ESTRUCTURAS DE DATOS: MÉTODOS

Ejercicios Extraordinario: 18 de junio del 2024

TEMA 3: CADENAS DE TEXTO

Primer Parcial: Cadenas

```python
# Cadenas: Cuantos Dígitos - - - - - - - - - - - - - - - - - - - - - - - - - - -
cadena = input()
limpia = cadena.lstrip('0')
print(len(limpia))

# Cadenas: Capitalizar nombre propio - - - - - - - - - - - - - - - - - - - - - -
def capitalizar(cadena):
    excepciones = ['de', 'la']
    partes = cadena.split(' ')

    resultado = []
    for palabra in partes:
        if palabra in excepciones:
            resultado.append(palabra)
        else:
            resultado.append(palabra.capitalize())

    nombre_capitalizado = ' '.join(resultado)
    return nombre_capitalizado

if __name__ == '__main__':
    cadena = input()
    capitalizar_nombre = capitalizar(cadena)
    print(capitalizar_nombre)
```

Ordinario: Cadenas

```python
# Cadenas: Mitad Cadena - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cadena = input()
longitud = len(cadena)

mitad = longitud // 2
mitad2 = mitad -1

if longitud % 2 == 0: # corrección de cambiar mitad por longitud
    impar = cadena[mitad2] + cadena[mitad]
    print(impar)
else:
    print(cadena[mitad])
```

TEMA 5: LISTAS

Primer Parcial: Listas

```python
# Listas: Intercalar listas 1 - - - - - - - - - - - - - - - - - - - - - - - - -
def intercalar(lista1, lista2):
    lista_intercalada = []

    longitud_lista1 = len(lista1)
    longitud_lista2 = len(lista2)
```

```
    longitud_listas = max(longitud_lista1, longitud_lista2)

    for i in range(longitud_listas):
        if i < longitud_lista1:
            lista_intercalada.append(lista1[i])
        if i < longitud_lista2:
            lista_intercalada.append(lista2[i])
    return lista_intercalada

if __name__ == '__main__':
    n = int(input())
    m = int(input())
    lista1 = [int(input()) for _ in range(n)]
    lista2 = [int(input()) for _ in range(m)]

    lista_intercalada = intercalar(lista1, lista2)
    print(lista_intercalada)
```

Segundo Parcial: Listas

```
# Listas: Juntar Ordenadas - - - - - - - - - - - - - - - - - - - - - - - - - -
def ordenar_lista(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
    return lista
if __name__ == '__main__':
    n = int(input())
    m = int(input())
    lista1 = [int(input()) for _ in range(n)]
    lista2 = [int(input()) for _ in range(m)]
    lista = lista1 + lista2
    resultado = ordenar_lista(lista)
    print(resultado)
```

En clase: Listas

```
# Listas: Intersección - - - - - - - - - - - - - - - - - - - - - - - - - - - -
def interseccion(lista1, lista2):
    resultado = []
    if len(lista1) > len(lista2):
        mayor = lista1
        menor = lista2
    else:
        mayor = lista2
        menor = lista1
    for elemento1 in mayor:
        for elemento2 in menor:
            if elemento1 == elemento2 and not elemento2 in resultado:
                resultado.append(elemento2)
    resultado.sort()
    return resultado

if __name__ == '__main__':
    n = int(input())
    m = int(input())
    lista1 = [int(input()) for _ in range (n)]
    lista2 = [int(input()) for _ in range (m)]
```

```
    resultado = interseccion(lista1, lista2)
    print(resultado)

# Listas: Lista repetida - - - - - - - - - - - - - - - - - - - - - - - - - - - -
def elementos_se_repiten(n, lista):
    for i in range(n):
        for j in range(i + 1, n):
            if lista[i] == lista[j]:
                return True
    return False
if __name__ == '__main__':
    n = int(input())
    lista = []
    for _ in range(n):
        numero = int(input())
        lista.append(numero)
    print(elementos_se_repiten(n, lista))
```

