell (1,25 puntos). Fichero `Seq.hs`

Dado un número `k` de un único dígito (0-9) y una secuencia cuyos nodos almacenan los dígitos de un entero no negativo, el objetivo del ejercicio es sumar `k` al número representado por la secuencia (trabajando directamente con la secuencia y **sin invertirla**).

El tipo de datos para representar la secuencia es el siguiente:

```
data Seq a = Empty | Node a (Seq a) deriving (Eq, Show)
```

Por ejemplo, la secuencia `Node 9 (Node 9 (Node 9 (Node 3 Empty)))` representa el número 9993. Si a esa secuencia le sumamos 7 (número de un único dígito) el resultado tiene que ser la secuencia `Node 1 (Node 0 (Node 0 (Node 0 Empty)))` que representa el número 10000.

La función que hay que implementar es la siguiente

```
addSingleDigit :: (Integral a) => a -> Seq a -> Seq a
```

que se encuentra en el fichero `Seq.hs`. El fichero `SeqDemo.hs` contiene una prueba de la función.

Java (1,25 puntos). Fichero `LinkedSeq.java`

Dado un número `k` de un único dígito (0-9) y una secuencia enlazada cuyos nodos almacenan los dígitos de un entero no negativo, el objetivo del ejercicio es sumar `k` al número representado por la secuencia (trabajando directamente con la secuencia y **sin invertirla**).

La clase para representar la secuencia enlazada es `LinkedSeq<T>`, y se proporciona con el material del examen.

Por ejemplo, la secuencia enlazada `9 -> 9 -> 9 -> 3 -> null` representa el número 9993. Si a esa secuencia le sumamos 7 (número de un único dígito), la secuencia enlazada se transforma en `1 -> 0 -> 0 -> 0 -> null`, que se corresponde con el número 10000.

El método que hay que implementar es el siguiente:

```
public static void addSingleDigit(int d, LinkedSeq<Integer> linkedSeq)
```

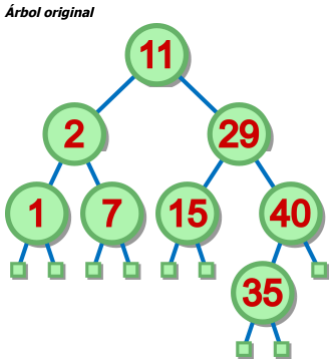
en la clase `LinkedSeq` del paquete `exercises`. La clase `LinkedSeqDemo` contiene una prueba.

Ejercicio 2

Subárboles que tienen todos sus nodos en un rango de valores.

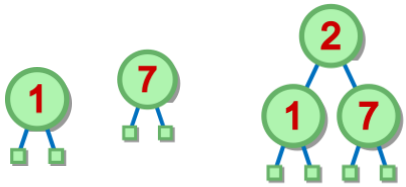
Dado un árbol binario de búsqueda (BST) `t` y dos valores `min` y `max`, el objetivo del ejercicio es devolver cuántos subárboles de `t` tienen todos sus nodos en el intervalo `[min,max]`. ***La solución debe tener una complejidad lineal*** respecto al número de nodos del árbol. Para ello, se propone seguir una estrategia *bottom-up* que transfiere información de los hijos al nodo padre. De este modo, podemos decir que un árbol tiene todos los nodos en el rango si su raíz está en el rango y tanto el subárbol izquierdo como el derecho están dentro del rango también.

Árbol original



Por ejemplo, *el árbol de la figura tiene 3 subárboles en el rango [0, 10]*, son los que se muestran a continuación:

Subárbol 1 Subárbol 2 ***Subárbol 3***



Haskell(1,25 puntos). Fichero BinaryTree.hs

Implementa la siguiente función del módulo `BinaryTree`

```
subTreesInRange :: (Ord a) => BinaryTree a -> a -> a -> Integer
```

que toma un árbol binario que cumple la propiedad de ordenación BST y dos valores (`min` y `max`) y devuelve el número de subárboles que tienen todos sus nodos en el rango de valores `[min,max]`. Se puede asumir que `min < max`. El fichero `BinaryTreeDemo.hs` permite probar dicha función sobre el árbol de ejemplo.

Java (1,25 puntos). Fichero BinaryTree.java

Implementa el siguiente método de la clase `BinaryTree<T>` del paquete `exercises`

```
public int subTreesInRange(T min, T max)
```

que devuelve cuántos subárboles tienen todos sus nodos en el rango `[min,max]`.

El método principal de la clase `BinaryTreeDemo` permite probar el método sobre el árbol de ejemplo.

Ejercicio 3

Tiempo de llegada y salida de los vértices en un recorrido en profundidad.

