

Septiembre-2015.pdf



CarlosGarSil98



Algorítmica y Modelos de Computación



3º Grado en Ingeniería Informática

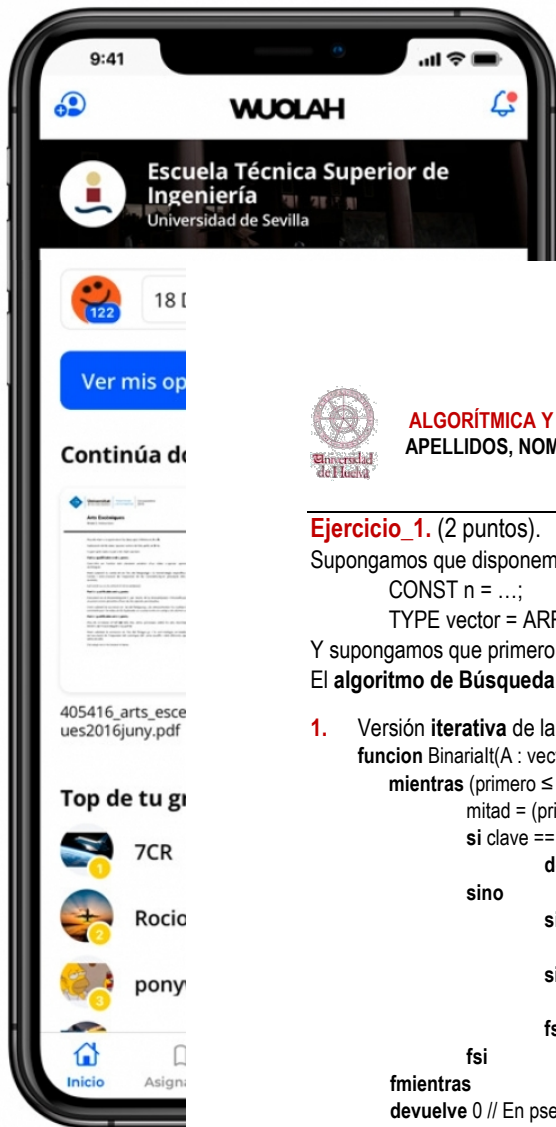


**Escuela Técnica Superior de Ingeniería
Universidad de Huelva**



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.





Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.



Universidad de Huelva. Escuela Técnica de Ingeniería. Departamento de Tecnologías de la Información.
ALGORÍTMICA Y MODELOS DE COMPUTACIÓN. 3º Grado Ingeniería Informática. La Rábida 7 de septiembre del 2015.
APELLIDOS, NOMBRE García Silva, Carlos NOTA _____

Ejercicio_1. (2 puntos).

Supongamos que disponemos de la siguiente definición de tipo:

CONST n = ...;

TYPE vector = ARRAY [1..n] OF INTEGER;

Y supongamos que primero y ultimo indican los límites del array (inicialmente primero=1 y ultimo=n)

El **algoritmo de Búsqueda Binaria**(dicotómica) puede ser implementado:

1. Versión **iterativa** de la Búsqueda binaria

funcion BinariaIt(A : vector, primero, ultimo, clave : entero)

mientras (primero ≤ ultimo) **hacer**

mitad = (primero + ultimo) / 2

si clave == A[mitad] **entonces**

devuelve mitad

sino

si clave > A[mitad] **entonces**

primero = mitad + 1

sino

ultimo = mitad - 1

fsi

fsi

fmientras

devuelve 0 // En pseudocódigo empieza en 1

ffuncion

2. Versión **recursiva** de la Búsqueda binaria:

funcion BinariaRc(A : vector, primero, ultimo, clave : entero)

si (primero > ultimo) **entonces**

devuelve 0

fsi

mitad = (primero + ultimo) / 2

si clave > A[mitad] **entonces**

devuelve BinariaRc(A, mitad+1, ultimo, clave)

sino

si clave < A[mitad] **entonces**

devuelve BinariaRc(A, primero, mitad-1, clave)

sino

devuelve mitad

fsi

fsi

ffuncion

Se pide:

- (0,5 puntos). Calcular la complejidad del algoritmo **iterativo** propuesto para el caso **peor** mediante el conteo del número de operaciones elementales.
- (0,5 puntos). Calcular la complejidad del algoritmo **recursivo** para el caso **peor** por el **Teorema maestro**.
- (0,5 puntos). Calcular la complejidad del algoritmo **recursivo** propuesto para el caso **peor** por el método de la **ecuación característica**.
- (0,5 puntos). Comprobar si ambas versiones, iterativa y recursiva, invierten el mismo tiempo.

