Universidad Europea Madrid

Actividad II

Técnicas de programación avanzadas

Cristina Díez Sobrino, 2º Ingeniería Informática

Índice

Inciso

****La contraseña del login es mi número de expediente: 21732599***

Apartado A

```
def AobtenerMediana (lista):
    if(len(lista)%2!=0):
        pos =int((len(lista)/2)+1)
        print("El numero de elementos es impar",
        ", la posicion de la mediana es ",pos,"y su valor es ",lista[pos-1])
    else:
        pos =int((len(lista)/2))
        dato = lista[pos-1]+lista[pos]
        med = dato/2
        print("El numero de elementos es par y la mediana es: ", med)


def AjuntarYOrdenar(lista1,lista2):
    #meto los elementos de la lista2 en la lista1
    for i in range (0,len(lista2)):
        lista1.append(lista2[i])
    print("Lista sin ordenar: ",lista1)
    lista1 = Aquicksort(lista1)
    print("Lista ordenada: ",lista1)
    return lista1

def Aquicksort(lista):
    #tres listas que marcaran los valores menores,mayores e iguales al pivote
    izquierda = []
    centro = []
    derecha = []
    if len(lista) > 1:
        #el primer elemento de la lista es el pivote
        pivote = lista[0]
        #se va recorriendo el array y se comparan los valores con el pivote
        for i in lista:
            #si el valor es menor que el pivote se pone a la izquierda
            if i < pivote:
                izquierda.append(i)
            #si el valor es igual que el pivote se pone en el centro
            elif i == pivote:
                centro.append(i)
            #si el valor es mayor que el pivote se pone a la derecha
            elif i > pivote:
                derecha.append(i)
        #se llama recursivamente con los arrays de derecha e izquierda
        return Aquicksort(izquierda)+centro+Aquicksort(derecha)
    else:
        return lista


#ejecucion
        lista1 = [1,2,3,7,8]
        lista2 = [0,4,6,9]
        listaV = AjuntarYOrdenar(lista1, lista2)
        AobtenerMediana(listaV)
```

El ejercicio A consta de 3 métodos:

AjuntarYOrdenar junta las dos listas en lista1 mediante un for [O(n)] y llama al método *Aquicksort* que es el algoritmo de ordenamiento QuickSort[O(n^2)] (además este algoritmo usa la metodología Divide y Vencerás) que ordena el array.

Por último, se le pasa el array ordenado al método *AobtenerMediana* que obtiene la mediana dependiendo de dos casos matemáticos:

Si el número de elementos en el array es **impar**:

Hay **quince** números. El del medio es el **octavo** número:

3, 5, 7, 12, 13, 14, 21, **23**, 23, 23, 23, 29, 39, 40, 56

La mediana de este conjunto de valores es **23**.

Ejemplo 1*

Si el número de elementos en el array es **par**:

Ahora hay **catorce** números así que no tenemos sólo uno en el medio, sino un par:

3, 5, 7, 12, 13, 14, **21**, **23**, 23, 23, 23, 29, 40, 56

En este ejemplo los números intermedios son **21 y 23**.

Para calcular el valor en medio de ellos, sumamos y dividimos entre 2:

21 + 23 = 44
44 ÷ 2 = 22

Así que la **mediana** en este ejemplo es **22**.

Ejemplo 2*

***Ejemplos obtenidos de: https://www.disfrutalasmatematicas.com/datos/mediana.html**

Siguiendo esta lógica, obtenemos la mediana filtrando si el largo del array es divisible de manera entera entre dos o no. [O(1)]
Se imprime el resultado.