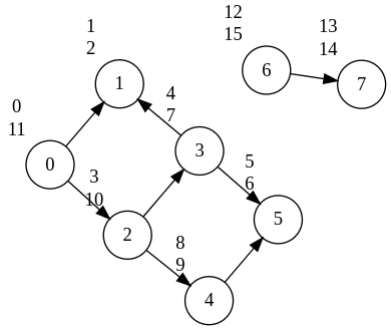
Haskell (1,25 puntos). Fichero DiGraphDftTimer.hs

Dado un grafo dirigido, el ejercicio consiste en encontrar el *tiempo de llegada* y *el tiempo de salida* de sus vértices en un recorrido en profundidad (DFT).

El *tiempo de llegada* es el instante en el que el vértice es explorado por primer vez por el recorrido DTF, mientras que el *tiempo de salida* es el instante en el que se ha explorado todos los sucesores del vértice y el algoritmo DFT se dispone a hacer *backtracking* para explorar otro nodo. Los instantes de tiempo se definen como una secuencia de enteros (0 es el instante en el que se visita el primer nodo, 1 cuando se visita el segundo y así sucesivamente).

Por ejemplo, en el grafo de la figura se han anotado los tiempos de llegada y salida junto a la esquina superior izquierda de cada vértice para un *posible recorrido DFT* en el que el nodo inicial es el 0.

Grafo con los tiempos de salida y llegada anotados



Hay que implementar la siguiente función en el módulo `DiGraphDftTimer`:

```
diGraphDftTimer :: (Ord v) => DiGraph v -> (Dictionary v Int, Dictionary v Int)
```

que devuelve un tupla con dos diccionarios. El primer diccionario asocia a cada vértice su tiempo de llegada, y el segundo asocia cada vértice con su tiempo de salida.

El módulo `DiGraphDftTimerDemo` contiene una prueba de dicha función.

Java (1,25 puntos). Fichero DiGraphDftTimer.java

Dado un grafo dirigido, el ejercicio consiste en encontrar el *tiempo de llegada* y *el tiempo de salida* de sus vértices en un recorrido en profundidad (DFT).

El *tiempo de llegada* es el instante en el que el vértice es explorado por primer vez por el recorrido DTF, mientras que el *tiempo de salida* es el instante en el que se ha explorado todos los sucesores del vértice y el algoritmo DFT se dispone a hacer *backtracking* para explorar otro nodo. Los instantes de tiempo se definen como una secuencia de enteros (0 es el instante en el que se visita el primer nodo, 1 cuando se visita el segundo y así sucesivamente).

Hay que completar la implementación de la clase `DiGraphDftTimer<V>` en el paquete `exercises`, de tal manera que el diccionario `arrivalD` relacione cada vértice con su tiempo de llegada y el diccionario `departureD` con su tiempo de salida.

La clase `DiGraphDftTimerDemo` contiene una prueba de la clase.

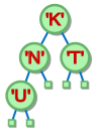
Ejercicio 4 (Solo para los que renuncian a la evaluación continua)

Árboles binarios equilibrados en peso.

Se llama *peso* de un árbol al número de nodos que lo componen. Un árbol binario (no necesariamente de búsqueda) está equilibrado en peso si satisface el siguiente invariante: en cada nodo del árbol el peso del subárbol izquierdo es igual o a lo sumo uno más que el peso del subárbol derecho.

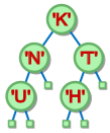
Por ejemplo, el árbol de la figura es un árbol equilibrado en peso:

Árbol antes



Si insertamos `H`, al mantener el invariante obtendremos el árbol equilibrado en peso:

Árbol después



Haskell (0,50 + 0,75 puntos). Fichero WBinTree.hs

Representaremos árboles binarios equilibrados en peso con el tipo `WBinTree` del módulo `WBinTree`. Completa la definición de las siguientes funciones:

```
isWeightBalanced :: WBinTree a -> Bool
```

que determina si un árbol binario está equilibrado en peso.

```
insert :: a -> WBinTree a -> WBinTree a
```

que inserta un dato en un árbol binario equilibrado en peso manteniendo el invariante.

El módulo `WBinTreeDemo` contiene una prueba de las funciones.

Java (0,50 + 0,75 puntos). Fichero WBinTree.java

Representaremos árboles binarios equilibrados en peso con la clase `WBinTree<T>` del paquete `exercises`. Completa la definición de los métodos:

```
public boolean isWeightBalanced()
```

que determina si un árbol binario está equilibrado en peso.

```
public void insert(T x)
```

que inserta un dato en un árbol binario equilibrado en peso manteniendo el invariante.

La clase `WBinTreeDemo` contiene una prueba de los métodos.

Última modificación: lunes, 25 de julio de 2022, 10:29

◄ Calificaciones Saltar a... Examen resuelto ►