import queue as queue

class NarioTreeNodeClass:

    def \_\_init\_\_(self, orden, elemento = None):

        ''' Class N-ary Search Tree Node Class which is a list

            representation of ADT Structure Node where it's possible

            to store N-1 elements.

            Args:

                orden:  The value will define the number of elements

                        and children the tree will manage.

                elemento: First Node's data value

            Attributes:

            Hijos[](list): A list where the child nodes will be stored.

            Elementos[](list): A list where the node's elements

                               will be stored

            Orden(int):  The value will define the number of elements

                         and children the tree will manage.

        '''

        self.\_\_Hijos = []

        self.\_\_Elementos = []

        self.\_\_orden = orden

        for i in range(0, self.\_\_orden):

            self.\_\_Hijos.append(None)

            if(i < (self.\_\_orden-1)):

                self.\_\_Elementos.append(None)

        self.\_\_Elementos[0] = elemento

    def getElemento(self, posicion):

        ''' Getter method for getting a specific element in the list by

            passing the position as argument

            Args:

                posicion(int): The position of the element

                               to be returned

            Return:

                The element in the specified position

        '''

        return self.\_\_Elementos[posicion]

    def setElemento(self, posicion, dato):

        ''' Setter method for setting an element in a specific position

            Args:

                posicion(int): The position of the element

                               to be set

                dato(object): The element its being set

        '''

        self.\_\_Elementos[posicion] = dato

    def getHijo(self, posicion):

        ''' Getter method for getting a specific child list by

            passing the position as argument

            Args:

                posicion(int): The position of the child list

                               to be returned

            Return:

                The child list in the specified position

        '''

        return self.\_\_Hijos[posicion]

    def setHijo(self, posicion, hijo):

        ''' Setter method for setting a child list in a specific

            position

            Args:

                posicion(int): The position of the child list

                               to be set

                hijo(list): The child list it's being set

        '''

        self.\_\_Hijos[posicion] = hijo

    def getNroElementosVacios(self):

        ''' Check how many empty slots there are in the elements list

            Return:

                cantidad(int): the quantity of empty slots there

                               are in the elements list

        '''

        cantidad = i = 0

        while(i < self.\_\_orden):

            if(self.\_\_Elementos[i] == None):

                cantidad += 1

        return cantidad

    def getNroElementosOcupados(self):

        ''' Check how many not empty slots there are in the elements list

            Return:

                cantidad(int): the quantity of not empty slots there

                               are in the elements list

        '''

        cantidad = i = 0

        while(i < self.\_\_orden -1 ):

            if(self.\_\_Elementos[i] != None):

                cantidad += 1

            i += 1

        return cantidad

    def getNroHijosVacios(self):

        ''' Check how many empty slots there are in the child list

            Return:

            cantidad(int): The quantity of empty slots there

                           are int the child list

        '''

        cantidad = i = 0

        while(i <= self.\_\_Hijos.\_\_len\_\_()):

            if(self.\_\_Hijos[i] == None):

                cantidad += 1

        return cantidad

    def getNroHijosOcupados(self):

        ''' Check how many not empty slots there are in the child list

            Return:

                cantidad(int): the quantity of not empty slots there

                               are in the child list

        '''

        cantidad = i = 0

        while(i < self.\_\_Hijos.\_\_len\_\_()):

            if(self.\_\_Hijos[i] != None):

                cantidad += 1

            i += 1

        return cantidad

    def EsElementoVacio(self, posicion):

        ''' Verify a specific slot if whether is empty or not

            in the element list

            Args:

                position(int): position to be checked.

            Return:

                True: the position is empty

                False: The position os not empty

        '''

        if (self.\_\_Elementos[posicion] == None):

            return True

        else:

            return False

    def esHijoVacio(self, posicion):

        ''' Verify a specific slot if whether is empty or not

            in the child list

            Args:

                position(int): position to be checked.

            Return:

                True: the position is empty

                False: The position is not empty

        '''

        return self.\_\_Hijos[posicion] == None

    def getUltimoElementoOcupado(self):

        ''' Getter method for getting the index of the last slot not

            empty in the element list

            Return:

                The position of the last slot not empty

                in the element list

        '''

        i = 0

        while(i < self.\_\_orden - 1):

            if( self.EsElementoVacio(i) ):

                return i #self.\_\_Elementos[i]

            i += 1

        return 0

    def EstaNodoLleno(self):

        ''' Verify if a node has all the slots not empty in its list

            Return:

                True: the entire node it's not empty

        '''

        if (self.getNroElementosOcupados() == self.\_\_orden-1):

            return True

        else:

            return False

    def insertarElemento(self, elemento):

        ''' Method to insert an element in its correct position within

            the list

            Args:

                elemento(object): the value its being attempted

                                  to insert

        '''

        posicion = self.getUltimoElementoOcupado()

        while (posicion > 0):

            if (elemento < self.\_\_Elementos[posicion-1]):

                self.\_\_Elementos[posicion] = self.\_\_Elementos[posicion -1 ]

            else:

                self.\_\_Elementos[posicion] = elemento

                return

            posicion -= 1

        self.\_\_Elementos[0] = elemento

from NarioTreeNodeClass import NarioTreeNodeClass

class NarioTreeClass:

    ''' Class N-ary Search Tree Class which is the representation of ADT

        structure where nodes can have no more than N nodes

        and N-1 elements.

