Febrero2018-Resuelto.pdf

alberto_fm_

Algorítmica y Modelos de Computación

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería
Universidad de Huelva

Febrero 2018

domingo, 6 de febrero de 2022 10:30

Ejercicio_1. (2 puntos)

Dado el esquema del algoritmo de ordenación QuickSort:

```
QuickSort (A, izq, der)  /* Ordena un vector A desde izq hasta der */
    if (izq < der) {
        piv=mediana (izq, der)
        div =partition (A, piv, izq, der)
        /* El vector A[izq..der] se particiona en dos subvectores A[izq..div] y A[div+1..der],
        de forma que los elementos de A[izq..div] son menores o iguales que los de A[div+1..der]
        (según elemento pivote) */
        QuickSort (A, izq, div)
        QuickSort (A, div+1, der)
        }
```

Donde, con "*mediana*" se obtiene la mediana de los elementos del array A entre las posiciones izq y der (el elemento que ocuparía la posición central si estuvieran ordenados), y "partition" es el procedimiento de particionar pero usando *piv* como pivote, con lo que el problema se divide en dos subproblemas de igual tamaño. Si el tiempo de ejecución del procedimiento "*mediana*" es $t_{med}(n)=20n$, y el de "partition" es $t_{par}(n)=n$:

a. (0,75 puntos). Calcular la complejidad del algoritmo propuesto por el método de la **ecuación característica**.

b. (0,25 puntos). Calcular la complejidad del algoritmo propuesto por el Teorema maestro.

c. (0,5 puntos). Calcular la complejidad del algoritmo propuesto por expansión de recurrencia.

d. (0,5 puntos). Si el método de la Burbuja tiene un tiempo de ejecución de n^2, justificar para qué valores de la entrada es preferible esta versión del QuickSort al método de la Burbuja.

El sistema Recurrente es el siguiente:

$$T(n) = \begin{cases} C_1 & \text{si } n \leq 1 \\ 2T(n/2) + 21n + C_2 & \text{si } n > 1 \end{cases}$$

a)

$$T(n) = 2T(n/2) + 21n + C_2 \rightarrow T(n) - 2T(n/2) = 21n + C_2 \rightarrow \text{No Homogénea}$$

$$\left[n = 2^K \leftrightarrow K = \log_2 n \right]$$

$$\rightarrow T(2^K) - 2T(2^{K-1}) = 21 \cdot 2^K + C_2 \cdot 1^K \rightarrow$$

$$\left[T(2^K) = t_K \right]$$

$$\rightarrow t_K - 2t_{K-1} = 21 \cdot 2^K + C_2 \cdot 1^K \rightarrow$$

$$(x-2)(x-2)(x-1) = 0 \rightarrow \text{raíces: } \begin{cases} r_{11} = 2 \\ r_{12} = 2 \\ r_2 = 1 \end{cases}$$

Por tanto, la solución tendrá la forma:

$$t_K = C_3 \cdot 2^K \cdot K^0 + C_4 \cdot 2^K \cdot K + C_5 \cdot 1^K \cdot K^0 =$$

$$= 2^K \cdot C_3 + 2^K \cdot K \cdot C_4 + C_5$$

Deshacemos los cambios

$$T(n) = c_3 \cdot n + c_4 \cdot n \cdot \log_2 n + c_5 .$$

$\boxed{\text{Concluimos que } T(n) \in O(n \log_2 n)}$

b) El teorema maestro es el siguiente:

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

En nuestra ecuación recurrente:

$$\left.\begin{array}{l} a = 2 \\ b = 2 \\ K = 1 \\ p = 0 \end{array}\right\} \quad a = b^k \rightarrow 2 = 2^1 \longrightarrow T(n) \in O(n \cdot \log(n))$$

c) Por expansión de recurrencias:

$$T(n) = 2T(\tfrac{n}{2}) + 21n + c_5$$
$$= 2\left(2T(\tfrac{n}{4}) + 21\tfrac{n}{2} + c_5\right) + 21n + c_5$$
$$= 2^2 T(\tfrac{n}{2^2}) + 21n + 21n + c_5(1+2)$$
$$= 2^2 T(\tfrac{n}{2^2}) + 2 \cdot (21n) + c_5(1+2)$$
$$= 2^2\left(2T(\tfrac{n}{2^3}) + 21\tfrac{n}{2^2} + c_5\right) + 2 \cdot 21n + c_5(1+2) =$$
$$= 2^3 T(\tfrac{n}{2^3}) + 21n \cdot 3 + c_5(1+2+2^2)$$

