Universidad de Alcalá

ENTREGA DE EJERCICIOS DE LABORATORIO

TEMA 2: Algoritmos Voraces

Algoritmia y Complejidad, GII, 2A
Víctor Gamonal Sánchez
02317427Q

Problema 3

Para la resolución de este problema, evitamos recorrer la lista buscando el máximo y el mínimo, ya que haríamos 2n comprobaciones y sería un algoritmo inválido.

Es por ello por lo que recorreremos la lista pero cogiendo valores de dos en dos (para alcanzar los 3/2n), comparando dos a dos cuál es el mayor de la pareja y mandándolo a una lista de máximos; por otro lado, el otro valor irá a una lista de mínimos. Para recorrer esta lista inicial hemos usado n/2 comprobaciones.

Posteriormente, recorremos la lista de máximos y mínimos para encontrar el máximo y el mínimo, respectivamente, en cada una de ellas. Ya que estas dos listas tienen n/2 elementos, recorrerlas será n/2 comprobaciones, llegando así a la solución final en 3/2n comprobaciones.

En caso de que la lista sea impar cuando la lista inicial tenga len 1, comprobaremos ese valor con cualquiera de los máximos (o mínimos) para ver si es mayor o menor que ellos, respectivamente. En caso de que no se cumpla esta condición, se añadirá a la lista contraria con la que se comparó.

Código en Python:

```
def mayor_menor(Lista):
    minimo = []
    maximo = []

    while (len(Lista) > 1):
        if (Lista[0] > Lista[1]):
            maximo.append(Lista[0])
            minimo.append(Lista[1])
            del(Lista[0])
            del(Lista[0])

        else:
            maximo.append(Lista[1])
            minimo.append(Lista[0])
            del(Lista[0])
            del(Lista[0])

        if (len(Lista) == 1):
            if(Lista[0]) > maximo[0]:
                maximo.append(Lista[0])
                del(Lista[0])

            else:
                minimo.append(Lista[0])
                del(Lista[0])

        maxM = maximo[0]
        for i in maximo:
            if (i > maxM):
                maxM = i

        minM = minimo [0]
        for i in minimo:
            if (i < minM):
                minM = i

    print("El valor MÍNIMO es: ", minM)
    print("El valor MÁXIMO es: ", maxM)

Lista = [2, 7, 15, 0, 200, 87, 151]
mayor_menor(Lista)
```

Como podemos comprobar, el resultado es coherente con los datos introducidos en la lista.

```
 RESTART: C:/Users/Víctor/Desktop/UNIVERSIDAD/
mia y Complejidad/T2_Ej3_Voraces.py
El valor MÍNIMO es:  0
El valor MÁXIMO es:  200
```

Problema 4

```
import math

def AlgoritmoPrim(matrixPrim):
    distanciaMin = [0]
    masProximo = [0]
    T = []
    conocidos = []

    for i in range (1, len (matrixPrim)):
        masProximo.append(0)
        distanciaMin.append(matrixPrim[i][0])

    for i in range(len(matrixPrim) - 1):
        min = math.inf

        for j in range (1, len(matrixPrim)):
            if (0 <= distanciaMin[j] and distanciaMin[j] < min and j not in conocidos):
                min = distanciaMin[j]
                k = j

        T.append([masProximo[k]+1, k+1])
        conocidos.append(k)

        for j in range (1, len(matrixPrim)):
            if matrixPrim[j][k] < distanciaMin[j]:
                distanciaMin[j] = matrixPrim[j][k]
                masProximo[j] = k

    return T

w = math.inf

matrix = [[0, w, 4, w, 5, 1, w],
          [w, 0, 3, 6, 2, w, w],
          [4, 3, 0, 5, w, w, 1],
          [w, 6, 5, 0, 9, w, 7],
          [5, 2, w, 9, 0, w, 5],
          [1, w, w, w, w, 0, 3],
          [w, w, 1, 7, 5, 3, 0]]

print(AlgoritmoPrim(matrix))
```

```
 RESTART: C:/Users/Víctor/Desktop/UNIVERSIDAD/Curso 19-20/
mia y Complejidad/T2_Ej4_Prim.py
[[1, 6], [6, 7], [7, 3], [3, 2], [2, 5], [3, 4]]
```

Problema 6

Para la primera fase del código: bucle for con ordenación (coste $n^2 \log(n)$)

```
def Escalar (listaShrek):
    coste = 0
    listaShrek.sort()

    for i in range(len(listaShrek)-1):
        suma = listaShrek[0] + listaShrek[1]
        coste = coste + suma

        del(listaShrek[0])
        del(listaShrek[0])

        listaShrek.insert(busquedaBinaria(listaShrek, suma), suma)

        print ("El coste es: ", coste)

    return coste
```

Para la segunda fase, el coste es de $n \log(n)$ ya que solo se realiza una ordenación ($n \log(n)$) y n-1 inserciones binarias ($\log(n)$).

```
def busquedaBinaria(lista, item):
    if len(lista) == 0:
        return 0
    else:
        puntoMedio = len(lista)//2
        if lista[puntoMedio-1] < item and lista[puntoMedio] > item:
            return puntoMedio
        else:
            if item < lista[puntoMedio]:
                return busquedaBinaria(lista[:puntoMedio], item)
            else:
                return busquedaBinaria(lista[puntoMedio+1:], item)+puntoMedio+1
```

La solución devuelta por ambos algoritmos es la misma, pero el tiempo de ejecución será diferente ya que la del segundo tiene menor coste.

```
 RESTART: C:/Users/Víctor/Desktop/UNIVE
mia y Complejidad/T2_Ej6_ShrekEscala.py
El coste es: 7
El coste es: 22
El coste es: 46
El coste es: 95
El coste es: 171
El coste es: 280
El coste es: 418
El coste es: 665
El coste es: 1015
El coste es: 1612
El coste final es: 1612
```