        Args:

            Orden:  The value will define the number of elements

                    and children the tree will manage.

            elemento:   First Node's data value ###put it in the

                        imported class.

        Attributes:

            Raiz:   It creates the link to the first structure node

                    in the N-ary Search Tree. It defaults to None.

            Orden:  The value will define the number of elements

                    and children the tree will manage.

    '''

    def \_\_init\_\_(self, orden = None):

        ''' Inits a instance of N-ary Search Tree with default values

            in their attirbutes

        '''

        self.\_\_raiz = None

        self.\_\_orden = orden

    def getOrden(self):

        ''' Getter method for returning the root node of AVLTree '''

        return self.\_\_orden

    def setOrden(self, orden):

        ''' Setter method for setting the  node of AVLTree

            Args:

                link(Node)

        '''

        self.\_\_orden = orden

    def getRaiz(self):

        ''' Getter method for returning the N-ary Search Tree's

            root node

            Return:

                The reference to the root node

        '''

        return self.\_\_raiz

    def setRaiz(self, raiz):

        ''' Setter method for setting the N-ary Search Tree's root

            node of AVLTree

            Args:

                raiz(Node)

        '''

        self.\_\_raiz = raiz

    def esNodoVacio(self, current):

        ''' Check if whether the node is empty or not

            Return:

                True: The node is empty

                False: The node has at least on element

        '''

        return current == None

    def esArbolVacio(self):

        ''' Check the whole tree if whether is empty or not

            Return:

                True: The tree is empty

                False: The node has at least on element

        '''

        return self.esNodoVacio(self.\_\_raiz)

    def esNodoHoja(self, current):

        ''' Check a node if whether is a leaf or not

            Args:

                current: the current node to check

            Return:

                True: The node is a leaf

                False:  The node has at least one child

        '''

        for i in range(0, self.\_\_orden -1):

            if( current.esHijoVacio(i) == True ):

                return True

        return False

    def existeDatoEnNodo(self, current, elemento):

        ''' Verify a element if it's already within a node or not

            Args:

                current(Node): the current node where the element has

                               to be verified

                elemento(object): the element to be verified

            Return:

                True: The element already exists within current node

                False: The element doesn't exists within current node

        '''

        i = 0

        while( (i < self.\_\_orden-1) and (current.EsElementoVacio(i) == False) ):

            if( elemento == current.getElemento(i)):

                return True

            i += 1

        return False

    def enQueHijoVa(self, current, elemento):

        ''' Method to determine in which position a child node

            will be created.

            Args:

                current(Node): The current node it's being analyzed

                element(object): The element is being attempted to

                                 insert

            Return:

                i: the position where a child node has to be created

                -1: the element already exists in the node

        '''

        cantidadNodosOcupados = current.getNroElementosOcupados()

        i = 0

        while( i < cantidadNodosOcupados ):

            if( elemento == current.getElemento(i)):

                return -1

            if( elemento < current.getElemento(i)):

                return i

            i += 1

        return i

    def insertar(self, elemento):

        ''' Method to insert an element in its correct position within

            the N-ary Search Tree

            Args:

                elemento(object): the value its being attempted

                                  to insert

        '''

        if( self.esArbolVacio()):

            self.setRaiz(NarioTreeNodeClass(self.\_\_orden, elemento))

            return

        current = self.getRaiz() #Makes a copy of the original root

        while( self.esNodoVacio(current) == False):

            if(self.esNodoHoja(current)):

                if(current.EstaNodoLleno() ):

                    posicionHijo = self.enQueHijoVa(current, elemento)

                    if (posicionHijo != -1 ):

                        nuevoNodo = NarioTreeNodeClass(self.\_\_orden, elemento)

                        current.setHijo(posicionHijo, nuevoNodo)

                        return

                    else:

                        return

                elif (self.existeDatoEnNodo(current, elemento) == False):

                    current.insertarElemento(elemento)

                else:

                    return

            else:

                posicionHijo = self.enQueHijoVa(current, elemento)

                if(posicionHijo == -1):

                    return

                if( current.esHijoVacio(posicionHijo)):

                    nuevoNodo = NarioTreeNodeClass(self.\_\_orden, elemento)

                    current.setHijo(posicionHijo, nuevoNodo)

                    return

                current = current.getHijo(posicionHijo)

def main():

    print("Test of the N-ary Tree")

    Kario = NarioTreeClass(3)

    Kario.insertar(100)

    Kario.insertar(80)

    Kario.insertar(135)

    Kario.insertar(70)

    Kario.insertar(90)

    Kario.insertar(91)

    Kario.insertar(92)

    print(str(0))

if \_\_name\_\_ == "\_\_main\_\_":

    main()