Algoritmos y Estructura de Datos I

TAD ÁRBOL: Práctico de Aplicación

Link Repl.it:
https://replit.com/@Paulonia/Martinez13866Practica-Arboles#main.py

Ejercicio 1:

Implementar una función que verifique si un árbol binario está balanceado. Recordar que un árbol balanceado es aquel en el que la altura del subárbol izquierdo y la del subárbol derecho difiera en como máximo 1.

main.py

```python
from binarytree import *
import mylinkedlist

# -- Ejercicio 1 --
def checkBalancedTree(B):
  #verifica si un árbol binario está balanceado (Retorna True si lo es, False si no)
  if B.root==None:
    return None
  else:
    check=True
    levelSub1=0;levelSub2=0
    if B.root.leftnode!=None:
      levelSub1=1
      levelSub1=checkBalance(B.root.leftnode,levelSub1)
    if B.root.rightnode!=None:
      levelSub2=1
      levelSub2=checkBalance(B.root.rightnode,levelSub2)
    if levelSub1>levelSub2:
      if (levelSub1-levelSub2)>1:
        check=False
    else:
      if (levelSub2-levelSub1)>1:
        check=False
    return check

def checkBalance(current,level):
  level=checkLevelperHead(current,level)
  if current.leftnode!=None:
    level=checkBalance(current.leftnode,level)
  if current.rightnode!=None:
    level=checkBalance(current.rightnode,level)
  return level

def checkLevelperHead(current,level):
  #Verifica si un nodo tiene hijos a izq y derecha y suma los que tenga (solo los que sean sus hijos, no va a sumar los hijos de sus hijos)
  if current.leftnode!=None:
    level+=1
  if current.rightnode!=None:
    level+=1
  return level

```

Para el **ejercicio 1** voy verificando la cantidad de hijos de cada nodo y los voy comparando por cada subárbol (izquierdo y derecho). Si la diferencia es mayor a 1, no está balanceado.

Ejercicio 2:

Sean BT1 y BT2 dos árboles binarios, donde BT1 es mayor (mayor cantidad de nodos) que BT2. Implementar un algoritmo que determine si BT2 es un subárbol de BT1.

```
42  # -- Ejercicio 2 --
43  def checkSubTree(B1,B2):
44    #determina si B2 es un subárbol de B1 (Retorna True si lo es, False si no, y None
    si alguno de los árboles ingresados está vacío)
45    #Primero verifico si el tamaño del árbol 1 es mayor al tamaño del árbol 2, luego
    si se ingresa algún arbol vacío
46    L1=traverseBreadFirst(B1)
47    L2=traverseBreadFirst(B2)
48    if mylinkedlist.length(L1)<mylinkedlist.length(L2):
49      print("Tamaños no válidos")
50      return None
51    elif B1.root==None or B2.root==None:
52      return None
53    else:
54      check=False
55      check=checkSubHead(B1.root,B2.root,check)
56      return check
57
58  def checkSubHead(currentB1,currentB2,check):
59    if currentB1!=None:
60      #Encuentro donde empieza el subarbol B2, si su raiz no se encuentra en B1
    delvuelve False
61      if currentB1.key==currentB2.key:
62        check=True
63        check=checkSubTreeComplete(currentB1,currentB2,check)
64        return check
65      else:
66        if currentB1.leftnode!=None:
67          check=checkSubHead(currentB1.leftnode,currentB2,check)
68        if currentB1.rightnode!=None:
69          check=checkSubHead(currentB1.rightnode,currentB2,check)
70    return check
```

```python
def checkSubTreeComplete(currentB1,currentB2,check):
    #Verifico si los nodos son distintos
    if currentB1!=None and currentB2!=None:
        if currentB1.key!=currentB2.key:
            check=False
    #Checkeo si el segundo arbol tiene nodos de más por izquierda o derecha
    if currentB1.leftnode==None and currentB2.leftnode!=None:
        check=False
    elif currentB1.rightnode==None and currentB2.rightnode!=None:
        check=False
    #Checkeo si los dos arboles tienen nodos a la izquierda
    if currentB1.leftnode!=None and currentB2.leftnode!=None:
        check=checkSubTreeComplete(currentB1.leftnode,currentB2.leftnode,check)
    #Checkeo si los dos arboles tienen nodos a la derecha
    if currentB1.rightnode!=None and currentB2.rightnode!=None:
        check=checkSubTreeComplete(currentB1.rightnode,currentB2.rightnode,check)
    return check
```

Para el **ejercicio 2** tuve en cuenta que primero hay que verificar la cabeza del subárbol dentro del árbol principal, y de ahí puedo empezar a verificar sus respectivos hijos (si son iguales entre si o no). Si el subárbol tiene hijos izquierdos o derechos de más, es Falso.

Ejercicio 3:

Escribir una función **checkBST(B)** que verifique que un árbol binario es un Árbol Binario de Búsqueda.
Es decir que se cumple la propiedad : leftnode < currentnode < rightnode.

```python
# -- Ejercicio 3 --
def checkBST(B):
    #verifica que un árbol binario es un Árbol Binario de Búsqueda (Retorna True si lo es, False si no)
    if B.root==None:
        return None
    checkBinary=True
    checkBinary=checkBTree(B.root,checkBinary)
    return checkBinary
def checkBTree(current,checkBinary):
    if current!=None:
        if current.leftnode!=None:
            if current.leftnode.key>=current.key:
                checkBinary=False
        if current.rightnode!=None:
            if current.rightnode.key<=current.key:
                checkBinary=False
        checkBinary=checkBTree(current.leftnode,checkBinary)
        checkBinary=checkBTree(current.rightnode,checkBinary)
    return checkBinary
```

Para el **ejercicio 3** pensé en verificar que todos los nodos izquierdos fueran menores al padre y todos los nodos derechos fueran mayores al padre.