Práctica adicional - Árboles

Árboles Binarios

1. Considere las siguientes definiciones:

- Un árbol binario *lleno* es aquel que tiene todos sus niveles llenos. Es decir, cada nivel tiene todos los nodos que puede alojar.
- Un árbol binario *completo* es aquel donde sus niveles tienen todos los nodos posibles, excepto quizás el último nivel, que se encuentra lleno de izquierda a derecha.
- Un árbol binario puede estar *balanceado* según distintos criterios de balanceo, como ser altura o cantidad de nodos, entre otros criterios posibles.

  Un árbol binario se dice *balanceado en cantidad* si:

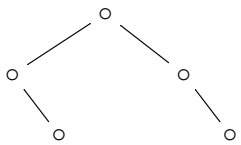
  - la cantidad de nodos en el subárbol izquierdo difiere a lo sumo en uno de la cantidad de nodos en el subárbol derecho, y
  - el subárbol izquierdo es *balanceado en cantidad*, y
  - el subárbol derecho es *balanceado en cantidad*.

  Un árbol binario se dice *balanceado en altura* si:

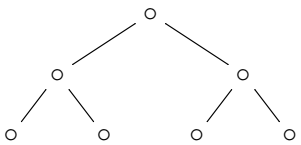
  - la altura del subárbol izquierdo difiere a lo sumo en uno de la altura del subárbol derecho, y
  - el subárbol izquierdo es *balanceado en altura*, y
  - el subárbol derecho es *balanceado en altura*.

Clasifique los árboles de las siguientes figuras en: *lleno, completo, balanceado en cantidad, balanceado en altura*. Observe que algunos árboles pueden ser clasificados de más de una forma o de ninguna.

a.

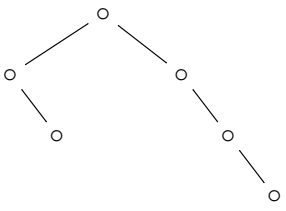


b.



c.

d.



2. Probar que todo árbol balanceado en cantidad está balanceado en altura. Ayuda: primero demostrar que todo árbol balanceado en cantidad de n elementos tiene altura $log\ n$.

3. Escribir una función `mirror` que dado un árbol binario, genere el árbol binario espejo, donde el hijo derecho de cada nodo pasa a ser izquierdo y el izquierdo pasa a ser derecho.

4. Escriba una función `btree_es_completo` que determine si el árbol es completo o no. *Ayuda:* piense cómo es la representación plana de un árbol binario completo con n nodos.

Árboles de Búsqueda Binaria (ABB)

5. Convertir un árbol binario de búsqueda en un árbol binario completo.

6. Implementar `bstree_cota_inferior(BSTree, int)` que dado un entero k, busque en el árbol binario de búsqueda el menor entero mayor a k.

7. Implementar `bstree_recorrer_intervalo(int cota_inf, int cota_sup, visitante)` que visite a los datos en el intervalo $[cota\_inf, cota\_sup]$.

8. Conceptualmente, un arbol binario de búsqueda y una lista doblemente circular están construidos a partir del mismo tipo de nodos, una estructura con dos referencias a otros nodos y un dato. Implementar `cdlist bstree_to_cdlist(bstree)` que dado un arbol de búsqueda binaria, reordene las referencias de los nodos de manera que se convierta en una lista doblemente circular (ordenada). Ayuda: crear un lista doblemente circular para el subárbol a izquierda, una para el subarbol a derecha, y otra con el nodo actual y luego unirlas en una única lista.

9. Implemente una función que, dado un arreglo con los datos del recorrido PRE ORDER de un árbol binario de búsqueda, reconstruya el árbol.

10. Implemente una función que, dada una secuencia de datos, determine si corresponde al recorrido BFS de un árbol binario de búsqueda *completo* y, en dicho caso, lo construya de manera eficiente. *Extra: considerar el caso en que no tenemos la información de que el árbol sea completo.*