TEMA 5: MATRICES

Primer Parcial: Matrices

```
# Matriz: Centro de la matriz - - - - - - - - - - - - - - - - - - - - - - - - -
def crear_matriz(filas, columnas):
    matriz = []
    for _ in range(filas):
        fila = []
        for _ in range(columnas):
            valor = int(input())
            fila.append(valor)
        matriz.append(fila)
    return matriz

if __name__ == '__main__':
    filas = int(input())
    matriz = crear_matriz(filas, filas)

    # Mitad de la matriz
    mitad = int(filas/2)
    print(matriz[mitad][mitad])
```

En clase: Matrices

```
# Matriz: Suma Diagonal 01 - - - - - - - - - - - - - - - - - - - - - - - - - - -
def suma_diagonal(matriz):
    filas = len(matriz)
    columnas = filas
    res = 0
    for posicion in range(columnas):
        res = res + matriz[posicion][posicion]
    return res

def leer_matriz(filas, columnas):
    matriz = []
    for f in range(filas):
        ff = []
        for c in range(columnas):
            ff.append(int(input()))
        matriz.append(ff)
```

```
    return matriz

if __name__ == '__main__':
    filas = int(input())
    columnas = int(input())
    matriz = leer_matriz(filas, columnas) #filas/columnas --> matriz cuadrada o no
    print(suma_diagonal(matriz))

# Matriz: Suma Diagonal 02 - - - - - - - - - - - - - - - - - - - - - - - - - - - -
def leer_matriz(filas, columnas):
    matriz = []
    for f in range(filas):
        ff = []
        for c in range(columnas):
            ff.append(int(input()))
        matriz.append(ff)
    return matriz

def suma_diagonal(matriz):
    filas = len(matriz)
    columnas = filas
    res = 0
    for posicion in range(columnas):
        res = res + matriz[posicion][posicion]
    return res

if __name__ == '__main__':
    filas = int(input()) # entrada: filas * filas --> matriz cuadrada
    matriz = leer_matriz(filas, filas)
    print(matriz)
    print(suma_diagonal(matriz))

# Matriz: Suma Columna - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
def leer_matriz(filas, columnas):
    matriz = []
    for f in range(filas):
        ff = []
        for c in range(columnas):
            ff.append(int(input()))
        matriz.append(ff)
    return matriz

def sumar_columnas(matriz):
    filas = len(matriz)
    columnas = len(matriz[0])
    res = [0 for _ in range(columnas)]
    for f in range(filas):
        for c in range(columnas):
           res[c] += matriz[f][c]
    return res

def dar_formato(lista):
    lista_cadenas = [str(ele) for ele in lista]
    return ','.join(lista_cadenas)

if __name__ == '__main__':
    filas = int(input())
    columnas = int(input())
    matriz = leer_matriz(filas, columnas)
    print(dar_formato(sumar_columnas(matriz)))
```

TEMA 10: RECURSIVIDAD

```python
# Matriz: Suma Diagonal 02 - - - - - - - - - - - - - - - - - - - - - - - - - - - -
def suma_impares_recursiva(n):
    if n <= 0:
        return 0
    elif n % 2 == 1:  # Si n es impar
        return n + suma_impares_recursiva(n - 2)
    else:  # Si n es par, buscar el siguiente impar
        return suma_impares_recursiva(n - 1)
if __name__ == '__main__':
    n = int(input().strip())
    resultado = suma_impares_recursiva(n)
    print(resultado)
```

