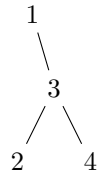

PRÁCTICA 3: ÁRBOLES BINARIOS, BÚSQUEDA Y BALANCEO

Ejercitación básica

Ejercicio 1. Diseñar algoritmos para resolver los siguientes problemas sobre árboles binarios. Implementar los algoritmos en Python.

- a) Mostrar un árbol en pantalla. El árbol vacío se muestra como (). Los hijos de un nodo se muestran con un nivel mayor de indentación que su padre. P. ej., el código de la derecha imprime en pantalla el árbol de la izquierda:

```
>>> mostrar(Nodo(None, 1, Nodo(Nodo(None, 2, None), 3, Nodo(None, 4, None))))
```



```
1
 \
  3
 / \
2   4
```

```
1
()
3
2
()
()
4
()
()
```

- b) Calcular una lista con todas las *ramas* del árbol. Recordemos que las ramas son todos los caminos desde la raíz hasta las hojas. P. ej., el siguiente código calcula las ramas del árbol de arriba:

```
>>> ramas(Nodo(None, 1, Nodo(Nodo(None, 2, None), 3, Nodo(None, 4, None))))
[[1, 3, 2], [1, 3, 4]]
```

- c) Calcular una lista con los *niveles* del árbol. El nivel 0 es la raíz, el nivel 1 está dado por los hijos de la raíz, el nivel 2 por los “nietos” de la raíz, etcétera. P. ej., el siguiente código calcula los niveles del árbol de arriba:

```
>>> niveles(Nodo(None, 1, Nodo(Nodo(None, 2, None), 3, Nodo(None, 4, None))))
[[1], [3], [2, 4]]
```

- d) Encontrar el subárbol de suma máxima.

Ejercicio 2. Dibujar todas las formas posibles que puede tener un árbol binario de 4 nodos. ¿Cuáles de ellos cumplen con el invariante de balanceo de un AVL?

Ejercicio 3. Diseñar un algoritmo que reciba como entrada un arreglo ordenado de n elementos y construya un AVL en tiempo $O(n)$ en peor caso. *Sugerencia:* usar D&C.

Ejercicio 4. Un *diccionario* es un tipo de datos que sirve para asociar *claves* a *significados*. Por ejemplo, la clave puede ser el nombre de un país y el significado el número de habitantes. Diseñar una estructura de datos basada en AVLs para proveer las siguientes operaciones y complejidades temporales en peor caso:

- Crear un diccionario vacío en $O(1)$.
- Insertar una clave en $O(\log n)$, asociándola a un significado. Si la clave ya se encontraba presente en el diccionario, su viejo significado se sobrescribe (es decir, el nuevo significado “pisa” al significado anterior).
- Buscar una clave en el diccionario, para determinar si está presente y recuperar su significado, en $O(\log n)$.
- Eliminar una clave en $O(\log n)$.
- Determinar el tamaño del diccionario (cantidad de claves distintas) en $O(1)$.
- Determinar la clave mínima y la clave máxima en $O(\log n)$.

Suponemos que las claves se pueden comparar en $O(1)$.

Ejercitación adicional

Ejercicio 5.

- a) Diseñar e implementar en Python una función `recorrido_en_orden` que reciba un árbol binario de búsqueda T de n nodos y construya un arreglo A de tamaño n que contenga a todos los elementos de A ordenados de menor a mayor. La complejidad temporal debe ser $O(n)$ en peor caso.
- b) Se propone el siguiente algoritmo de ordenamiento:
- Dado un arreglo A , insertar uno por uno los elementos en un AVL T .
 - Construir un arreglo B usando la función `recorrido_en_orden` del ítem “a”).
- ¿Cuál es la complejidad temporal en peor caso de este algoritmo?

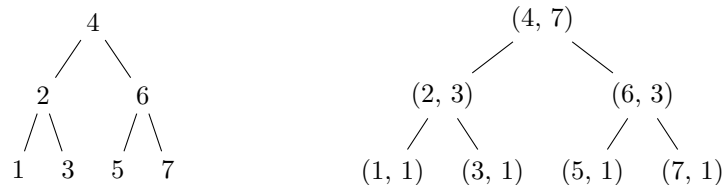
Ejercicio 6. Los algoritmos de búsqueda, inserción y eliminación de un ABB presentados en clase son recursivos. Diseñar e implementar en Python versiones iterativas de estos algoritmos, es decir, usando `while` y sin usar recursión.

Ejercicio 7. Diseñar un algoritmo que reciba un ABB T y un elemento x que aparece en T y determine el siguiente elemento que aparece en T , es decir, el menor y tal que y aparece en T , y tal que además vale $x < y$. La complejidad temporal debe ser $O(h)$ en peor caso, donde h es la altura de T .

Ejercicio 8. Diseñar una estructura de datos para hacer un muestreo de una secuencia de valores numéricos que implemente las siguientes operaciones:

- Inicializar el muestreo en $O(1)$.
- Registrar una muestra en $O(\log n)$, donde n es el total de muestras registradas hasta el momento.
- Determinar la *frecuencia* de una muestra en $O(\log n)$. La frecuencia (o “probabilidad”) de una muestra x se define como $\frac{\#(x)}{n}$ donde $\#(x)$ es el número de veces que se registro la muestra x y n es el total de muestras que se registraron.

Ejercicio 9. En este ejercicio, llamamos **AVL-con-tamaños** a una variante de los AVLs que en cada nodo contiene —además del valor correspondiente a ese nodo— un número que indica cuál es el tamaño de ese subárbol, es decir, el número de descendientes de dicho nodo. Por ejemplo, el AVL de la izquierda se puede modificar, agregando en cada nodo el tamaño del subárbol, para conseguir el AVL-con-tamaños de la derecha:



- a) Pensar cómo se podrían adaptar las operaciones de búsqueda, inserción y eliminación usuales de los AVLs para mantener el invariante de AVL-con-tamaños, manteniendo las complejidades temporales asintóticas en peor caso.
- b) Diseñar un algoritmo que reciba como entrada un AVL-con-tamaños T de n nodos y un elemento x y determine cuántos elementos menores que x hay en T en tiempo $O(\log n)$ en peor caso.