En general, para i veces...

$$T(n) = 2^i\, T(\tfrac{n}{2^i}) + 21n \cdot i + \sum_{j=0}^{i-1} c_5\,(2^j)$$
$$= 2^i\, T(n/2^i) + 21n \cdot i + c_5 \cdot (2^i - 1)$$

En el caso base $\frac{n}{2^i} = 1 \longrightarrow n = 2^i \rightarrow i = \log_2 n$

Sustituimos...

$$T(n) = n \cdot c_1 + 21n \cdot \log_2 n + nc_5 - c_5 =$$
$$= 21n \cdot \log_2 n + (c_5 + c_1) \cdot n - c_5$$

Por tanto, $\boxed{T(n) \in O(n \log_2 n)}$

$= 21n \cdot \log_2 n + (C_5 + C_1) \cdot n - C_5$

Por tanto, $\boxed{T(n) \in O(n \log_2 n)}$

d) Burbuja $= n^2$; Quicksort $= (21 n \log_2 n + n)$

Debemos hallar el punto de corte de ambas funciones

Quicksort − Burbuja $= 0 \rightarrow$

$(21n \log_2(n) + n) - n^2 = 0$ Podemos simplificar la ec.

$\cancel{n}(21 \log_2(n) + 1) - n \cdot \cancel{n} = 0$

$(21 \log_2(n) + 1) - n = 0$. Para probar números comenzaré utilizando potencias de 2 para simplificar el logaritmo.

$n = 2^8$

$21 \log_2(2^8) + 1 - 2^8 =$

$21 \cdot 8 + 1 - 2^8 = 169 - 256 < 0 \rightarrow$ Quicksort es más eficiente.

Como Quicksort es $n \log n$, cuando $n \rightarrow \infty$ seguirá siendo más eficiente que Burbuja. Por tanto, tenemos una cota superior para nuestro intervalo.

$n = 2^7$

$21 \log_2(2^7) + 1 - 2^7 =$

$21 \cdot 7 + 1 - 128 = 148 - 128 > 0 \rightarrow$ Burbuja es más eficiente.

Hemos encontrado un valor para el que Burbuja es más eficiente. Así que, el valor exacto debe estar en el intervalo

$(128, 256)$

Podemos seguir acotando el intervalo.

$n = 192$

$21 \cdot \log_2(192) + 1 - 192 =$

$160.28.. - 192 < 0 \rightarrow$ Quicksort es más eficiente.

$(128, 192)$

$n = 160$

$21 \cdot \log(160) + 1 - 160 < 0 \rightarrow$ Quicksort más eficiente

$(128, 160)$

$n = 144$

$21 \cdot \log(144) + 1 - 144 > 0 \rightarrow$ Burbuja más eficiente

$(144, 160)$

$n = 152$

$21 \cdot \log(152) + 1 - 152 > 0 \rightarrow$ Burbuja más eficiente

$(152, 160)$

$n = 156$

$21 \cdot \log(156) + 1 - 156 < 0 \rightarrow$ Quicksort más eficiente

$(152, 156)$

$n = 154$

$21 \cdot \log(154) + 1 - 154 < 0 \rightarrow$ Quicksort más eficiente

$(152, 154)$

$n = 153$

$21 \cdot \log(153) + 1 - 153 < 0 \rightarrow$ Quicksort más eficiente

$(152, 153) \rightarrow$ Como ya no hay más números enteros, concluimos que:

- Para valores mayores de 152 es recomendable usar Quicksort
- Para valores menores a 153 es recomendable usar Burbuja.

Ejercicio_2. (3 puntos)

- 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 (p_i) y un valor o beneficio (b_i)
      - 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 de capacidad máxima **M**.
    - Se supondrá que los objetos **NO** se pueden partir en trozos.
- **Se pide**:

a. (1.5 puntos). Diseñar un algoritmo **voraz** para resolver el problema aunque no se garantice la solución óptima. Es necesario marcar en el código propuesto a que 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 puntos). 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:

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

a)

```
algoritmo MochilaVoraz(M: Integer, b, p: array[1..N] of Integer)
    X : array [1..N] of Integer;
    peso = 0;
    ArticulosOrd = ordenarArticulosSegunCriterio(b, p);
      para i=0 hasta N hacer
          si ( peso + p[articulosOrdenados[i]] ≤ M) entonces
              peso ← peso + p[articulosOrdenados[i]];
              X[articulosOrdenados[i]] = 1;
          sino
              X[articulosOrdenados[i]] = 0;
          fsi
     fpara
falgoritmo
```

Criterio: Se han ordenado los artículos descendientemente según el cociente b/p.