NOTA: el Teorema maestro es:

$$T(n) \in \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \cdot \log^{p+1} n) & \text{si } a = b^k \\ O(n^k \cdot \log^p n) & \text{si } a < b^k \end{cases}$$

Apartado A:

El peor caso se da cuando el elemento no se encuentra porque no está, entonces se realizan todas las iteraciones posibles

$$T(n) = 1 + 1 + \sum_{i=1}^? (1 + 1 + 3 + T_{\text{cond}} + T_{\text{salto}} + T_{\text{verpo}}) + 1$$

Para el caso peor, el elemento a buscar siempre será menor que el elemento de la posición mitad, ya que es cuando más veces se ejecuta el if.

La última iteración posible es que primero == último. Como el algoritmo reduce a la mitad en cada iteración, sabemos que iniciará en $n, n/2, n/4, \dots$ es decir, potencias de 2, hasta $n/n = 1$; $n = 2^k$. El bucle iniciará en $i=0$ hasta $i = \log_2(n)$

$$T(n) = 1 + 1 + \sum_{i=0}^{\log_2(n)} (13) + 1; \quad T(n) = 3 + 13(\log_2(n) - 0 + 1);$$

$$T(n) = 13 \cdot \log_2(n) + 16 \in O(\log(n))$$

Apartado b:

Para el caso base, índices cruzados, se comprueba la condición y se devuelve 0
Para el caso general, concretamente el peor, hacer el recorrido hasta la llamada donde se cruzan los índices, ya que el elemento siempre será menor que el que se encuentra en la mitad.

$$T(n) = \begin{cases} 2 & \text{si } n = 0 \\ T(n/2) + 12 & \text{si } n > 0 \end{cases}$$

Fórmula del Teorema maestro: $T(n) = aT(n/b) + O(n^k \cdot \log^p(n))$

En este caso: $a=1, b=2, k=0, p=0$

$a > b^k$; $1 > 2^0 \rightarrow$ No se cumple

$a = b^k$; $1 = 2^0 \rightarrow$ si se cumple

$$T(n) \in O(\log(n))$$

Apartado C:

$$T(n) = \begin{cases} 2 & \text{si } n = 0 \\ T(n/2) + 12 & \text{si } n > 0 \end{cases}$$

$$T(n) - T(n/2) = 12 \left[\begin{array}{l} \text{cambio de base} \\ n = 2^k \end{array} \right] \quad T(2^k) - T(2^{k-1}) = 12 \quad \text{No Homogénea}$$



**KEEP
CALM
AND
ESTUDIA
UN POQUITO**

$$T(2^k) - T(2^{k-1}) \rightarrow (x-1)$$

$$b^k \cdot p(k)^d = 12 = 12 \cdot 1^k; \quad b=1, d=0 \rightarrow (x-1)^{0+1}$$

$$p(x) = (x-1)(x-1); \quad \text{Raíces: } r_1 = 1 \text{ doble}$$

$$T(2^k) = C_0 \cdot 1^k \cdot k^0 + C_1 \cdot 1^k \cdot k^1 \left[\begin{array}{c} \text{cambio de base} \\ 2^k = n \end{array} \right]_{k=\log(n)} \quad T(n) = C_0 + C_1 \log(n)$$

Se deberían de calcular las constantes, a partir de valores del caso base y hacia valores mayores. como son dos constantes, tendremos dos ecuaciones.

