Programación y Algoritmos I
Tarea 9: Arboles rojos-negros

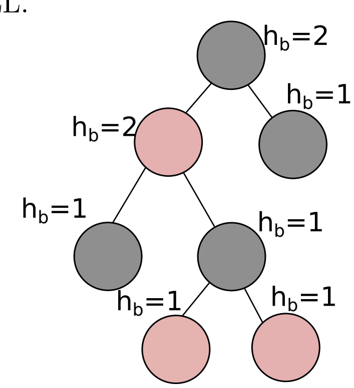
Nota 1: Como la tarea anterior, esta tarea es *colaborativa* (en pares y con un repositorio de git). Un medio punto se atribuirá en función del bueno uso de git entre ambos participantes (contribuciones equilibradas y revisiones de uno para el otro).

Nota 2: Se atribuirá otro medio punto en la elaboración de tests (por ejemplo con las funciones assert); la idea es que cada uno en el equipo pruebe (con tests) la robustez del código del otro.

En la clase vimos que una clase de arbóles binarios de búsqueda (ABBs) balanceados son los **arboles AVL**. En esta tarea, estudiaremos otra clase de ABBs llamados **arboles rojos-negros**. Un árbol rojo-negro es un ABB que tiene las siguientes restricciones:

- Cada nodo tiene un elemento (un **label**) llamado **color**, que es o rojo, o negro.
- La raíz del ABB es siempre negra.
- Los hijos de un nodo rojo *tienen que ser ambos negros* (o sea, no puede haber two rojos consecutivos en un camino de la raíz a una hoja).
- Las ligas vacías (apuntadores NULL en nodos terminales) cuentan como **negro**.
- Cualquier camino de un nodo v del arbol hacia un NULL tiene el mismo número de nodos negros (sin contar v). Ese número se llama **altura negra** de v y lo notaremos $h_b(v)$.

En la figura siguiente, se ilustra un arbol rojo-negro con los valores de $h_b(v)$ correspondientes. No se ha representado los apuntadores NULL.



Pregunta 1 [0.5 puntos]

Demostrar que si r es la raíz de un árbol rojo-negro de altura h, tenemos

$$h_b(r) \geq \frac{h}{2}.$$

Pregunta 2 [0.5 puntos]

Demostrar por inducción sobre la altura de los nodos que un subarbol enraizado en un nodo v tiene al menos $2^{h_b(v)} - 1$ nodos internos.

Pregunta 3 [0.5 puntos]

Deducir de lo anterior que, si n es el número total de nodos, la altura h del arbol satisface:

$$h \leq 2\log_2(n+1).$$

Pregunta 4 [0.5 puntos]

Definimos la inserción de un nuevo dato en un arbol rojo-negro como sigue: Insertamos el nuevo nodo w como en un ABB normal (bajando hacia su lugar, por búsqueda) y lo coloreamos como **rojo**. Si ese nodo es la raíz (w fue el primer nodo), lo coloreamos como negro. Mostrar que el único caso en que se puede generar una violación de las reglas de arbol rojo-negro es cuando el padre de w es **rojo**.

Pregunta 5 [0.5 puntos]

Mostrar que, en el caso anterior de violación, si el nodo tío de w (es decir, el otro hijo de su abuelo) es *también rojo*, hay una correción muy simple que se puede hacer al *cambiar de colores el abuelo, el papa y el tío*. Cómo cambia la altura negra de los nodos del árbol con esta correción? Mostrar que la correción puede provocar una violación al nivel el abuelo. Cuál es la complejidad total de la correción?

Pregunta 6 [0.5 puntos]

Mostrar que en el otro caso (si el tío es negro), se puede usar las mismas **rotaciones** que vimos en el caso de árboles AVL para corregir el arbol rojo-negro. Cómo cambia la altura negra de los nodos del árbol con esta correción? Cuál es la complejidad de esta correción?

Pregunta 7 [6 puntos]

Implementar la estructura de árbol rojo-negro, y las funciones correspondiendo a la API de tabla de símbolo:

```
void put(RBTree *s, int key, int val); // Insert a pair key/val in the ST
int get(RBTree *s,int key); // Gets the data associated to key
int contains(RBTree *s,int key); // Is there a data paired with key?
void delete(RBTree *s,int key); // Remove a key/value from the ST
int isEmpty(RBTree *s); // Is the table empty?
int size(RBTree *s);
```

e implementar un ejemplo que muestre que su implementación funcione bien.