

# Pauta Auxiliar 5

September 2, 2019

Pregunta 1

Primero se definen los nodos que utilizaremos en la lista enlazada.

```
In [7]: class NodoLista:
        def __init__(self, v, s=None):
            self.val = v
            self.sig = s

        def __repr__(self):
            return repr(self.val)
```

Ahora definimos la lista con sus funciones:

```
In [8]: class ListaCircular:
        def __init__(self):
            self.primeros = None
            self.size = 0

        # Retorna la cantidad de elementos en la lista
        def largo(self):
            return self.size

        # Retorna el i-ésimo elemento de la lista, contando desde 1
        def obtener(self, i):
            if self.size == 0:
                return -1
            buscador = self.primeros
            for x in range(1, i):
                buscador = buscador.sig
            return buscador.val

        # Retorna el índice del elemento si existiese, de lo contrario retorna -1
        def existe(self, num):
            buscador = self.primeros
            for x in range(self.size):
                if int(buscador.val) == num:
                    return x
            buscador = buscador.sig
```

```

        return -1

# Elimina el i-ésimo elemento y retorna la referencia al siguiente nodo del eliminado
def eliminar(self, i):
    if self.size == 0:
        return None
    # Avanzar hasta quedar en la posición anterior al elemento a eliminar
    buscador = self.primeros
    if (i == 1): # Primer elemento es un caso especial
        # Recorremos hasta el último nodo para que quede en el nodo anterior al primer
        for x in range(1, self.size):
            buscador = buscador.sig
    else:
        # Caso general
        for x in range(1, i - 1):
            buscador = buscador.sig
    # Aquí estamos en el nodo anterior al que queremos eliminar, ahora lo eliminamos
    print("Eliminado: ", buscador.sig.val)
    buscador.sig = buscador.sig.sig # Cambio la referencia
    self.size -= 1
    return buscador.sig # Retornar el siguiente al eliminado

def agregar(self, v):
    if self.size == 0: # No elementos
        self.primeros = NodoLista(v)
        self.primeros.sig = self.primeros # Lista Circular
        self.size += 1
        return
    # Si existen elementos
    # Como lo inserto al final, apunta al primero
    nodo_nuevo = NodoLista(v, self.primeros)
    ultimo_antiguo = self.primeros
    for x in range(1, self.size):
        ultimo_antiguo = ultimo_antiguo.sig
    ultimo_antiguo.sig = nodo_nuevo # Ahora apunta al nuevo
    self.size += 1

```

Ahora probaremos lo que realizamos:

```

In [14]: a = ListaCircular()
         a.agregar(5)
         a.agregar(7)
         a.agregar(9)
         a.eliminar(3) # se pone el índice del elemento a eliminar, 3 representa al 9

```

Eliminado: 9

Out[14]: 5

## Pregunta 2

Primero, definiremos la función Josephus

```
In [17]: def josephus(clista, k):
        nodo = clista.primerono
        while clista.largo() > 1:
            # El siguiente al eliminado (recordar que cuenta desde 1)
            nodo = clista.eliminar(k+1)
            clista.primerono = nodo
        return nodo.val
```

Ahora definiremos una función con la que generaremos una lista qprobarla con el problema de Josephus

```
In [20]: def generar_lista(cant):
        lista = ListaCircular()
        for i in range(1, cant + 1):
            lista.agregar(i)
        return lista
```

Probaremos la función de Josephus con la función que creamos recién, creando nodos del 1 al 6 y usaremos valores de k desde 1 a 10.

```
In [22]: lista = ListaCircular()
        for k in range(1, 11): # De uno a diez
            lista = generar_lista(6)
            print("Último niño en la lista para k = ", k, ": ", josephus(lista, k))
```

```
Eliminado: 2
Eliminado: 4
Eliminado: 6
Eliminado: 3
Eliminado: 1
Último niño en la lista para k = 1 : 5
Eliminado: 3
Eliminado: 6
Eliminado: 4
Eliminado: 2
Eliminado: 5
Último niño en la lista para k = 2 : 1
Eliminado: 4
Eliminado: 2
Eliminado: 1
Eliminado: 3
Eliminado: 6
Último niño en la lista para k = 3 : 5
Eliminado: 5
Eliminado: 4
Eliminado: 6
```

```

Eliminado: 2
Eliminado: 3
Último niño en la lista para k = 4 : 1
Eliminado: 6
Eliminado: 1
Eliminado: 3
Eliminado: 2
Eliminado: 5
Último niño en la lista para k = 5 : 4
Eliminado: 1
Eliminado: 3
Eliminado: 6
Eliminado: 2
Eliminado: 4
Último niño en la lista para k = 6 : 5
Eliminado: 2
Eliminado: 5
Eliminado: 4
Eliminado: 1
Eliminado: 6
Último niño en la lista para k = 7 : 3
Eliminado: 3
Eliminado: 1
Eliminado: 2
Eliminado: 6
Eliminado: 4
Último niño en la lista para k = 8 : 5
Eliminado: 4
Eliminado: 3
Eliminado: 6
Eliminado: 1
Eliminado: 5
Último niño en la lista para k = 9 : 2
Eliminado: 5
Eliminado: 6
Eliminado: 3
Eliminado: 1
Eliminado: 2
Último niño en la lista para k = 10 : 4

```

### Pregunta 3

Definiremos los máximos y mínimos enteros que pueden representarse en el lenguaje para usarlos en la solución.

```
In [23]: import sys
```

```

MAX_VALUE = sys.maxsize
MIN_VALUE = -sys.maxsize - 1

```

Ahora definiremos la clase Arbol Binario, con sus métodos estáticos (para simplificar la implementación).

