Grabación de programas

Metodología de la programación y algoritmia

Javier Maciá Sempere

2020 / 2021

ÍNDICE

[1. ALgoritmo para grabar el mayor número de programas. 1](#_Toc74434368)

[1.1- Descripción y tipificación 1](#_Toc74434369)

[1.2- Estrategia de programación 1](#_Toc74434370)

[1.3- Pseudocódigo 2](#_Toc74434371)

[Mergesort 2](#_Toc74434372)

[Merge 2](#_Toc74434373)

[1.4- Ejemplo 4](#_Toc74434374)

[1.5- Complejidad asintótica 5](#_Toc74434375)

[2. ALgoritmo para grabar el máximo espacio en la memoria. 8](#_Toc74434376)

[2.1- Descripción y tipificación 8](#_Toc74434377)

[2.2- Estrategia de programación 8](#_Toc74434378)

[2.3- Pseudocódigo 8](#_Toc74434379)

[2.4- Ejemplo 9](#_Toc74434380)

[2.5- Complejidad asintótica 10](#_Toc74434381)

[3. Bibliografía 11](#_Toc74434382)

[1. Inspiración para buscar algoritmos óptimos (Stackoverflow, Github y GeeksForGeeks) 11](#_Toc74434383)

[2. Apuntes de la asignatura consultados: 11](#_Toc74434384)

# 1. ALgoritmo para grabar el mayor número de programas.

## 1.1- Descripción y tipificación

El programa que se presenta tiene la función de ordenar un listado de programas utilizando dos métodos diferentes. El primer método graba el mayor número de programas posibles.

Para conseguir este objetivo, primero se solicita al usuario que introduzca un fichero donde se encontrarán los programas, así como también el espacio de memoria del que dispone. Luego, crearemos dos listas: una contendrá los nombres de los programas, y otra el peso ligado a ellas. Para introducir el máximo número de programas en esa memoria, será tan sencillo como ordenar dichos programas e ir introduciéndolos de menor a mayor.

## 1.2- Estrategia de programación

Para esta tarea de ordenación, he empleado el algoritmo de “Mergesort”. Este algoritmo consiste el dividir la lista en dos mitades. Estas dos sublistas, las volveremos a dividir en dos mitades, y así sucesivamente. Finalmente, cuando no podamos dividir más, ordenaremos ambas sublistas resultantes, y las juntaremos. Luego, volveremos a ordenar esta unión, y las juntaremos. Así, sucesivamente.

Para ello, he utilizado dos funciones: “merge” y “mergesort”. La primera, se encargará de dividir y ordenar, y la segunda se encargará de crear las sublistas y llamar a la primera.

## 1.3- Pseudocódigo

### Mergesort

función mergeSort( arr:real[], izq:entero, der:entero, progs:cadena[] )

med:entero

si (izq >=der)

devolver

fsi

med = izq + (der-izq)/2

mergeSort(arr,izq,med,progs)

mergeSort(arr,med+1,der,progs)

merge(arr,izq,med,der,progs)

ffuncion

### Merge

función merge(arr:real[], med:entero, der:entero, progs:cadena[])

n1, n2: enteros

i, j, k: enteros

L:real[n1] , R:real[n2]

Lp:cadena[n1], Rp:cadena[n2]