- Otro criterio sería ordenarlos por mayor beneficio o menor peso.

Candidatos: Los candidatos serán todos los objetos que su peso sea menor o igual que M.

Seleccionados: Los artículos seleccionados serán aquellos cuyo peso más el peso actual no supere a M.

Solución: La solución será una N-tupla donde cada x_i será un objeto: Si $x_i = 0$ no se ha seleccionado, si $x_i = 1$, sí se ha seleccionado.

El algoritmo no garantiza la solución óptima por ejemplo:

$n = 3$
$M = 6$
$p = \{2, 3, 4\}$
$b = \{1, 7, 6\}$
$b_n/p_n = \{0{,}5;\ 2{,}33;\ 1{,}25\}$

$p = \{2, 4, 3\}$
$b = \{1, 6, 7\}$

En este caso, se elegirían los objetos $X = \{1, 0, 1\}$ dando un beneficio de 7. Mientras que la solución óptima sería

$X=\{1,1,0\}$ dando un beneficio de 8.

Aplicamos el algoritmo al ejemplo:

$n=3$, $M=6$, $p\{2,3,4\}$, $b=\{1,2,5\}$

1º ordenamos los objetos según el criterio:

$b/p=\{0.5; 0,\hat{6}; 1.25\}$ → Ya están ordenados.

Inicialmente:
- peso = 0
- $i=0$

$i=0$

$(0+2)\leq 6$ → VERDAD

$X[0]=1$
peso = 2

$i=1$

$(2+3)\leq 6$ → Verdad

$X[1]=1$
peso = 5

$i=2$

$(5+4)\leq 6$ → FALSO.

$\boxed{\text{Solución: } X\{1,1,0\}}$ → Podemos ver que no es la solución óptima.

b)

```
algoritmo MochilaPD(T: array [0...N][0...M])
    para i=1 hasta N hacer T[i][0] = 0 fpara;
    para j=0 hasta M hacer T[0][j] = 0 fpara;

  para i=1 hasta N hacer
    para j=1 hasta M hacer
        si (j < p[i]) entonces
            T[i][j] = T[i-1][j];
        sino
            T[i][j] = max(T[i-1][j], T[i-1][j-p[i]]+b[i]);
        fsi
    fpara
  fpara
falgoritmo
```

fpara
fpara
falgoritmo

Para recomponer la solución hacemos el siguiente algoritmo:

```
algoritmo Recomponer-Solucion (T: array [1..N][1..M] of Integer)
   j = M
   X: array [1..N] of Integer;
   para i = N hasta 1 hacer
      si ( T[i][j] != T[i-1][j])
         X[i] = 1;
         j = j - p[i];
      sino
         X[i] = 0;
      fsi
   fpara
falgoritmo
```

La ecuación recurrente de este algoritmo es:

$$T(K,m) = \begin{cases} 0 & \text{si } K=0 \text{ ó } m=0 \\ -\infty & \text{si } K<0 \text{ ó } m<0 \\ \max\left(T(K-1,m),\ T(K-1, m-p_K)+b_K\right) & \text{si } K>0,\ m>0 \end{cases}$$

} CASOS BASE

Aplicamos el algoritmo:

$n=3$; $M=6$; $p(2,3,4)$, $b=(1,2,5)$

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1
2	0	0	1	2	2	3	3
3	0	0	1	2	5	5	6

Recomponer la solución:

$\left.\begin{array}{l} i=3 \\ j=6 \end{array}\right\}$

$T[i,j] \,!= T[i-1, j] \rightarrow 6 \,!= 3$

$X[3] = 1$

$j = 6-4 = 2$

$T[i,j] \,!= T[i-1, j] \rightarrow 1 == 1$

$X[2] = 0$

$T[i,j] \,!= T[i-1, j] \rightarrow 1 \,!= 0$

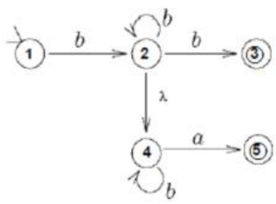
$X[1] = 1$

$j = 2-2 = 0$

Solución: $X\{1, 0, 1\} \rightarrow$ óptima.