Este paso nos lo vamos a saltar ($C_0 = 14, C_1 = 12$)

$$T(n) = C_0 + C_1 \log(n) \rightarrow C_1 \neq 0 \rightarrow T(n) \in O(\log(n))$$

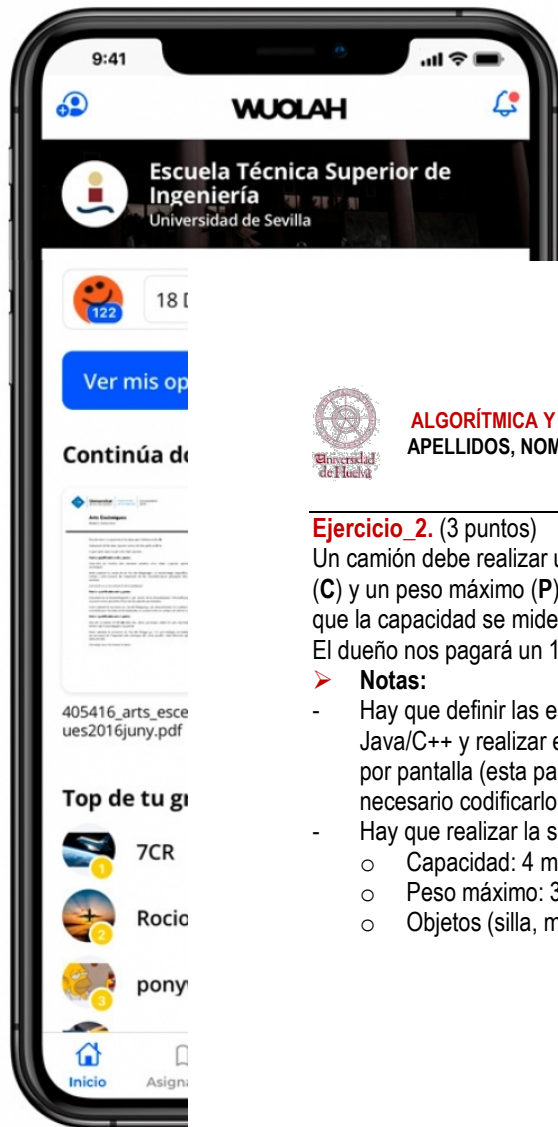
Apartado d:

Como ambos algoritmos son del mismo orden de complejidad, debemos comparar con los valores de las constantes

$$\text{iterativo} = 13 \cdot \log(n) + 16$$

$$\text{recursivo} = 12 \cdot \log(n) + 14$$

Los valores del iterativo son mayores, portanto, no invierten el mismo tiempo, pero sí similar.



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.



Universidad de Huelva. Escuela Técnica de Ingeniería. Departamento de Tecnologías de la Información.
ALGORÍTMICA Y MODELOS DE COMPUTACIÓN. 3º Grado Ingeniería Informática. La Rábida 7 de septiembre del 2015.
 APELLIDOS, NOMBRE García Silva, Carlos NOTA _____

Ejercicio 2. (3 puntos)

Un camión debe realizar un transporte optimizando la ganancia de este. El camión tiene una capacidad máxima (C) y un peso máximo (P) que no deben sobrepasarse. Los objetos van unos tras otros de manera lineal, por lo que la capacidad se mide en metros. Por otro lado, cada objeto tiene un beneficio (bi), longitud (Li) y peso (pi). El dueño nos pagará un 10% del beneficio transportado menos 30€ * número de objetos que dejemos fuera.

➤ Notas:

- Hay que definir las estructuras o clases necesarias para realizar el problema en un pseudocódigo basado en Java/C++ y realizar el programa de prueba que llama a la función de optimización y devuelve el resultado por pantalla (esta parte puede ser común a ambos). Si se utiliza algún algoritmo de ordenación, no es necesario codificarlo, sólo indicar qué tipo de algoritmo se usa.
- Hay que realizar la simulación para los datos dados y mostrar la solución propuesta
 - Capacidad: 4 metros lineales
 - Peso máximo: 35 kilos
 - Objetos (silla, mesa, arcón, cama, televisor, ordenador)

Armario:	1000€	2 metros	20 kilos
Mesa:	400€	1.5 metros	15 kilos
Arcón:	1100€	3 metros	12 kilos
Televisor:	250€	50 cms	5 kilos
Ordenador:	350€	50 cms	3 kilos

