Febrero-2016.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 9 de febrero del 2016.**
APELLIDOS, NOMBRE García Silva, Carlos **NOTA**_____

Ejercicio_1. (2 puntos).

➢ Analizar el algoritmo de la Búsqueda del k-ésimo menor elemento. Dado un vector de n elementos, el problema de la selección consiste en buscar el k-ésimo menor elemento.

- Supongamos que disponemos de la siguiente definición de tipo:
  CONST n = ...;
  TYPE vector = ARRAY [1..N] OF INTEGER;
  Y supongamos que primero y último indican los límites del array (inicialmente primero=1 y último=n)

- Para la solución del problema utilizamos la idea del agoritmo **Partition** (utilizado en Quicksort): El vector A[p...r] se particiona(reorganiza) en dos subvectores A[p..q] y A[q+1..r] de forma que los elementos de A[p..q] son menores o iguales que el pivote(por ej: primer elemento) y los de A[q+1..r] mayores o iguales.

```
int función Partition (A:vector; primero, ultimo: int)
    piv = A[ primero ]; i = primero-1; j = ultimo+1;
    mientras j >= i hacer
            mientras A[ j ] >= piv hacer
                    j = j – 1;
            fmientras
            mientras A[ ji] <= piv hacer
                    i = i +1 1;
            fmientras
            si i < j entonces    /* A[ i ] <=> A[ j ] */
                    temp = A[ j ]; A[ j ] = A[ i ]; A[ i ] = temp;
            fsi
    fmientras
    devuelve j;          /* retorna el índice para la división (partición) */
ffuncion Partition
```

➢ El algoritmo de la Búsqueda del k-ésimo menor elemento puede ser implementado:

1. Versión **iterativa** de la **Búsqueda del k-ésimo menor elemento** puede ser implementado:

```
funcion SelectIt(A : vector, primero, ultimo, k : entero)
    mientras (primero < ultimo) hacer
            q = Partition(A, primero, ultimo)
            si (k ≤ q) entonces
                    ultimo = q
            sino
                    primero = q + 1
            fsi
    fmientras
    devuelve A[primero]
ffuncion
```

2. Versión **recursiva** de la **Búsqueda del k-ésimo menor elemento**

```
funcion SelectRc(A : vector, primero, ultimo, k : entero)
    si (primero == ultimo) entonces
            devuelve A[primero]
    fsi
    q = Partition(A, primero, ultimo)
    i = q – primero + 1
    si (k ≤ i) entonces
            devuelve SelectRc(A, primero, q, k)
    sino
            devuelve SelectRc(A, q+1, ultimo, k-i)
    fsi
ffuncion
```

➢ **Se pide:**

a. (0,5 puntos). Calcular la complejidad del algoritmo **iterativo** propuesto para el caso **promedio** mediante el conteo del número de operaciones elementales.

b. (0,5 puntos). Calcular la complejidad del algoritmo **recursivo** propuesto para el caso **promedio** por el método de la **ecuación característica**.

c. (0,5 puntos). Calcular la complejidad del algoritmo **recursivo** propuesto para el caso **promedio** por el **Teorema maestro**.

d. (0,5 puntos). Comprobar si ambas versiones, iterativa y recursiva, invierten el mismo tiempo.

Apartado a:

$$T(n) = 1 + 1 + \sum_{i=1}^{?} (1 + 1 + 2 + T_{part} + 4) + 2$$

$$T_{part}(n) = 4 + 1 + 1 + \sum_{i=1}^{1} \left(2 + 1 + \sum_{j=1}^{n/2} (2 + 2 + 1) + 2 + 1 \right.$$

$$\left. + \sum_{j=1}^{n/2} (2 + 2 + 1) + 1 + 1 + 7 \right) + 1$$

$$T_{part}(n) = 7 + \left(15 + \sum_{j=1}^{n/2} (5) + \sum_{j=1}^{n/2} (5) \right) = 5 \cdot n + 22 \in O(n)$$