n1 <- med – izq +1

n2 <- der – med

k <- izq

para i <- 0 hasta n1 hacer

Li = arrizq+i

Lpi = progsizq+i

fpara

para j <- 0 hasta n2 hacer

Ri = arrmed+1+j

Rpi = progsmed+1+j

fpara

mientras i<n1 y j<n2 hacer

si Li <= Rj

arrk = Li

progsk = Lpi

fsi

si no

arrk = Rj

progsk = Rpj

fsi

k <- k + 1

fmientras

mientras i<n1 hacer

arrk = Li

progsk = Lpi

i <- i + 1

k <- k +1

fmientras

mientras j<n2 hacer

arrk = Rj

progsk = Rpj

j <- j + 1

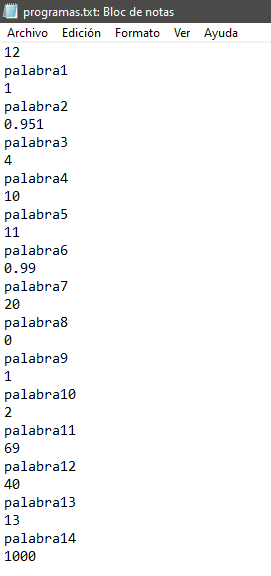
k <- k +1

fmientras

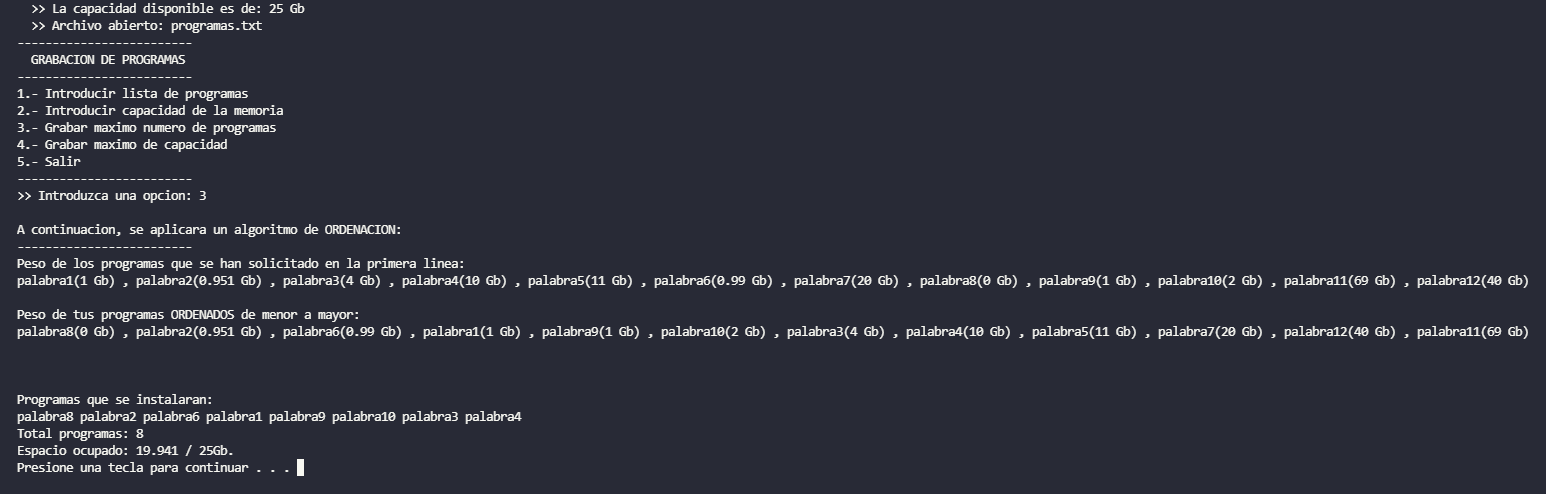
ffuncion

## 1.4- Ejemplo

A lo largo de esta práctica vamos a usar este texto para probar el programa:



**Es importante que nos fijemos en el detalle de que arriba del todo pone 12. Esto quiere decir que el programa no contemplará ni 13 ni 14**. Introduciremos, además, 25 Gb como espacio disponible.