- a. (1.5 pts). Diseñar un algoritmo voraz para resolver el problema aunque no se garantice la solución óptima. Es necesario marcar en el código propuesto a qué corresponde cada parte en el esquema general de un algoritmo voraz (criterio, candidatos, función, ...). Si hay más de un criterio posible, elegir uno razonadamente y discutir los otros. Indicar **razonadamente el orden** de dicho algoritmo (no es necesario realizar el desarrollo completo, pero si no se razona, se puede realizar para demostrar el orden de manera matemática).
- b. (1.5 pts). Resolver el problema mediante **programación dinámica sin tener en cuenta el peso de cada objeto ni la penalización por objetos dejados fuera**. Definir la ecuación recurrente, los casos base, las tablas y el algoritmo para rellenarlas.

Ejercicio 3. (2 puntos).

Dado el AFND = ($\{a,b\}$, $\{p,q,r,s\}$, f , p , $\{s\}$) donde f viene dada por la siguiente tabla de transiciones:

	a	b	λ
$\rightarrow p$	{q, s}	{p}	{q, r}
q		{q, r}	{r}
r		{p, s}	{q}
* s	{s}	{q, r, s}	

Se pide:

- (1 punto). El AFD equivalente.
- (1 punto). El AFD mínimo.

Apartado a:

$$\rightarrow Q_0 = \lambda\text{-clausura}(p) = \{p, q, r\}$$

$$f'(Q_0, a) = \{q, s\} = \lambda\text{-clausura}(\{q, s\}) = \{q, s, r\} \rightarrow Q_1 \text{ estado final}$$

$$f'(Q_0, b) = \{p, q, r, s\} \rightarrow Q_2 \text{ Estado final}$$

$$* Q_1 = \{q, s, r\}$$

$$f'(Q_1, a) = \{s\} \rightarrow Q_3 \text{ Estado final}$$

$$f'(Q_1, b) = \{q, r, s, p\} \rightarrow Q_2$$

$$* Q_2 = \{p, q, r, s\}$$

$$f'(Q_2, a) = \{q, s, r\} \rightarrow Q_1$$

$$f'(Q_2, b) = \{p, q, r, s\} \rightarrow Q_2$$

$$* Q_3 = \{s\}$$

$$f'(Q_3, a) = \{s\} \rightarrow Q_3$$

$$f'(Q_3, b) = \{q, r, s\} \rightarrow Q_1$$

f	a	b
$\rightarrow Q_0$	Q_1	Q_2
* Q_1	Q_3	Q_2
* Q_2	Q_1	Q_2
* Q_3	Q_3	Q_1

Apartado b:

Agrupamos en no finales y finales

$$Q/E_0 = (C_0 = \{Q_0\}, C_1 = \{Q_1, Q_2, Q_3\})$$

$$f'(Q_0, a) = C_1 \quad f'(Q_0, b) = C_1$$

$$f'(Q_1, a) = C_1 \quad f'(Q_1, b) = C_1$$

$$f'(Q_2, a) = C_1 \quad f'(Q_2, b) = C_1$$

$$f'(Q_3, a) = C_1 \quad f'(Q_3, b) = C_1$$

f	a	b
$\rightarrow C_0$	C_1	C_1
* C_1	C_1	C_1

como todos coinciden, no hace falta seguir dividiendo

Ejercicio 4. (3 puntos)

Dada la siguiente gramática:

$S \rightarrow AS$
 $S \rightarrow \epsilon AS \mid \lambda$
 $A \rightarrow BA$
 $A \rightarrow AA''A'' \mid \lambda$
 $A \rightarrow +B \mid -B$
 $B \rightarrow (S) \mid a \mid b$

- (0.5 pts). Comprobar si es LL(1) mediante el cálculo de los conjuntos Primero y Siguiente.
- (1 pt). Implementar la tabla de análisis sintáctico y especificar el pseudocódigo de análisis sintáctico tabular.
- (0.75 pts). Construir la traza correspondiente al reconocimiento de la frase "a = b" según el pseudocódigo especificado en el apartado b anterior.
- (0.75 pts). Especificar el pseudocódigo de análisis sintáctico dirigido por la sintaxis.