La complejidad total = O(n)+O(n^2)+O(1) = **O(n^3)**

Apartado B

```
def divideB(array,brray,inicio,fin):
    #metodo que utiliza divide and conquer para dividir al caso unidad
    #los arrays e ir comparando las posiciones
    #llama al metodo imprimirB
    if inicio==fin : sonDistintos = imprimirB(array[fin],brray[fin])
    else:
        ma = int((inicio+fin)/2)
        sonDistintos = ma1=divideB(array,brray,inicio,ma)
        sonDistintos=ma2=divideB(array,brray,ma+1,fin);
    return sonDistintos

def compararLargoB(a,b):
#metodo que compara si los array tienen el mismo tamano y si es asi llama a divideB
    if(len(a) == len(b)):
        print("Los arrays tienen el mismo largo")
        sonDistintos = divideB(a,b,0,len(a)-1)
    else :
        print("Los arrays tienen distinto largo")
        sonDistintos = 1
    return sonDistintos

def imprimirB (a,b):
    #imprime los resultados y retorna el dato "sonDistintos"
    #que se iguala a 0 si los arrays tienen son iguales y 1 si son distintos
    if a==b:
        print("a:",a , " es igual a b:" , b)
        sonDistintos=0
        print("Iguales\n")
    else:
        print("a:",a , " es distinto a b:" , b)
        sonDistintos=1
        print("Distintas\n");
    return sonDistintos

#ejecucion
        a=[1,2,3]
        b=[1,2,4]
        sonDistintos = compararLargoB(a,b)
        if sonDistintos== 1 : print("Las listas son DISTINTAS")
        else: print("Las listas son IGUALES")
```

El apartado B consta de 3 métodos:

1. ***compararLargoB*** **:** Compara si la longitus del array a y el array b son las misma. Si son distinta retorna el valor sonDistintos a 1, lo que hace que se imprima directamente que los arrays son distintos. Si tienen el mismo tamaño, llamará a la función divideB.

2. ***divideB*** **:** Utiliza divide and conquer para reducir los arrays al caso unidad y así comparar las posiciones individualmente. Se llama recursivamente a sí misma para realizar esta operación.

3. ***imprimirB*** **:** Imprime los resultados y declara “sonDistintos” a 0 si los arrays no son distintos, es decir, si son iguales; y lo declara a 1 si los arrays son distintos. Gracias a la variable sonDistintos se imprimie el mensaje final para saber si los arrays son distintos o iguales.

Debido a la recursividad que se aplica en el método divideB para la variable ma1 y ma2, la complejidad del ejercicioB es de **O(n^2).**

Apartado C

```
def CtrasponerDC(m , fInicio, fFin, cInicio, cFin):
    if(fInicio<fFin):
        fmitad = int((fInicio+fFin)/2)
        cmitad = int((cInicio+cFin)/2)
        CtrasponerDC(m, fInicio, fmitad, cInicio, cmitad)
        CtrasponerDC(m, fInicio, fmitad, cmitad+1, cFin)
        CtrasponerDC (m, fmitad+1, fFin, cInicio, cmitad)
        CtrasponerDC (m, fmitad+1, fFin, cmitad+1, cFin)
        mtx =Cintercambiar (m, fmitad+1, cInicio, fInicio, cmitad+1,
fFin-fmitad)
        return mtx

def Cintercambiar(m, fIniA, cIniA,fIniB,cIniB,dimen):
    i = 0
    j = 0
    while i<= dimen-1:
        while j<= dimen-1:
            aux = m[fIniA+i][cIniA+j]
            m[fIniA+i][cIniA+j] = m[fIniB+i][cIniB+j]
            m[fIniB+i][cIniB+j] = aux
            i+=1
            j+=1
        return m

 #ejecucion
        m=[[1,2],[3,4]]
        print("Matriz: ",m)
        mtx = CtrasponerDC(m, 0, len(m)-1, 0, len(m)-1)
        print("Matriz traspuesta: ",mtx)
```

El apartado C utiliza dos métodos. *CtrasponerDC* [O(n^4]es un método que usa el algoritmo divide y vencerás con las columnas y las filas de la matriz para poder hallar el caso base y utilizar el método *Cintercambiar* [O(n^2] para intercambiar los componentes de la matriz cuadrada para formar la matriz traspuesta con la ayuda de una variable aux que guarda el valor original de la variable antes de intercambiarlo para podérselo asignar a la posición correspondiente.

