

Grabación de programas

METODOLOGÍA DE LA PROGRAMACIÓN Y ALGORITMIA

JAVIER MACÍÁ SEMPERE

ÍNDICE

1. Algoritmo para grabar el mayor número de programas.	1
1.1- Descripción y tipificación.....	1
1.2- Estrategia de programación.....	1
1.3- Pseudocódigo.....	2
Mergesort.....	2
Merge	2
1.4- Ejemplo.....	4
1.5- Complejidad asintótica.....	5
2. Algoritmo para grabar el máximo espacio en la memoria.	8
2.1- Descripción y tipificación.....	8
2.2- Estrategia de programación.....	8
2.3- Pseudocódigo.....	8
2.4- Ejemplo.....	9
2.5- Complejidad asintótica.....	10
3. Bibliografía.....	11
1. Inspiración para buscar algoritmos óptimos (Stackoverflow, Github y GeeksForGeeks) ...	11
2. Apuntes de la asignatura consultados:.....	11

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 en 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

$L_i = arr_{izq+i}$

$Lp_i = progs_{izq+i}$

 fpara

```

para j <- 0 hasta n2 hacer

     $R_i = arr_{med+1+j}$ 

     $R_{p_i} = progs_{med+1+j}$ 

fpara

mientras i < n1 y j < n2 hacer

    si  $L_i \leq R_j$ 

         $arr_k = L_i$ 

         $progs_k = L_{p_i}$ 

    fsi

    si no

         $arr_k = R_j$ 

         $progs_k = R_{p_j}$ 

    fsi

    k <- k + 1

fmientras

mientras i < n1 hacer

     $arr_k = L_i$ 

     $progs_k = L_{p_i}$ 

    i <- i + 1

    k <- k + 1

fmientras

mientras j < n2 hacer

     $arr_k = R_j$ 

     $progs_k = R_{p_j}$ 

    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:

```
programas.txt: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
12
palabra1
1
palabra2
0.951
palabra3
4
palabra4
10
palabra5
11
palabra6
0.99
palabra7
20
palabra8
0
palabra9
1
palabra10
2
palabra11
69
palabra12
40
palabra13
13
palabra14
1000
```

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.

```
>> 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 línea:
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.
Presione una tecla para continuar . . . █
```

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	\	
L _i = arr _{izq+i}		O(n)
Lp _i = progs _{izq+i}		
fpara	/	
para j <- 0 hasta n2 hacer	\	
R _j = arr _{med+1+j}		O(n)
Rp _j = progs _{med+1+j}		
fpara	/	
mientras i < n1 y j < n2 hacer	\	
si L _i <= R _j	\	
arr _k = L _i		O(1)
progs _k = Lp _i		
fsi	/	
si no	\	O(n)
arr _k = R _j		
progs _k = Rp _j		O(logn)
fsi		
k <- k + 1	/	
fmientras		/
mientras i < n1 hacer	\	
arr _k = L _i		

progs _k = Lp _i	O(nlogn)
i <- i + 1	
k <- k + 1	
fmientras	/
mientras j < n/2 hacer	\
arr _k = R _j	
progs _k = Rp _j	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 $O(n \log n)$.

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

$X_k \leftarrow i$

 peso = CalcularSuma(X,números,k)

 si (peso<=espacioLib)

 si (k = progsAInstalar)

 valor = CalcularSuma(X,números,k)

 si (valor>Vóptimo)

 para i <- 0 hasta progsAInstalar

$X_{\text{óptimo}_i} = X_i$

 fpara

 Vóptimo = valor

 fsi

```

                                si no
                                Mochila(progsAlInstalar,espacioLib,números,X,k+1,Xoptimo,
                                Voptimo)
                                fsi
                        fsi
                fpara
        fsi
ffuncion

```

2.4- EJEMPLO

```

>> 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 . . . █

```

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 opción: 4

A continuación, se aplicará un algoritmo de MOCHILA:

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

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 instalarán:

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

$X_k \leftarrow i$ $\backslash O(n)$

$O(n) \rightarrow$ 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 $O(2^n)$. Tiene una complejidad tan grande porque tiene subproblemas que son redundantes.

3. BIBLIOGRAFÍA

1. INSPIRACIÓN PARA BUSCAR ALGORITMOS ÓPTIMOS (STACKOVERFLOW, GITHUB Y GEEKSFORGEES)

- <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.