Ejercicio_3. (2 puntos)

- Dado el AFND definido en el grafo:



➤ **Se pide**:

a. (0.25 puntos). Si son aceptadas o no por el autómata las siguientes cadenas:

1. f'(1, ***ba***)
2. f'(1,***ab***)
3. f'(1,***bb***)
4. f'(1,***b***)
5. f'(1,***bba***)

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.**

a) El autómata en forma de tabla:

	a	b	λ
→1	∅	{2,4}	∅
2	∅	{2,3,4}	{4}
*3	∅	∅	∅
4	{5}	{4}	∅
*5	∅	∅	∅

a1) $f'(1, ba) = f'(CL(1), ba) =$

$f'(\{1\}, b) = \{2,4\}$

$f'(\{2,4\}, a) = \{5\} \cap F = \{5\} \neq \emptyset \rightarrow$ ACEPTADA

a2) $f'(1, ab) = f'(CL(1), ab) =$

$f'(\{1\}, a) = \emptyset$

$f'(\emptyset, b) = \emptyset \cap F = \emptyset \rightarrow$ RECHAZADA

a3) $f'(1, bb) = f'(CL(1), bb) =$

$f'(\{1\}, b) = \{2,4\}$

$f'(\{2,4\}, b) = \{2,3,4\} \cap F = \{3\} \neq \emptyset \rightarrow$ ACEPTADA

a4) $f'(1, b) = f'(CL(1), b) =$

$f'(\{2,4\}, b) = \{2,3,4\} \cap F = \{3\} \neq \emptyset \to$ ACEPTADA

a4) $f'(1, b) = f'(CL(1), b) =$

$f'(\{1\}, b) = \{2,4\} \cap F = \emptyset \to$ RECHAZADA

a5) $f'(1, bba) = f'(CL(1), b) =$

$f'(\{1\}, b) = \{2,4\}$

$f(\{2,4\}, b) = \{3,2,4\}$

$f'(\{3,2,4\}, a) = \{5\} \cap F = \{5\} \neq \emptyset \to$ ACEPTADA

b) El AFD equivalente será:

	a	b
→ Q_0	Q_1	Q_2
Q_1	Q_1	Q_1
Q_2	Q_3	Q_4
* Q_3	Q_1	Q_1
* Q_4	Q_3	Q_4

$Q_0 = CL(1) = \{1\}$

$f'(Q_0, a) = \emptyset = Q_1$

$f'(Q_0, b) = \{2,4\} = Q_2$

$f'(Q_2, a) = \{5\} = Q_3$

$f'(Q_2, b) = \{2,4,3\} = Q_4$

c) Minimizamos el AFD anterior mediante el algoritmo conjunto/cociente.

$E/Q_0 = (c_0 = \{Q_0, Q_1, Q_2\},\ c_1 = \{Q_3, Q_4\})$

Dividimos el conjunto $\left\{ \begin{array}{l} f'(Q_0, a) = c_0 \ ;\ f'(Q_1, a) = c_0 \ ;\ f'(Q_2, a) = c_1 \\ f'(Q_0, b) = c_0 \ ;\ f'(Q_1, a) = c_0 \ ;\ f'(Q_2, b) = c_1 \end{array} \right.$

$E/Q_1 = (c_0 = \{Q_0, Q_1\},\ c_1 = \{Q_3, Q_4\},\ c_2 = \{Q_2\})$

Dividimos el cjto. $\left\{ \begin{array}{l} f'(Q_0, a) = c_1 \ ;\ f'(Q_1, a) = c_0 \\ f'(Q_0, b) = c_2 \ ;\ f'(Q_1, b) = c_0 \end{array} \right.$

$E/Q_2 = (c_0 = \{Q_0\},\ c_1 = \{Q_3, Q_4\},\ c_2 = \{Q_2\},\ c_3 = \{Q_1\})$

$\left. \begin{array}{l} f(Q_3, a) = c_3 \ ;\ f'(Q_4) = c_1 \\ f'(Q_3, b) = c_3 \ ;\ f'(Q_4) = c_1 \end{array} \right\}$ Dividimos

$E/Q_3 = (c_0 = \{Q_0\},\ c_1 = \{Q_3\},\ c_2 = \{Q_2\},\ c_3 = \{Q_1\},\ c_4 = \{Q_4\})$