En cada iteración el tamaño será la mitad: n/2, n/4,... potencia de dos, por tanto cuando llegue al final de i; $n/2^i = 1$; $i = \log(n)$

$$T(n) = 4 + \sum_{i=1}^{\log(n)} (30 + 5(n/2^i)) = 4 + \sum_{i=1}^{\log(n)} (30) + 5 \cdot n \cdot \sum_{i=1}^{\log(n)} (1/2^i)$$

$$T(n) = 4 + 30 \cdot \log(n) + 5 \cdot n \cdot \left(\frac{2(2^i - 1)}{2^i} \right) = 4 + 30 \log(n) + 5 \cdot n \left(\frac{2(n-1)}{n} \right)$$

$$T(n) = 30 \log(n) + 10n - 6 \in O(n)$$

Apartado b:

$$T(n) \begin{cases} 3 & \text{si } n = 1 \\ T(n/2) + 5n + 36 & \text{si } n > 1 \end{cases}$$

No Homogénea

$$T(n) - T(n/2) = 5n + 36 \quad \left[\begin{array}{c} \text{cambio de base} \\ n = 2^k \end{array} \right] \quad T(2^k) - T(2^{k-1}) = 5 \cdot 2^k + 36$$

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

$b^k \cdot p(k)^d = 5 \cdot 2^k \cdot k^0$; $b = 2$, $d = 0$ $\longrightarrow (x-2)^{0+1}$

$b^k \cdot p(k)^d = 36 \cdot 1^k \cdot k^0$; $b = 1$, $d = 0$ $\longrightarrow (x-1)^{0+1}$

$p(x) = (x-1)(x-2)(x-1)$; Raíces: $r1 = 1$ doble, $r2 = 2$

$$T(2^k) = c_0 \cdot 1^k \cdot k^0 + c_1 \cdot 1^k \cdot k^1 + c_2 \cdot 2^k \cdot k^0 = c_0 + c_1 \cdot k + c_2 \cdot 2^k$$

$$\left[\begin{array}{c} \text{cambio de base} \\ 2^k = n \end{array} \right] \quad T(n) = c_0 + c_1 \cdot \log(n) + c_2 \cdot n$$

KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa

$$T(n) \begin{cases} 3 & \text{si } n = 1 \\ T(n/2) + 5n + 36 & \text{si } n > 1 \end{cases}$$

$T(1) = 3$
$T(2) = T(1) + 10 + 36 = 3 + 46 = 49$
$T(4) = T(2) + 20 + 36 = 105$
$T(8) = T(4) + 40 + 36 = 181$

$$\begin{cases} C_0 + C_1 \log(2) + 2C_2 = 49 \\ C_0 + C_1 \log(4) + 4C_2 = 105 \\ C_0 + C_1 \log(8) + 8C_2 = 181 \end{cases}$$

$C_0 = -7$, $C_1 = 36$, $C_2 = 10$

$T(n) = -7 + 36 \log(n) + 10n \in O(n)$

Apartado c:

Según el teorema maestro: $T(n) = aT(n/b) + O(n^k \cdot \log^p(n))$
En este caso: $a = 1$, $b = 2$, $k = 1$, $p = 0$

$a > b^k \longrightarrow 1 > 2^1$; No se cumple
$a = b^k \longrightarrow 1 = 2^1$; No se cumple
$a < b^k \longrightarrow 1 < 2^1$; Sí se cumple $\longrightarrow T(n) \in O(n)$

Apartado d:

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

iterativo = $30 \log(n) + 10n$
recursivo = $-7 + 36 \log(n) + 10n$

Los valores del recursivo son mayores, portanto, no invierten el mismo tiempo, pero sí similar y mismo orden de complejidad ($O(n)$)

Ejercicio_2. (3 pts)

- Resolver el problema de la mochila para el caso en que no se permita partir los objetos (es decir, un objeto se coge entero o no se coge nada).
  - Problema de la mochila:
    - Tenemos:
      - **n** objetos, cada uno con un peso (**pi**) y un beneficio (**bi**).
      - Una mochila en la que podemos meter objetos, con una capacidad de peso máximo **M**.
    - Objetivo: llenar la mochila con esos objetos, maximizando la suma de los beneficios (valores) transportados, y respetando la limitación dada por la capacidad máxima M.
    - Se supondrá que los objetos NO se pueden partir en trozos.
- **Se pide:**
  - **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. Comprobar si el algoritmo garantiza la solución óptima en este caso (la demostración se puede hacer con un contraejemplo).
    - Aplicar el algoritmo al caso: **n** = 3, **M** = 6, **p** = (2, 3, 4), **b** = (1, 2, 5).
  - **b.** (1.5 pts). Resolver el problema mediante programación dinámica. Definir la ecuación recurrente, los casos base, las tablas y el algoritmo para rellenarlas y especificar cómo se recompone la solución final a partir de los valores de las tablas.
    - Aplicar el algoritmo al caso: n = 3, M = 6, p = (2, 3, 4), b = (1, 2, 5).
  - **NOTA:** una posible ecuación recurrente es:

$$\text{Mochila}(k, m) = \begin{cases} 0 & \text{Si } \mathbf{k}=0 \text{ ó } \mathbf{m}=0 \\ -\infty & \text{Si } \mathbf{k}<0 \text{ ó } \mathbf{m}<0 \\ \max\{\text{Mochila}(k-1, m),\ b_k + \text{Mochila}(k-1, m-p_k)\} & \end{cases}$$

Universidad de Huelva. Escuela Técnica de Ingeniería. Departamento de Tecnologías de la Información.
ALGORÍTMICA Y MODELOS DE COMPUTACIÓN. 3º Grado Ingeniería Informática. **La Rábida 9 de febrero del 2016.**
APELLIDOS, NOMBRE García Silva, Carlos **NOTA**

Ejercicio_3. (2 pts)

- Dado el AFND = ({a, b, c}, {p, q, r, s, t, u, v}, f, p, {v}) donde f viene dado por la siguiente tabla de trancisiones:

f	a	b	c	λ
→ p				{q, t}
q		{r, s}		{r, s}
r				{q, u}
s	{t, p}		{u}	
t		{v}		{q}
u	{q, s}		{v}	{s}
* v				{r}

➢ **Se pide:**

a. (0,25 puntos). Si son aceptadas o no por el autómata las siguientes cadenas:
1. f'(p,bbcc)
2. f'(p,acbcac)
3. f'(p,bcacaa)
4. f'(p,caa)
5. f'(p,abac)

b. (0,5 puntos). El AFD equivalente.

c. (0,5 puntos). El AFD mínimo.

d. (0,25 puntos). Corroborar el resultado obtenido para las palabras del apartado **a** con el AFD obtenido en el apartado **c**.

e. (0,5 puntos). Obtener una expresión regular equivalente al AFD obtenido en el apartado **c**.

Apartado a:

1 f'(p, bbcc)
f'({p,q,t,r,s,u}, b) = {r,s,v,q,u}
f'({r,s,v,q,u}, b) = {r,s,q,u}
f'({r,s,q,u}, c) = {u,v,q,r,s}
f'({u,v,q,r,s}, c) = {u,v,q,r,s}
v ∈ {u,v,q,r,s} Cadena Aceptada

2. f'(p, acbcac)
f'({p,q,t,r,s,u}, a) = {p,t,s,q,r,u}
f'({p,t,s,q,r,u}, c) = {u,v,s,r,q}
f'({u,v,s,r,q}, b) = {r,s,q,u}
f'({r,s,q,u}, c) = {u,v,s,r,q}
f'({u,v,s,r,q}, a) = {q,s,t,p,r,u}
f'({q,s,t,p,r,u}, c) = {u,v,s,r,q}
v ∈ {u,v,s,r,q} Cadena Aceptada

