Universidad Europea

Memoria realización ejercicio 2 Técnicas de Programación Avanzada

Arturo Alba Sánchez-Mayoral

Nº expediente: 21838473

Repo GitHub: https://github.com/ArtySaurio/Actv2TPAE

EJERCICIO A

```python
def mediana(n):
    vr1 = np.random.randint(1, 30, n)
    vr2 = np.random.randint(1, 30, n)
    vr1.sort()
    vr2.sort()

    print("\nVector aleatorio 1:")
    print(vr1)
    print("\nVector aleatorio 2:")
    print(vr2)

    #Ordenamos y concatenamos con .concatenate el array
    v = np.concatenate([vr1, vr2])
    v.sort()
    print(v)
    i = int(len(v) / 2)
    mediana = ((v[i - 1] + v[i]) / 2)
    print("Mediana:", mediana)
```

EJERCICIO B

```python
def veceq(n):
    vr3 = np.random.randint(1, 30, n)
    vr4 = np.random.randint(1, 30, n)

    print("\nVector aleatorio 3:")
    print(vr3)
    print("\nVector aleatorio 4:")
    print(vr4)
    return vecrec(vr3, vr4)

#Dividimos array en otros dos arrays iguales
def divarrays(arr):
    cont = 1
    arr1 = []
    arr2 = []

    for i in arr:
        if (cont <= (len(arr) / 2)):
            arr1.append(i)
        else:
            arr2.append(i)
        cont += 1
    res = [arr1, arr2]
    return res

#Aplicamos divide y venceras comprobando los nmors
def vecrec(arr, brr):
    if (len(arr) == 1):
        if (arr == brr):
            return True
        else:
            return False
    else:
        res1 = divarrays(arr)
        res2 = divarrays(brr)
        if (vecrec(res1[0], res2[0]) and vecrec(res1[1], res2[1])):
            return True
        else:
            return False
```

EJERCICIO C

```python
def mtraspuesta(A):
    p = A.shape[0]
    q = A.shape[1]

    if p == 0 or q == 0:
        return A
    elif p == 1:
        return fila_columna(A)
    elif q == 1:
        return columna_fila(A)
    else:
        A11 = A[:p // 2, :q // 2]
        A12 = A[:p // 2, q // 2:]
        A21 = A[p // 2:, :q // 2]
        A22 = A[p // 2:, q // 2:]

        AT = np.vstack((np.hstack((mtraspuesta(A11),mtraspuesta(A21))),np.hstack((mtraspuesta(A12),mtraspuesta(A22)))))
        return AT

#Cast filas a columnas con base de origen y venceras
def fila_columna(A):
    p = A.shape[0]
    q = A.shape[1]

    if p == 1 and q == 1:
        return A
    else:
        return np.vstack((A[:, 0:1],
                          fila_columna(A[:, 1:])))

#Cast columnas a filas con base de origen y venceras
def columna_fila(A):
    p = A.shape[0]
    q = A.shape[1]

    if p == 1 and q == 1:
        return A
    else:
        return np.hstack((A[0:1, :],
                          columna_fila(A[1:, :])))
```

