## **Evaluación asincrónica 2** de Estrategias Algorítmicas, ITESO, Luis Gatica

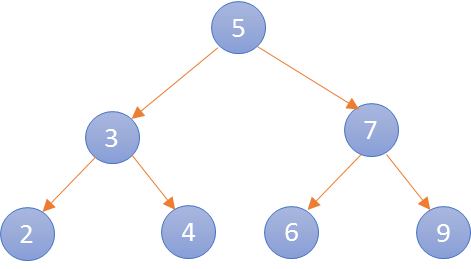
Nombre: [de 1 a 2 integrantes]

# Parte 2: árboles

Para los siguientes ejercicios, asuma que la clase correspondiente a cada árbol contiene el tamaño (número de llaves actualmente en el árbol) y una referencia al nodo raíz. Considere que para cada tipo de árbol además de la clase árbol es necesaria una clase nodo. Los valores que almacenarán los nodos son enteros.

1. **Árboles Binarios de Búsqueda (ABB)**

1) Crea una función(es) que determine(n) si existe una ruta de la raíz a alguna hoja tal que el número de claves impares de los nodos visitados es igual a *x*.

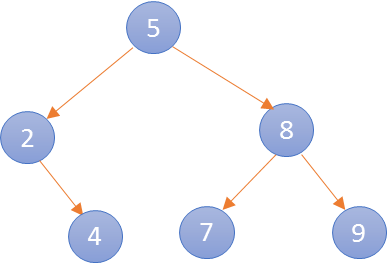
Así se usaría el método:

boolean exists1 = *existsPath*(root, 1); // exists1 = false

boolean exists2 = *existsPath*(root, 2); // exists2 = true

boolean exists3 = *existsPath*(root, 3); // exists3 = true

boolean exists4 = *existsPath*(root, 4); // exists4 = false

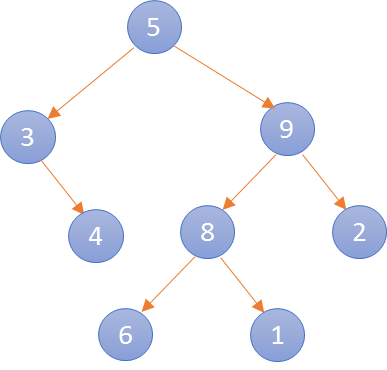
2) Crea una función(es) que determine(n) si un árbol binario es de búsqueda.

Así se usaría el método:

boolean isBST1 = *isBST*(root); // isBST1 = true

Si cambiamos, por ejemplo, a 7 por 3, ó a 4 por 1:

boolean isBST2 = *isBST*(root); // isBST2 = false

3) Crea una función(es) que construya(n) una lista de listas, tal que la lista *k* contiene las claves de todos los nodos que viven el nivel *k*, de izquierda a derecha.

Así se usaría el método:

List<List<Node>> nodeList = *getNodeListPerLevel*(root);

// nodeList = [ [5], [3, 9], [4, 8, 2], [6, 1] ]

1. **Árboles AVL**

Cada nodo contiene su dato o llave, referencia a sus hijos izquierdo y derecho, y un entero que representa la altura del nodo.

Escriba un método (puede escribir otros más y llamarlos) que reciba el árbol y determine si se cumplen todas las propiedades debidas:

1. A la izquierda de cada nodo sólo hay valores menores y a la derecha mayores. (O hijos nulos, en su defecto.)
2. El factor de balance de todos los nodos pertenece a {-1, 0, 1}
3. **Árboles rojinegros**

Cada nodo contiene su dato o llave, referencia a sus hijos izquierdo y derecho, y un booleano que representa si el nodo es rojo.

Escriba un método (puede escribir otros más y llamarlos) que reciba el árbol y determine si se cumplen todas las propiedades debidas:

1. A la izquierda de cada nodo sólo hay valores menores y a la derecha mayores. (O hijos nulos, en su defecto.)
2. Propiedad raíz (*root property* en las diapositivas).
3. Propiedad roja
4. Propiedad negra
5. **Árboles B**

La clase árbol además contiene un entero para el parámetro inmutable *t*. Cada nodo contiene los siguientes atributos: *parent*, *keys*, *children*, *n* (número de llaves actual), *isLeaf*.

Escriba un método (pueden escribir otros más y llamarlos) que reciba el árbol y determine si se cumplen las propiedades debidas:

1. Las llaves de todos los nodos siguen un orden no descendente (AKA: el orden "de siempre").
2. A la “izquierda” de cada llave sólo hay hijos con valores menores y a la “derecha” sólo hijos con valores mayores. (O hijos nulos, en su defecto.)
3. Todos los nodos (salvo, en su defecto, la raíz) contienen *t-1* llaves como mínimo.
4. Todos los nodos contienen *2t-1* llaves como máximo.
5. Todos los nodos internos tienen *n+1* hijos.
6. Todos los nodos internos tienen i*sLeaf* falso.
7. Todos los nodos hoja tienen *isLeaf* verdadero.
8. **Arboles Digitales de Búsqueda (*DST*)**

La clase árbol además contiene un entero que indica la longitud de bits con la que se está trabajando.

Cada nodo contiene su dato o llave, referencia a sus hijos izquierdo y derecho.

Escriba un método (pueden escribir otros más y llamarlos) que reciba el árbol y determine si se cumple que todos los nodos en el árbol coinciden con su ruta de inserción (considerando la longitud de bits indicada en el atributo respectivo de la clase árbol). Observe que la raíz puede ser cualquier valor. Por ejemplo, en el siguiente árbol (*casi* correcto), considerando una longitud de 4 bits las claves *5* y *4* sí son aptas para su ruta de inserción, pero *3* no.

7 [0111]

5 [0101]

3 [0011]

4 [0100]

1. **Árboles *Radix***

La clase árbol además contiene un entero que indica la longitud de bits con la que se está trabajando.

Cada nodo contiene su dato o llave, referencia a sus hijos izquierdo y derecho. Observe que el dato o llave puede ser nulo.

Escriba un método (pueden escribir otros más y llamarlos) que reciba el árbol y determine si se cumple que todos los nodos en el árbol coinciden con su ruta de inserción (considerando la longitud de bits indicada en el atributo respectivo de la clase árbol). Observe que la condición anterior implica que todos los nodos internos están vacíos (no tienen llave). Por ejemplo, en el siguiente árbol (*casi* correcto), considerando una longitud de 4 bits las llaves *4* y *11* sí son aptas para su ruta de inserción, pero *6* no.

4 [0100]

0

1

1

1

1

6 [0110]

0

0

0

1

1

11 [1011]

# Entregables y rúbrica

Código correcto en cada caso:

1. Árboles ABB
   * 1: 5%
   * 2: 5%
   * 3: 5%
2. Árboles AVL: 15%
3. Árboles rojinegros: 20%
4. Árboles B: 20%
5. Árboles digitales de búsqueda: 20%
6. Árboles radix: 10%

**Valor porcentual por cada parte:**

* Parte 1 (análisis divide y vencerás): 50%
* Parte 2 (árboles): 50%

Por ejemplo: si un estudiante obtiene todos los puntos de la primera parte, pero sólo la mitad de los puntos de la segunda, su puntaje final sería: 50% + 25% = 75% del total de la evaluación asincrónica.