3. $f'(p, bcacaa)$

$f'(\{p,q,t,r,s,u\}, b) = \{r,s,v,q,u\}$

$f'(\{r,s,v,q,u\}, c) = \{u,v,s,r,q\}$

$f'(\{u,v,s,r,q\}, a) = \{q,s,t,p,r,u\}$

$f'(\{q,s,t,p,r,u\}, c) = \{u,v,s,r,q\}$

$f'(\{u,v,s,r,q\}, a) = \{q,s,t,p,r,u\}$

$f'(\{q,s,t,p,r,u\}, a) = \{t,p,q,s,r,u\}$

$v \notin \{t,p,q,s,r,u\}$ Cadena No Aceptada

4. $f'(p, caa)$

$f'(\{p,q,t,r,s,u\}, c) = \{u,v,s,r,q\}$

$f'(\{u,v,s,r,q\}, a) = \{q,s,t,p,r,u\}$

$f'(\{q,s,t,p,r,u\}, a) = \{t,p,q,s,r,u\}$

$v \notin \{t,p,q,s,r,u\}$ Cadena No Aceptada

5. $f'(p, abac)$

$f'(\{p,q,t,r,s,u\}, a) = \{p,t,s,q,r,u\}$

$f'(\{p,t,s,q,r,u\}, b) = \{v,r,s,q,u\}$

$f'(\{v,r,s,q,u\}, a) = \{t,p,q,s,r,u\}$

$f'(\{t,p,q,s,r,u\}, c) = \{u,v,s,r,q\}$

$v \in \{u,v,s,r,q\}$ Cadena Aceptada

Apartado b:

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

$f'(Q_0, a) = \{t,p,q,s,r,u\}$ Q_0

$f'(Q_0, b) = \{r,s,v,q,u\}$ Q_1 Estado final

$f'(Q_0, c) = \{u,v,s,r,q\}$ Q_1

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

$f'(Q_1, a) = \{t,p,s,r,u\}$ Q_0

$f'(Q_1, b) = \{r,s,q,u\}$ Q_2 Estado Normal

$f'(Q_1, c) = \{u,v,s,r,q\}$ Q_1

$Q_2 = \{r,s,q,u\}$

$f'(Q_2, a) = \{t,p,q,s,r,u\}$ Q_0

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

$f'(Q_2, c) = \{u,v,s,r,q\}$ Q_1

$\rightarrow$

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

Apartado c:

Agrupamos en estados no finales y finales:

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

$f'(Q_0, a) = C_0$ $f'(Q_0, b) = C_1$ $f'(Q_0, c) = C_1$

$f'(Q_2, a) = C_0$ $f'(Q_2, b) = C_0$ $f'(Q_2, c) = C_1$

No coinciden hay que dividir

$Q/E_1 = (C_0 = \{Q_0\}, C_1 = \{Q_1\}, C_2 = \{Q_2\})$

como tenemos un conjunto por cada estado, podemos decir que ya nos encontrábamos ante el AFD mínimo

Apartado d:

1. $f'(Q_0, bbcc)$.
   $f'(Q_0, b) = Q_1$
   $f'(Q_1, b) = Q_2$
   $f'(Q_2, c) = Q_1$
   $f'(Q_1, c) = Q_1$
   Estado final, Aceptada

2. $f'(Q_0, acbcac)$:
   $f'(Q_0, a) = Q_0$
   $f'(Q_0, c) = Q_1$
   $f'(Q_1, b) = Q_2$
   $f'(Q_2, c) = Q_1$
   $f'(Q_1, a) = Q_0$
   $f'(Q_0, c) = Q_1$
   Estado final, Aceptada

3. $f'(Q_0, bcacaa)$:
   $f'(Q_0, b) = Q_1$
   $f'(Q_1, c) = Q_1$
   $f'(Q_1, a) = Q_0$
   $f'(Q_0, c) = Q_1$
   $f'(Q_1, a) = Q_0$
   $f'(Q_0, a) = Q_0$
   Estado No final, Rechazada

4. $f'(Q_0, caa)$:
   $f'(Q_0, c) = Q_1$
   $f'(Q_1, a) = Q_0$
   $f'(Q_0, a) = Q_0$
   Estado No final, Rechazada

5. $f'(Q_0, abac)$.
   $f'(Q_0, a) = Q_0$
   $f'(Q_0, b) = Q_1$
   $f'(Q_1, a) = Q_0$
   $f'(Q_0, c) = Q_1$
   Estado final, Aceptada

Apartado e:

Ecuación característica

$$\begin{cases} X_0 = aX_0 + bX_1 + cX_1 + b + c \\ X_1 = aX_0 + bX_2 + cX_1 + c \\ X_2 = aX_0 + bX_2 + cX_1 + c \end{cases}$$

Utilizaremos el proceso de sustitución

$X_2 = aX_0 + bX_2 + cX_1 + c$; $X_2 = b^*(aX_0 + cX_1 + c)$; $X_2 = b^*aX_0 + b^*cX_1 + b^*c$