TEMA 12: ÁRBOLES

Árboles Binarios: Árboles

```python
# Árboles: Índice menor - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
class Nodo():
    def __init__(self, indice, valor=None):

    def __repr__(self) -> str:

class Arbol_binario():

    def __init__(self):

    def agregar_nodo(self, indice, valor=None):

    def regresar_valor(self, indice: int) -> str:

    def imprimir_arbol_rec(self, nodo, nivel, res):

    def __repr__(self) -> str:

    def borrar_nodo(self, indice:int) -> None:

def leer_arbol(n: int) -> Arbol_binario:

   # - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
def indice_menor_rec(nodo):
    if nodo is None:
        return None
    if nodo.izquierda is None:
        return nodo.indice
    return indice_menor_rec(nodo.izquierda)

def indice_menor(arbol):
    if arbol.raiz is None:
        return None
    return indice_menor_rec(arbol.raiz)
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
if __name__ == '__main__':
    n = int(input())
    arbol = leer_arbol(n)
    print(indice_menor(arbol))
```

```python
# Árboles: Segundo índice menor - - - - - - - - - - - - - - - - - - - - - - - - - -
class Nodo():

    def __init__(self, indice, valor=None):

    def __repr__(self) -> str:

class Arbol_binario():

    def __init__(self):

    def agregar_nodo(self, indice, valor=None):

    def regresar_valor(self, indice: int) -> str:

    def imprimir_arbol_rec(self, nodo, nivel, res):

    def __repr__(self) -> str:

    def borrar_nodo(self, indice:int) -> None:

def leer_arbol(n: int) -> Arbol_binario:

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
def indice_menor_rec(nodo, menor, segundo_menor):
    if nodo is None:
        return segundo_menor  # Retorna el segundo menor

    # Si nodo.izquierda es None, nodo es el menor hasta ahora
    if nodo.izquierda is None:
        if nodo.indice < menor:
            segundo_menor = menor
            menor = nodo.indice
        elif nodo.indice > menor and nodo.indice < segundo_menor:
            segundo_menor = nodo.indice
        return segundo_menor

    # Si nodo.izquierda no es None, seguimos buscando por la izquierda
    if nodo.indice < menor:
        segundo_menor = menor
        menor = nodo.indice
    elif nodo.indice > menor and nodo.indice < segundo_menor:
        segundo_menor = nodo.indice
    return indice_menor_rec(nodo.izquierda, menor, segundo_menor)

def indice_menor(arbol):
    if arbol.raiz is None:
        return None
    menor = arbol.raiz.indice
    segundo_menor = None
    segundo_menor = indice_menor_rec(arbol.raiz, menor, segundo_menor)
    return segundo_menor
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
if __name__ == '__main__':
    n = int(input())
    arbol = leer_arbol(n)
    print(indice_menor(arbol))
```

Árboles Generales: Generales

```python
# Árboles: Ancestro árbol - - - - - - - - - - - - - - - - - - - - - - - - - - - -
class Nodo():

    def __init__(self, indice, padre=None):

    def agregar_hijo(self, hijo):

    def __repr__(self):

class Arbol():

    def __init__(self):

    def regresarNodo_rec(self, indice, actual, resultado):

    def regresarNodo(self, indice):

    def agregar_hijo(self, indice, indicePadre=None):

    def es_hoja(self, indice: int) -> bool:

    def regresar_padre(self, indice: int) -> int:

    def convertir_a_cadena_arbol_rec(self, nodo: Nodo, nivel, res) -> None:

    def __repr__(self) -> str:

    def borrar_nodo(self, indice: int) -> None:

def leer_arbol(nodos: int) -> Arbol:

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
def es_ancestro(arbol: Arbol, indiceA: int, indiceB: int) -> bool:
    """
    Dado un objeto Arbol,
    Determina si el nodo en indiceA es ancestro
    de el nodo en el indiceB.
    arbol: Arbol, indiceA: int, indiceB: int
    returns: bool, True si A es ancestro de B
    """
    nodoB = arbol.regresarNodo(indiceB)
    if not nodoB:
        return False
    siguiente = nodoB.padre
    while siguiente != None:
        if siguiente.indice == indiceA:
            return True
        siguiente = siguiente.padre
    return False
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
if __name__ == '__main__':
    indiceA = int(input())
    indiceB = int(input())
    n = int(input())
    arbol = leer_arbol(n)
    print(es_ancestro(arbol, indiceA, indiceB))
```