La complejidad obtenida es elevada: **O(n^6)**

Apartado D

```
listaInicial = []
picosyValles = []
cuenta=0
p=0
v=0

def Drandom(lista,j):
    n = -1 #variable para comprobar que dos numeros consecutivos no sean iguales
    while j < 10 :
        num = randint(0, 10)
        j+=1
        if n != num:
            n = num
            lista.append(num)
        else:
            Drandom(lista,j)


def DgenerarArray():
    lista =[] #array vacio que se rellenara aleatoriamente
    Drandom(lista,0)
    return lista

def Dcomparar(valor):
    global p
    global v
    global posp
    global posv
    if (cuenta < len(listaInicial)-1):
        Dcontador()
        if valor > listaInicial[cuenta] and valor > listaInicial[cuenta-2]:
            if cuenta-1 != 0:
                print("pico en la posicion ", cuenta-1)
                #p sirve para saber si no se encuentra otro pico antes de encontrar
un valle
                #posp sirve para guardar la posicion del pico
                if p==0:
                    p=1
                    posp=cuenta-1
                else: p=0
        if valor < listaInicial[cuenta] and valor < listaInicial[cuenta-2]:
            if cuenta-1 != 0:
                print("valle en la posicion ", cuenta-1)
                #v sirve para saber si no se encuentra otro valle antes de
encontrar un pico
                #posv sirve para guardar la posicion del valle
                if v==0:
                    v=1
                    posv=cuenta-1
                else: v=0
        if(v==1 and p ==1):
            dato = listaInicial[posp]-listaInicial[posv]
            print("La diferencia del valle y el pico es: " , dato)
print("-----------------------------------------------------------------
------------------\n")
menu()
```

```
def Ddivide(lista,inicio,fin):
    if inicio == fin:
        n1 = Dcomparar(lista[fin])
    else:
        mitad = int((inicio+fin)/2)
        n1 = Ddivide(lista,inicio,mitad)
        n1 = Ddivide(lista,mitad+1,fin)
    return(lista[fin])


def Dcontador():
    global cuenta
        cuenta+=1

#ejecucion
        array = DgenerarArray()
        print(array)
        global listaInicial
        listaInicial = array
        Ddivide(array, 0, len(array)-1)
```

En el ejercicio C se genera un array de forma aleatoria con el método *DgenerarArray*[O(1)] y *Drandom*[O(n^2)] que controlan también que los números consecutivos no sean nunca repetidos, guardando el valor del anterior y comparándolo con el nuevo número generado, si el número es igual al anterior, se llama recursivamente a sí misma la función y genera un nuevo número random.

Ese array generado se guarda en *listaInicial* y se llama al método *Ddivide*[O(n^2)] que es un algoritmo de divide y vencerás que parte el array hasta el caso unidad.

Cuando llega al caso unidad llama al método *Dcomparar*[O(1)] que lo que hace es, gracias a un contador que genera *Dcontador*[O(1)] , comprar el valor anterior y el siguiente para ver si son mayores o menores al número obtenido en el caso unidad que nos dio el método Ddivide. Si el número anterior es menor y el siguiente es también es menor, habremos encontrado un pico. Si el número anterior y el siguiente son mayores, habremos encontrado un valle. Cada vez que encontramos un pico la variable *p* se pone a 1, y cada vez que encontramos un valle la variable *v* se pone a 1.
Cuando tengamos *p* y *v* a 1 tan solo nos queda restar el valor de las posiciones que hemos guardado donde se han generado el pico y el valle y el problema está resuelto.

La complejidad obtenida es de **O(n^4)**

Enlace GitHub

https://github.com/crisselene/Actividad2TPA