$X_1 = aX_0 + bX_2 + cX_1 + c$; $X_1 = aX_0 + b(b^*aX_0 + b^*cX_1 + b^*c) + cX_1 + c$;

$X_1 = aX_0 + bb^*aX_0 + bb^*cX_1 + bb^*c + cX_1 + c$;

$X_1 = (bb^*c + c)^*(aX_0 + bb^*aX_0 + bb^*c + c)$;

$X_1 = (bb^*c + c)^*aX_0 + (bb^*c + c)^*bb^*aX_0 + (bb^*c + c)^*bb^*c + (bb^*c + c)^*c$

$$X_0 = \begin{bmatrix} a + b(bb^*c + c)^*a + b(bb^*c + c)^*bb^*a + c(bb^*c + c)^*a + c(bb^*c + c)^*bb^*a \\ b(bb^*c + c)^*bb^*c + b(bb^*c + c)^*c + c(bb^*c + c)^*bb^*c + c(bb^*c + c)^*c + b + c \end{bmatrix}^*$$

Se debería de seguir simplificando, pero no da tiempo en el examen, mejor asegurar otros puntos.

Ejercicio_4. (3 pts)

- Dada la siguiente gramática:
  S → - S | (S) | AB
  A → i C
  B → - S | λ
  C → (S) | λ

➢ **Se pide:**

a. (0.25 pts). Comprobar si es LL(1) mediante el cálculo de los conjuntos Primero y Siguiente.
b. (0.25 pts). Convertir la gramática del apartado anterior en un autómata con pila que acepte el mismo lenguaje por pila vacía.
c. (0.5 pts). Analizar, teniendo en cuenta el principio de preanálisis (lectura de un símbolo de la entrada con anticipación) la entrada "**i - - i ((i))**" según el AP especificado en el apartado **b** anterior.
d. (0.75 pts). Implementar la tabla de análisis sintáctico y especificar el pseudocódigo de análisis sintáctico tabular.
e. (0.75 pts). Construir la traza correspondiente al reconocimiento de la frase "**i - - i ((i))**" según el pseudocódigo especificado en el apartado d anterior.
f. (0.5 pts). Especificar el pseudocódigo de análisis sintáctico dirigido por la sintaxis para la gramática obtenida LL(1).

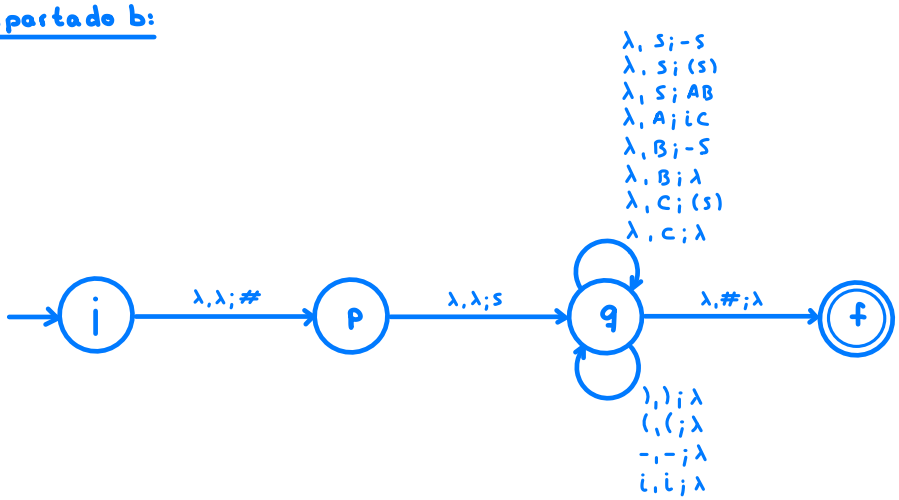
Apartado a.

S → - S
S → (S)
S → A B
A → i C
B → - S | λ
C → (S) | λ

	Primeros	Siguientes	Predicción	
S	-	) $	-	intersección vacía
	(		(	
	i		i	
A	i	- λ	i	
B	-	) $	-	intersección vacía
	λ		) $	
C	(	- λ	(	intersección vacía
	λ		- λ	

Como todas las intersecciones son vacías, podemos decir que nos encontramos con la gramática equivalente LL(1).