**Output por si el texto no es fácilmente legible:**

***>> La capacidad disponible es de: 25 Gb***

***>> Archivo abierto: programas.txt***

***-------------------------***

***GRABACION DE PROGRAMAS***

***-------------------------***

***1.- Introducir lista de programas***

***2.- Introducir capacidad de la memoria***

***3.- Grabar maximo numero de programas***

***4.- Grabar maximo de capacidad***

***5.- Salir***

***-------------------------***

***>> Introduzca una opcion: 3***

***A continuacion, se aplicara un algoritmo de ORDENACION:***

***-------------------------***

***Peso de los programas que se han solicitado en la primera linea:***

***palabra1(1 Gb) , palabra2(0.951 Gb) , palabra3(4 Gb) , palabra4(10 Gb) , palabra5(11 Gb) , palabra6(0.99 Gb) , palabra7(20 Gb) , palabra8(0 Gb) , palabra9(1 Gb) , palabra10(2 Gb) , palabra11(69 Gb) , palabra12(40 Gb)***

***Peso de tus programas ORDENADOS de menor a mayor:***

***palabra8(0 Gb) , palabra2(0.951 Gb) , palabra6(0.99 Gb) , palabra1(1 Gb) , palabra9(1 Gb) , palabra10(2 Gb) , palabra3(4 Gb) , palabra4(10 Gb) , palabra5(11 Gb) , palabra7(20 Gb) , palabra12(40 Gb) , palabra11(69 Gb)***

***Programas que se instalaran:***

***palabra8 palabra2 palabra6 palabra1 palabra9 palabra10 palabra3 palabra4***

***Total programas: 8***

***Espacio ocupado: 19.941 / 25Gb.***

## 1.5- Complejidad asintótica

función merge(arr:real[], med:entero, der:entero, progs:cadena[])

n1, n2: enteros \

i, j, k: enteros |

L:real[n1] , R:real[n2] |

Lp:cadena[n1], Rp:cadena[n2] | O(1)

n1 <- med – izq +1 |

n2 <- der – med |

k <- izq /

para i <- 0 hasta n1 hacer \

Li = arrizq+i | O(n)

Lpi = progsizq+i |

fpara /

para j <- 0 hasta n2 hacer \

Ri = arrmed+1+j | O(n)

Rpi = progsmed+1+j |

fpara /

mientras i<n1 y j<n2 hacer \

si Li <= Rj \ |

arrk = Li |O(1) |

progsk = Lpi | |

fsi / |

si no \ | O(n)

arrk = Rj | |

progsk = Rpj | O(logn) |

fsi | |

k <- k + 1 / |

fmientras /

mientras i<n1 hacer \

arrk = Li |

progsk = Lpi | O(nlogn)

i <- i + 1 |

k <- k +1 |

fmientras /

mientras j<n2 hacer \

arrk = Rj |

progsk = Rpj | O(nlogn)

j <- j + 1 |

k <- k +1 |

fmientras /

ffuncion

**RESULTADO:** Merge tiene una complejidad de O(nlogn).

función mergeSort( arr:real[], izq:entero, der:entero, progs:cadena[] )

med:entero \

si (izq >=der) |

devolver | O(1)

fsi |

med = izq + (der-izq)/2 /

mergeSort(arr,izq,med,progs) \

mergeSort(arr,med+1,der,progs) | O(nlogn)

merge(arr,izq,med,der,progs) /

ffuncion

**RESULTADO:** MergeSort tiene una complejidad de O(nlogn).

# 2. ALgoritmo para grabar el máximo espacio en la memoria.

## 2.1- Descripción y tipificación

Este segundo método consiste en tratar de llenar lo máximo posible la memoria. Este problema se puede resolver de dos maneras: si tienes 25Gb libres, y cuatro programas (5Gb, 10Gb, 10Gb, 25Gb) podemos instalar los 25Gb y cumpliríamos el objetivo, o podemos instalar 5, 10 y 10Gb y además de cumplir el objetivo habríamos instalado más programas. Yo he optado por esta última opción.

## 2.2- Estrategia de programación

Para resolver este problema, he empleado un algoritmo de la mochila. Concretamente uno en el cuál el valor de cada programa y su peso es el mismo. Este algoritmo lo que hará será crear una tabla en la cuál se evaluará qué programas son más óptimos para ser instalados, y cuál es el peso que ocupa instalar todos estos programas. Esto lo hará comparando los resultados obtenidos según se añadan unos pesos u otros. También, para facilitar las cosas, he reciclado el algoritmo de MergeSort para tener los pesos ordenados.