EJERCICIO D

```python
def peak_valley(n):
    print("\nCreo y verifico array")

    arr = creamosarray(n)
    loops = 0
    arr_verified = verifarray(arr, loops)
    print_array(arr_verified)

    print("\nCarga de los picos y los valles")
    peakvalleydc(arr_verified)
    draw_data(arr_verified)
    max_dist = find_distdc(arr_verified, arr_verified)

    print("\nMax distancia entre los picos y los valles: ", max_dist)
    plt.show()

#Creamos clase y almacenamos dentro los valores de antes y despues para saber distinguir entre pico y valle
class nmor:
    def __init__(self, data, prev, next, indice):
        self.data = data
        #Valor previo y valor siguiente
        self.prev = prev
        self.next = next
        self.indice = indice
        #Tipo o pico o valle
        self.tipo = None

    def __str__(self):
        template = "data:{} prev:{} next:{} indice:{} tipo:{}"
        return template.format(self.data, self.prev, self.next, self.indice, self.tipo)

    def __repr__(self):
        return str(self)
```

```
157 def creamosarray(n):
158     arrayaleat = np.random.randint(20, size=n)
159     numesarray = []
160 
161     for i in range(n):
162         if (i == 0):
163             numb = nmor(arrayaleat[i], None, arrayaleat[i + 1], i)
164         elif (i == (n - 1)):
165             numb = nmor(arrayaleat[i], arrayaleat[i - 1], None, i)
166         else:
167             numb = nmor(arrayaleat[i], arrayaleat[i - 1], arrayaleat[i + 1], i)
168         numesarray.append(numb)
169     return numesarray
170 
171 #Comprobamos que el array cumple requisitos
172 def comprobarray(a):
173 
174     for i in a:
175         if (i.data == i.next):
176             return False
177     return True
178 #Creamos funcion para verificar array, si no cumple requisitos loop de generarlo
179 def verifarray(a, loops):
180 
181     if (comprobarray(a)):
182         print("\nVeces necesarias para validar el array: ", loops)
183         print("\nEl array ha sido verificado correctamente")
184         return a
185     else:
186         loops += 1
187         new_arr = creamosarray(len(a))
188         return verifarray(new_arr, loops)
189 
190 #Funcion que imprime el array por pantalla
191 def print_array(arrn):
192     sample = []
193     for i in arrn:
194         sample.append(i.data)
195     print(sample)
```

```
198     def split_array(a):
199         half = len(a) // 2
200         res = [a[:half], a[half:]]
201         return res
202 
203     #Funcion que asigna si el valor del array es unb pico o un valle
204     def peakvalleydc(a):
205         if (len(a) == 1):
206             numb = a[0]
207             if (numb.prev == None or numb.next == None):
208                 pass
209             elif (numb.prev < numb.data > numb.next):
210                 numb.tipo = 'pico'
211             elif (numb.prev > numb.data < numb.next):
212                 numb.tipo = 'valle'
213         else:
214             res = split_array(a)
215             peakvalleydc(res[0])
216             peakvalleydc(res[1])
```

```python
def find_distdc(arr, full_arr):
    l = 0
    r = 0
    if (len(arr) == 1):
        num = arr[0]
        if (num.tipo == 'pico'):
            l = find_point(num, 1, full_arr, 'valle')
            r = find_point(num, -1, full_arr, 'valle')
        elif (num.tipo == 'valle'):
            l = find_point(num, 1, full_arr, 'pico')
            r = find_point(num, -1, full_arr, 'pico')
        else:
            return 0
    else:
        res = split_array(arr)
        l = find_distdc(res[0], full_arr)
        r = find_distdc(res[1], full_arr)
    if (l > r):
        return l
    else:
        return r

#Funcion que encuentra el maximo punto lejano entre un pico o un valle
def find_point(num, orientation, full_arr, tipo):
    if (orientation == -1):
        for i in range(1, num.indice):
            if (full_arr[num.indice - i].tipo == tipo):
                return i
    else:
        for i in range(1, len(full_arr) - num.indice):
            if (full_arr[num.indice + i].tipo == tipo):
                return i
    return 0
```

```python
def draw_data(a):
    x = []
    for i in range(len(a)):
        x.append(i)
    y = []
    for j in a:
        y.append(j.data)
    fig = plt.figure(figsize=(10, 6))
    plt.plot(x, y, color='red', linestyle='-', linewidth=3, marker='o', markerfacecolor='blue', markersize=8)

    plt.ylim(0, 21)
    plt.xlim(-1, len(a))

    for k in range(len(a)):
        label = "data:{} ({})".format(a[k].data, a[k].tipo)
        plt.annotate(label, (k, a[k].data), textcoords="offset points", xytext=(0, 10), ha='center')

    plt.xlabel('Indice')
    plt.ylabel('Dato')

    plt.title('Distancia maxima entre picos y valles')
    ax = plt.axes()
    ax.set_facecolor("white")
    plt.show(block=False)
```

EJERCICIO E

```
281    def genMatrix(n):
282        newMarix = []
283
284        for i in range(0,n):
285            dataline = []
286            for j in range(0,n):
287                dataline.append(randint(0,10))
288
289            for j in range(0, n):
290                Vals = list(map(float, dataline))
291
292            newMarix.append(Vals)
293        return newMarix
294
295    def split(matrix):
296
297        row, col = matrix.shape
298        row2, col2 = row//2, col//2
299        return matrix[:row2, :col2], matrix[:row2, col2:], matrix[row2:, :col2], matrix[row2:, col2:]
300
301    def strassen(x,y):
302
303        if len(x) == 1:
304            return x * y
305
306        a, b, c, d = split(x)
307        e, f, g, h = split(y)
308
309        p1 = strassen(a, f - h)
310        p2 = strassen(a + b, h)
311        p3 = strassen(c + d, e)
312        p4 = strassen(d, g - e)
313        p5 = strassen(a + d, e + h)
314        p6 = strassen(b - d, g + h)
315        p7 = strassen(a - c, e + f)
316        c11 = p5 + p4 - p2 + p6
317        c12 = p1 + p2
318        c21 = p3 + p4
319        c22 = p1 + p5 - p3 - p7
320        c = np.vstack((np.hstack((c11, c12)), np.hstack((c21, c22))))
321
322        return c
```