Apartado b:



Apartado c:

Estado	Pila	Entrada	Acción	Indetermina	Acción
i	λ	i--i((i))i$	i λ λ ; p #		
p	#	i--i((i))i$	p λ λ ; q S		
q	S #	i--i((i))i$	q λ S ; q AB		S::= AB
q	A B #	i--i((i))i$	q λ A ; q iC		A::= iC
q	i C B #	i--i((i))i$	q i i ; q λ		Reconoce(i)
q	C B #	--i((i))i$	q λ C ; q (S)	q λ C ; q λ	C::= λ
q	B #	--i((i))i$	q λ B ; q -S		B::= -S
q	- S #	--i((i))i$	q - - ; q λ		Reconoce(-)
q	S #	-i((i))i$	q λ S ; q -S		S::= -S
q	- S #	-i((i))i$	q - - ; q λ		Reconoce(-)
q	S #	i((i))i$	q λ S ; q AB		S::= AB
q	A B #	i((i))i$	q λ A ; q iC		A::= iC
q	i C B #	i((i))i$	q i i ; q λ		Reconoce(i)
q	C B #	((i))i$	q λ C ; q (S)		C::= (S)
q	(S) B #	((i))i$	q ((; q λ		Reconoce(()
q	S) B #	(i))i$	q λ S ; q (S)		S::= (S)
q	(S)) B #	(i))i$	q ((; q λ		Reconoce(()
q	S)) B #	i))i$	q λ S ; q AB		S::= AB
q	A B)) B #	i))i$	q λ A ; q iC		A::= iC
q	i C B)) B #	i))i$	q i i ; q λ		Reconoce(i)
q	C B)) B #	))i$	q λ C ; q (S)	q λ C ; q λ	C::= λ
q	B)) B #	))i$	q λ B ; q -S	q λ B ; q λ	B::= λ
q	)) B #	))i$	q)) ; q λ		Reconoce())
q	) B #	)i$	q)) ; q λ		Reconoce())
q	B #	i$	q λ B ; q -S	q λ B ; q λ	B::= λ
q	#	i$	Rechazar		

Apartado d:

La tabla se obtiene mediante el siguiente algoritmo:

```
∀ A → α
    ∀ 'a' terminal != λ ∈ PRIM(α)
        Tabla[A, a] = α
    fin ∀
    si λ ∈ PRIM(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:

Construir traza →

Pila	Entrada	Acción
λ	i--i((i))$	Apilar(#)
#	i--i((i))$	Apilar(S)
S #	i--i((i))$	S::=AB
A B #	i--i((i))$	A::=iC
i C B #	i--i((i))$	Leer(i)
C B #	--i((i))$	C::=λ
B #	--i((i))$	B::=-S
- S #	--i((i))$	Leer(-)
S #	-i((i))$	S::=-S
- S #	-i((i))$	Leer(-)
S #	i((i))$	S::=AB
A B #	i((i))$	A::=iC
i C B #	i((i))$	Leer(i)
C B #	((i))$	C::=(S)
(S) B #	((i))$	Leer(()
S) B #	(i))$	S::=(S)
(S)) B #	(i))$	Leer(()
S)) B #	i))$	S::=AB
A B)) B #	i))$	A::=iC
i C B)) B #	i))$	Leer(i)
C B)) B #	))$	C::=λ
B)) B #	))$	B::=λ
)) B #	))$	Leer())
) B #	))$	Leer())
B #	))$	B::=λ
#	))$	error()

Apartado f:

```
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 -:
            Reconoce(-);
            S();
        case (:
            Reconoce(();
            S();
            Reconoce());
        case i:
            A();
            S();
        default:
            error_sintactico();
    fswitch
ffuncion
```

```
funcion A()
    switch SLA
        case i:
            Reconoce(i);
            C();
        default:
            error_sintactico();
    fswitch
ffuncion
```

```
funcion B()
    switch SLA
        case -:
            Reconoce(-);
            S();
        case ), $:
        default:
            error_sintactico();
    fswitch
ffuncion
```

```
funcion C()
    switch SLA
        case (:
            Reconoce(();
            S();
            Reconoce());
        case -:
    fswitch
ffuncion
```