Podemos comprobar que existe un cjto para cada estado. Por tanto, ya tenemos el AFD mínimo.

	a	b
→ c_0	c_3	c_2

Puede definirse por la quíntupla:

	a	b
→ c_0	c_3	c_2
* c_1	c_3	c_3
c_2	c_1	c_4
c_3	c_3	c_3
* c_4	c_1	c_4

Puede definirse por la quíntupla:

$AF = (\{c_0, c_1, c_2, c_3, c_4\}, \{a, b\}, c_0, f, \{c_1, c_4\})$

donde f son las transiciones:

$$f = \begin{cases} f(c_0, a) = c_3 \ ; \ f(c_0, b) = c_2 \ ; \\ f(c_1, a) = c_3 \ ; \ f(c_1, b) = c_3 \ ; \\ f(c_2, a) = c_1 \ ; \ f(c_2, b) = c_4 \ ; \\ f(c_3, a) = c_3 \ ; \ f(c_3, b) = c_3 \ ; \\ f(c_4, a) = c_1 \ ; \ f(c_4, b) = c_4 \ ; \end{cases}$$

d)

$f'(c_0, ba) =$

$f(c_0, b) = c_2$

$f(c_2, a) = c_1 \in F \rightarrow$ ACEPTADA

$f'(c_0, ab) =$

$f(c_0, a) = c_3$

$f(c_3, b) = c_3 \notin F \rightarrow$ RECHAZADA

$f'(c_0, bb) =$

$f(c_0, b) = c_2$

$f(c_2, b) = c_4 \in F \rightarrow$ ACEPTADA

$f'(c_0, b) =$

$f'(c_0, b) = c_2 \notin F \rightarrow$ RECHAZADA

$f'(c_0, bba) =$

$f'(c_0, b) = c_2$

$f'(c_2, b) = c_4$

$f'(c_4, a) = c_1 \in F \rightarrow$ ACEPTADA

e) El sistema recurrente será:

	a	b
→ c_0	c_3	c_2
* c_1	c_3	c_3
c_2	c_1	c_4
c_3	c_3	c_3
* c_4	c_1	c_4

c_3 = Muerto.

$c_3 = \emptyset$

$$\begin{cases} x_0 = b x_2 \\ x_1 = \lambda \\ x_2 = a x_1 + b x_4 + b \\ x_4 = a x_1 + b x_4 + a + b \end{cases}$$

$x_4 = b x_4 + (a x_1 + a + b) =$

$= b x_4 + (a\lambda + a + b) =$

$$= bX_4 + (a+b) \quad \Leftrightarrow$$

$$\boxed{X_4 = b^*(a+b)}$$

$$X_2 = a\lambda + b(b^*(a+b)) + b \rightarrow$$

$$X_2 = a + bb^*(a+b)) + b \rightarrow$$

$$\boxed{X_2 = a + b^*(a+b) + b}$$

$$X_0 = b(a + b^*(a+b) + b) \rightarrow$$

$$X_0 = ba + bb^*(a+b) + bb \rightarrow$$

$$X_0 = ba + b^* \cdot (a+b) + bb \rightarrow$$

$$\boxed{X_0 = ba + b^*a + b^* + bb}$$

Ejercicio_4. (3 puntos)

Considérese la siguiente gramática:

S → (L)
 | a
L → L % S
 | S

a. (0,25 puntos). Comprobar si es LL(1) mediante el cálculo de los conjuntos Primero y Siguiente.

b. (0.25 puntos). Con la gramática equivalente LL(1), especificar un autómata con pila que acepte el mismo lenguaje por pila vacía.

c. (0.5 puntos). Analizar por el autómata del apartado **b.** anterior, teniendo en cuenta el principio de preanálisis (lectura de un símbolo de la entrada con anticipación) la entrada "**(a%(a%a))**".

d. (0,75 puntos) Con la gramática equivalente LL(1), construir la tabla de análisis LL(1) y especificar el pseudocódigo de análisis sintáctico tabular.

e. (0,75 puntos) Construir la traza correspondiente al reconocimiento de la frase: "**(a%(a%a))**" según el pseudocódigo especificado en el apartado **d.** anterior.

f. (0,5 puntos) Especificar el pseudocódigo de análisis sintáctico dirigido por la sintaxis para la gramática obtenida LL(1).

