Septiembre-2015.pdf

CarlosGarSil98

Algorítmica y Modelos de Computación

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería
Universidad de Huelva

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 limites 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 BinarialIt(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:

a. (0,5 puntos). Calcular la complejidad del algoritmo **iterativo** propuesto para el caso **peor** mediante el conteo del número de operaciones elementales.
b. (0,5 puntos). Calcular la complejidad del algoritmo **recursivo** para el caso **peor** por el **Teorema maestro**.
c. (0,5 puntos). Calcular la complejidad del algoritmo **recursivo** propuesto para el caso **peor** por el método de la **ecuación característica**.
d. (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

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

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 $n, n/2, n/4, \ldots$ es decir, potencias de 2, hasta $n/n = 1$; $n = 2^x$. El bucle iniciará en $i = 0$ hasta $i = \log_2(n)$

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

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

Apartado b:

Para el caso base, indices 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: $a = 1$, $b = 2$, $k = 0$, $p = 0$

$a > b^k$; $1 > 2^0 \longrightarrow$ No se cumple
$a = b^k$; $1 = 2^0 \longrightarrow$ Sí se cumple

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

Apartado c:

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

$T(n) - T(n/2) = 12$ [cambio de base $n = 2^k$] $T(2^k) - T(2^{k-1}) = 12$ No Homogénea

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

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

$p(x) = (x-1)(x-1)$; Raices: $r1 = 1$ doble

$T(2^k) = C_0 \cdot 1^k \cdot k^0 + C_1 \cdot 1^k \cdot k^1$ [cambio de base $2^k = n$] $k = \text{Log}(n)$ $T(n) = C_0 + C_1 \text{ Log}(n)$

Se deberian de calcular las contantes, 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 ($C_0 = 14$, $C_1 = 12$)

$T(n) = C_0 + C_1 \text{ Log}(n) \longrightarrow C_1 \neq 0 \longrightarrow T(n) \in O(\text{Log}(n))$

Apartado d:

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

iterativo = $13 \cdot \text{Log}(n) + 16$
recursivo = $12 \cdot \text{Log}(n) + 14$

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

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
→ p	{q, s}	{p}	{q, r}
q		{q, r}	{r}
r		{p, s}	{q}
* s	{s}	{q, r, s}	

Se pide:

a. (1 punto). El AFD equivalente.
b. (1 punto). El AFD mínimo.

Apartado a:

→Q_0 = λ-clausura(p) = {p, q, r}
f'(Q_0, a) = {q, s} = λ-clausura({q, s}) = {q, s, r} Q_1 estado final
f'(Q_0, b) = {p, q, r, s} Q_2 Estado final

*Q_1 = {q, s, r}
f'(Q_1, a) = {s} Q_3 Estado final
f'(Q_1, b) = {q, r, s, p} Q_2

*Q_2 = {p, q, r, s}
f'(Q_2, a) = {q, s, r} Q_1
f'(Q_2, b) = {p, q, r, s} Q_2

*Q_3 = {s}
f'(Q_3, a) = {s} Q_3
f'(Q_3, b) = {q, r, s} Q_1

f	a	b
→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_1, a) = C_1 f'(Q_1, b) = C_1
f'(Q_2, a) = C_1 f'(Q_2, b) = C_1
f'(Q_3, a) = C_1 f'(Q_3, b) = C_1

f	a	b
→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 → AS
S → =AS | λ
A → BA
A → AA"A" | λ
A → +B | -B
B → (S) | a | b

A → BA | AA"A" | λ | +B | -B

A → +BA' | -BA' | BAA' | λ
A' → AA"A" A' | λ

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

Apartado a:

S → AS
S → =AS | λ
A → BA
A → AA"A" | λ
A → +B | -B
B → (S) | a | 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 regra necesaria, por tanto, no se trata de una gramática LL(1).

Apartado b:

La tabla se obtiene mediante el siguiente algoritmo:

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


```
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 → AS
A S #	a = b $	A → BA
B A S #	a = b $	B → a
a A S #	a = b $	Leer(a)
A S #	= b $	A → λ
S #	= b $	S → =AS
= A S #	= b $	Leer(=)
A S #	b $	A → BA
B A S #	b $	B → b
b A S #	b $	Leer(b)
A S #	$	A → λ
S #	$	S → λ
#	$	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_simbollo();
    sino
        error_sintactico();
    fsi
fprocedimiento
```

```
funcion S()
    switch SLA
        case (, a, b, t, -:
            A();
            S();
        case = .
            Reconocc(=);
            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_sintáctico();
    fswitch
ffuncion
```

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