Estructura de Datos y Algoritmos **Grado Ingeniería Informática** **Doble Grado Ingeniería Informática y Administración de Empresas** **Universidad Carlos III de Madrid** **CURSO 2020-2021**	uc3m | Universidad Carlos III de Madrid

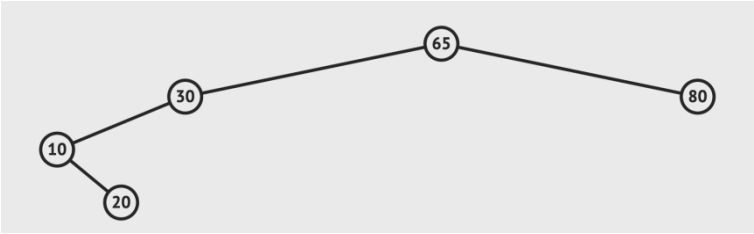
Convocatoria Extraordinaria. 23 Junio 2021.

Problema 1 (3 puntos): En la clase BST (subclase de BinarySearchTree), implementa la función ***countPairs***, que toma como argumentos otro objeto instancia de BST, otherTree, y un valor entero, k, y devuelve el número de parejas de nodos que cumplan que la suma de sus claves sea igual a k. Cada pareja de nodos debe estar formada necesariamente por un nodo de cada árbol.

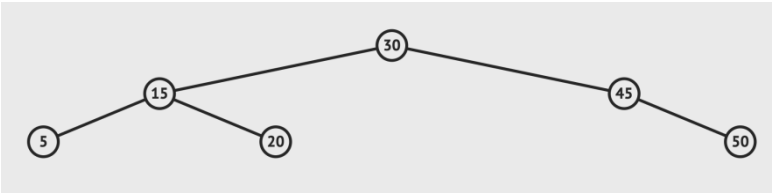
Ejemplo:

Dados las instancias tree1 y tree2 de la clase BST:

tree1



tree2



El resultado de la operación tree1.countPairs(tree2,50) será 2

(hay dos parejas de nodos cuyas claves cuya suma es 50: [20,30] y [30,20])

El resultado de la operación tree1.countPairs(tree2,45) será 1
(hay una pareja de nodos cuyas claves suman 45: [30,15]

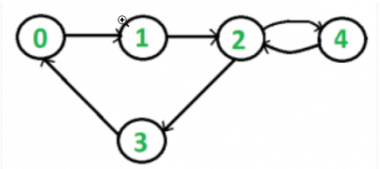
El resultado de la operación tree1.countPairs(tree2,87) será 0
(no hay ninguna pareja de claves que sumen 87)

Implementa a continuación tu solución (utiliza las cuadrículas para indicar la correcta indentación de cada bloque). Recuerda que tu solución debe ser lo más eficiente posible en términos de complejidad temporal.

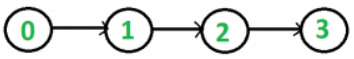
```
from binarysearchtree import BinarySearchTree
class BST (BinarySearchTree):
    def countPairs (self,otherTree,k):
        ...
```

Problema 2 (3 puntos): La clase MyGraph es una implementación basada en diccionarios para representar un grafo dirigido. Para simplificar el problema, supondremos que los vértices son números enteros no negativos (0,1,2,..). Implementa la función ***isStronglyConn,*** que devuelva True si existe siempre un trayecto entre cualesquiera dos vértices del grafo y Falso en caso contrario. Está permitido utilizar estructuras de Python como las listas o los diccionarios.
Ejemplo:

graph1:



graph2:



La operación mg1.isStronglyConn() devolverá True
La operación mg2.isStronglyConn() devolverá False

Implementa a continuación tu solución (utiliza las cuadrículas para indicar la correcta indentación de cada bloque).

```
class MyGraph:
    def __init__(self,n):
```

```
        """Crea un grafo con n vértices (0,1,...,n-1)"""
            self.vertices={}
            for i in range(n):
                self.vertices[i]=[]
        def addConnection(self,i,j):
            if i not in self.vertices.keys():
                return
            if j not in self.vertices.keys():
                return
            self.vertices[i].append(j)
        def isStronglyConn(self):
```

Problema 3 (4 puntos): Implementa el algoritmo **quicksort** para ordenar una lista doblemente enlazada. Este algoritmo debe ser un método en la clase DList2, subclase de DList. Dicho método ordena la lista representada por el objeto invocante (self). El método no devuelve nada. Puedes añadir funciones auxiliares que te ayuden a resolver tu problema. Las funciones de DList no deben implementarse, es decir, puedes invocarlas directamente. No está permitido usar estructuras de Python (listas de python, diccionarios, etc) para resolver el problema.

```
class DList2(DList):
    def quicksort(self):
```