## 2.3- Pseudocódigo

función Mochila(progsAInstalar:entero, espacioLib:real, números:real[],X:entero{0,1}[], k:entero, Xóptimo:entero{0,1}[], Vóptimo:&real)

i, j:enteros

peso, valor:reales

si(k<=progsAInstalar)

para i<-0 hasta 1 hacer

Xk <- i

peso = CalcularSuma(X,números,k)

si (peso<=espacioLib)

si (k = progsAInstalar)

valor = CalcularSuma(X,números,k)

si (valor>Voptimo)

para i <- 0 hasta progsAInstalar

Xoptimoi = Xi

fpara

Voptimo = valor

fsi

si no

Mochila(progsAInstalar,espacioLib,números,X,k+1,Xoptimo,Voptimo)

fsi

fsi

fpara

fsi

ffuncion

## 2.4- Ejemplo

**Output por si el texto no es fácilmente legible:**

***>> La capacidad disponible es de: 25 Gb***

***>> Archivo abierto: programas.txt***

***-------------------------***

***GRABACION DE PROGRAMAS***

***-------------------------***

***1.- Introducir lista de programas***

***2.- Introducir capacidad de la memoria***

***3.- Grabar maximo numero de programas***

***4.- Grabar maximo de capacidad***

***5.- Salir***

***-------------------------***

***>> Introduzca una opcion: 4***

***A continuacion, se aplicara un algoritmo de MOCHILA:***

***-------------------------***

***Peso de los programas que se han solicitado en la primera linea:***

***palabra1(1 Gb) , palabra2(0.951 Gb) , palabra3(4 Gb) , palabra4(10 Gb) , palabra5(11 Gb) , palabra6(0.99 Gb) , palabra7(20 Gb) , palabra8(0 Gb) , palabra9(1 Gb) , palabra10(2 Gb) , palabra11(69 Gb) , palabra12(40 Gb)***

***Programas que se instalaran:***

***palabra3(4 Gb) palabra4(10 Gb) palabra5(11 Gb)***

***Se pueden almacenar 25 Gb /25 Gb.***

***Total programas: 3***

***Presione una tecla para continuar . . .***

## 2.5- Complejidad asintótica

función Mochila(progsAInstalar:entero, espacioLib:real, números:real[],X:entero{0,1}[], k:entero, Xóptimo:entero{0,1}[], Vóptimo:&real)

i, j:enteros

peso, valor:reales

si(k<=progsAInstalar)

para i<-0 hasta 1 hacer

Xk <- i \ O(n)

O(n)-> peso = CalcularSuma(X,números,k) /

si (peso<=espacioLib)

si (k = progsAInstalar)

valor = CalcularSuma(X,números,k) | O(n)

si (valor>Voptimo) \

para i <- 0 hasta progsAInstalar |

Xoptimoi = Xi | O(n)

Fpara |

Voptimo = valor |

fsi /

si no \

Mochila(progsAInstalar,espacioLib,números,X, | k+1,Xoptimo,Voptimo) | O(2n)

fsi /

fsi

fpara

fsi

ffuncion

**RESULTADO:** Mochila tiene una complejidad de O(2n). Tiene una complejidad tan grande porque tiene subproblemas que son redundantes.

# 3. Bibliografía

## 1. Inspiración para buscar algoritmos óptimos (Stackoverflow, Github y GeeksForGeeks)

- <https://stackoverflow.com/questions/40721107/algorithm-for-filling-bag-maximally-this-is-not-the-knapsack-0-1>

- <https://marcodiiga.github.io/knapsack-problem>

- <https://www.geeksforgeeks.org/subset-sum-problem-dp-25/>

- <https://www.geeksforgeeks.org/merge-sort/>

- <https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/>

## 2. Apuntes de la asignatura consultados:

- Tema 1: Análisis de algoritmos.

- Tema 4: Algoritmos voraces.

- Tema 5: Programación dinámica.

- Tema 6: Algoritmos de vuelta atrás.

- Tema 7: Ramificación y poda.