Un método estático es uno que no necesita de una instancia concreta de una clase para poder ser utilizado.

Para hacer un método estático en Python, utilizamos el tag @staticmethod y no usamos el self en la definición de los parámetros que recibe.

La gracia de hacer una recursión en un árbol binario es invocar al método a la izquierda y a la derecha.

```
In [25]: class ArbolBinario:
    def __init__(self, v, izq=None, der=None):
        self.val = v
        self.izq = izq
        self.der = der

    @staticmethod
    def numero_nodos(arbol):
        if arbol == None:
            return 0
        return 1 + ArbolBinario.numero_nodos(arbol.der) + ArbolBinario.numero_nodos(arbol.izq)

    @staticmethod
    def maximo(arbol):
        if arbol == None:
            return MIN_VALUE
        return max(arbol.val, max(ArbolBinario.maximo(arbol.izq), ArbolBinario.maximo(arbol.der)))

    @staticmethod
    def minimo(arbol):
        if arbol == None:
            return sys.maxsize
        return min(arbol.val, min(ArbolBinario.minimo(arbol.izq), ArbolBinario.minimo(arbol.der)))

    @staticmethod
    def suma(arbol):
        if arbol == None:
            return 0
        return arbol.val + ArbolBinario.suma(arbol.izq) + ArbolBinario.suma(arbol.der)

    @staticmethod
    def pisos(arbol):
        if arbol == None:
            return 0
        return 1 + max(ArbolBinario.pisos(arbol.izq), ArbolBinario.pisos(arbol.der))

    @staticmethod
    def es_arbol_binario(arbol, min=MIN_VALUE, max=MAX_VALUE):
        if arbol == None:
```

```

        return True
    if arbol.val < min or arbol.val > max:
        return False
    return ArbolBinario.es_arbol_binario(arbol.izq, min, arbol.val) and ArbolBinario.es_arbol_binario(arbol.dere, min, arbol.val)

```

Ahora definiremos dos árboles, a mano: uno binario y otro no binario, para probar nuestro código. Están identados para facilitar su lectura, por niveles.

```

In [26]: binario = ArbolBinario(5,
                                ArbolBinario(3,
                                                ArbolBinario(2),
                                                ArbolBinario(4)),
                                ArbolBinario(7,
                                                ArbolBinario(6),
                                                ArbolBinario(8,
                                                                None,
                                                                ArbolBinario(9))))
no_binario = ArbolBinario(5,
                           ArbolBinario(3,
                                           ArbolBinario(2),
                                           ArbolBinario(16)),
                           ArbolBinario(8,
                                           ArbolBinario(7),
                                           ArbolBinario(9)))

```

Además, probaremos nuestro código con un árbol binario Nulo (None). A continuación iremos probando si los árboles son binarios o no.

```

In [27]: print("Es binario el arbol binario? ", ArbolBinario.es_arbol_binario(binario))
        print("Es binario el arbol no binario? ",
              ArbolBinario.es_arbol_binario(no_binario))
        print("Es binario el arbol nulo? ", ArbolBinario.es_arbol_binario(None))

```

```

Es binario el arbol binario?  True
Es binario el arbol no binario?  False
Es binario el arbol nulo?  True

```

Ahora contaremos el número de nodos:

```

In [28]: print("Número de nodos Ejemplo binario: ", ArbolBinario.numero_nodos(binario))
        print("Número de nodos Ejemplo no binario: ",
              ArbolBinario.numero_nodos(no_binario))
        print("Número de nodos Ejemplo nulo: ", ArbolBinario.numero_nodos(None))

```

```

Número de nodos Ejemplo binario:  8
Número de nodos Ejemplo no binario:  7
Número de nodos Ejemplo nulo:  0

```

Contaremos cuáles son los máximos y mínimos nodos de los tres árboles

```
In [30]: print("Máximo valor del árbol binario: ", ArbolBinario.maximo(binario))
         print("Mínimo valor del árbol binario: ", ArbolBinario.minimo(binario))

         print("Máximo valor del árbol no binario: ", ArbolBinario.maximo(no_binario))
         print("Mínimo valor del árbol no binario: ", ArbolBinario.minimo(no_binario))

         print("Máximo valor del árbol no binario: ", ArbolBinario.maximo(None))
         print("Mínimo valor del árbol nulo: ", ArbolBinario.minimo(None))
```

```
Máximo valor del árbol binario:  9
Mínimo valor del árbol binario:  2
Máximo valor del árbol no binario:  16
Mínimo valor del árbol no binario:  2
Máximo valor del árbol no binario: -9223372036854775808
Mínimo valor del árbol nulo:  9223372036854775807
```

Finalmente, imprimimos el número de pisos de cada árbol:

```
In [31]: print("Número de pisos del árbol binario: ", ArbolBinario.pisos(binario))
         print("Número de pisos del árbol no binario: ", ArbolBinario.pisos(no_binario))
         print("Número de pisos del árbol Nulo: ", ArbolBinario.pisos(None))
```

```
Número de pisos del árbol binario:  4
Número de pisos del árbol no binario:  3
Número de pisos del árbol Nulo:  0
```

```
In [ ]:
```