Apartado a:

$S \rightarrow AS$
 $S \rightarrow \epsilon AS$
 $\mid \lambda$
 $A \rightarrow BA$
 $A \rightarrow AA''A''$
 $\mid \lambda$
 $A \rightarrow +B$
 $\mid -B$
 $B \rightarrow (S)$
 $\mid a$
 $\mid b$

	Primeros	Siguientes	Predicción	
S	(a b + -) \$	(a b + -	} intersección vacía
	=		=	
	λ) \$	
A	(= + - (a b " \$	(} intersección no vacía
	a		a	
	b		b	
	λ		= +- (a b \$	
	+ -		+ -	
B	(+ - (a b	(} intersección vacía
	a		a	
	b		b	

No se cumple la regla necesaria, por tanto, no se trata de una gramática LL(1).

Apartado b:

La tabla se obtiene mediante el siguiente algoritmo:

```

V A → α
  V 'a' terminal != λ ∈ PRin(α)
    Tabla[A, a] = α
  fin V
  si λ ∈ PRin(α)
    V 'b' terminal != λ ∈ sig(α)
      Tabla[A, a] = λ
    fin V
  fsi
fin V
  
```




Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.



```

procedimiento Analisis_tabular()
  Apilar(#);
  Apilar(S);      S = axioma
  Leer(simbolo);  preanalisis = simbolo
  mientras NOT pila_vacia hacer
    switch cima_pila
      case terminal:
        si cima_pila == simbolo entonces
          Desapilar(simbolo);
          Leer(simbolo);
        -sino
          error_sintactico();
        fsi
      case No_terminal:
        si Tabla(cima_pila, simbolo) != error entonces
          Desapilar(cima_pila);
          Apilar(Tabla(cima_pila, simbolo));
        -sino
          error_sintactico();
        fsi
    fswitch
  fmientras
  si cima_pila == # entonces
    Desapilar(#);
    Escribir(cadena_aceptada);
  -sino
    error_sintactico();
  fsi
fprocedimiento
  
```

Apartado c:

Pila	Entrada	Acción
λ	$a = b \$$	Apilar($\#$)
$\#$	$a = b \$$	Apilar(S)
S $\#$	$a = b \$$	$S \rightarrow AS$
A S $\#$	$a = b \$$	$A \rightarrow BA$
B A S $\#$	$a = b \$$	$B \rightarrow a$
a A S $\#$	$a = b \$$	Leer(a)
A S $\#$	$= b \$$	$A \rightarrow \lambda$
S $\#$	$= b \$$	$S \rightarrow =AS$
= A S $\#$	$= b \$$	Leer(=)
A S $\#$	b $\$$	$A \rightarrow BA$
B A S $\#$	b $\$$	$B \rightarrow b$
b A S $\#$	b $\$$	Leer(b)
A S $\#$	$\$$	$A \rightarrow \lambda$
S $\#$	$\$$	$S \rightarrow \lambda$
$\#$	$\$$	Desapilar($\#$)
λ	λ	Aceptar

Apartado d:

```
programa_Principal ()
    SLA = leer_simbolo();
    S();
    si SLA != $ entonces
        Error ();
    fsi
fprograma

procedimiento Reconocer (simbolo T)
    Si SLA == T entonces
        leer_simbolo();
    sino
        error_sintactico();
    fsi
fprocedimiento

funcion S()
    switch SLA
        case (, a, b, +, -:
            A();
            S();
        case = .
            Reconoce(=);
            A();
            S();
        case ), $ :
            default:
                error_sintactico();
    fswitch
funcion
```

```
funcion A()
    switch SLA
        case (, a, b:
            B();
            A();
        case + :
            Reconoce(+);
            B();
        case - :
            Reconoce(-);
            B();
        case =, ", $ :
            default:
                error_sintactico();
    fswitch
ffuncion

funcion B()
    switch SLA
        case (:
            Reconoce(());
            S();
            Reconoce());
        case a:
            Reconoce(a);
        case b:
            Reconoce(b);
        default:
            error_sintactico();
    fswitch
ffuncion
```