a) Para que una gramática sea LL(1), debemos eliminar la recursividad a izquierdas:

$$\left.\begin{array}{l} S \rightarrow (L) \mid \\ \quad a \\ L \rightarrow L\%S \mid \\ \quad S \end{array}\right\} \begin{array}{l} 1.\ S \rightarrow (L) \mid \\ 2.\ \quad\quad a \\ 3.\ L \rightarrow SL' \\ 4.\ L' \rightarrow \%SL' \mid \lambda \end{array}$$

Ahora que tenemos una gramática no recursiva a izquierdas, podemos comprobar la condición suficiente y necesaria para ver si es LL(1):

Una gramática será LL(1) si para cada par de producciones con el mismo antecedente, la intersección de sus símbolos directores es vacía. Esto es:

$$PRIM((L)) \cap PRIM(a) = \{(\} \cap \{a\} = \varnothing$$

PRIM (%.SL') ∩ SIG(L') = /* Calcular */

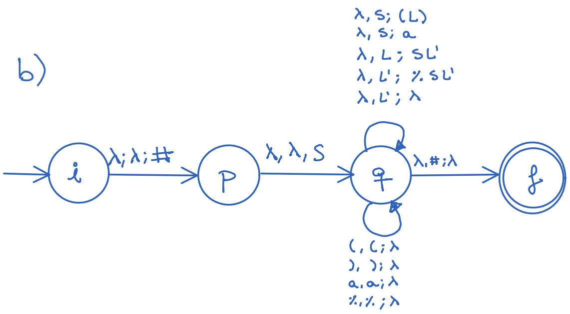
Calculemos los conjuntos PRIMERO y SIGUIENTE:

	PRIMERO	SIGUIENTE
S	{ (, a }	{ %,), $ }
L	{ (, a }	{), }
L'	{ %, λ }	{) }

Entonces: PRIM (%.SL') ∩ SIG(L') = { % } ∩ {) } = ∅

Ambas intersecciones son ∅, por lo que estamos ante una gramática LL(1).

b)



c)

ESTADO	PILA	ENTRADA	ACCIÓN	INDET.	ACCIÓN
i	λ	(a%(a%a))$	(i, λ, λ; p, #)		
p	#	(a%.(a%.a))$	(p, λ, λ; q, S)		
q	S#	(a%.(a%.a))$	(q, λ, S; q, (L))		S → (L)
q	(L)#	(a%.(a%.a))$	(q, (, (; q; λ)		RECONOCER('(');
q	L)#	a%.(a%.a))$	(q, λ, L; q, SL')		L → SL'
q	SL')#	a%.(a%.a))$	(q, λ, S; q, a)		S → a
q	aL')#	a%.(a%.a))$	(q, a, a; q, λ)		RECONOCER('a');
q	L')#	%(a%.a))$	(q, λ, L'; q, %.SL')		L' → %.SL'
q	%.SL')#	%(a%.a))$	(q, %., %.; q, λ)		RECONOCER('%');
q	SL')#	(a%.a))$	(q, λ, S; q, (L))		S → (L)
q	(L)L')#	(a%.a))$	(q, (, (; q, λ)		RECONOCER('(');
q	L)L')#	a%.a))$	(q, λ, L; q, SL')		L → SL'
q	SL')L')#	a%.a))$	(q, λ, S; q, a)		S → a
q	aL')L')#	a%.a))$	(q, a, a; q, λ)		RECONOCER('a')
q	L')L')#	%.a))$	(q, λ, L'; q, %.SL')		L' → %.SL'
q	%.SL')L')#	%.a))$	(q, %., %.; q, λ)		RECONOCER('%')
q	SL')L')#	a))$	(q, λ, S; q, a)		S → a
q	aL')L')#	a))$	(q, a, a; q, λ)		RECONOCER('a');
q	L')L')#	))$	(q, λ, L'; q, λ)	(q, λ, L'; q, λ)	L' → λ
q	)L')#	))$	(q, (, (; q, λ)		RECONOCER('(')
q	L')#	)$	(q, λ, L'; q, λ	(q, λ, L'; q, λ)	L' → λ
q	)#	)$	(q,),); q, λ)		RECONOCER(')');
q	#	$	(q, λ, #; f, λ)		
f	λ	λ	ACEPTADA		

d) El algoritmo para construir la tabla:

```
∀ A → α
    ∀ 'a' terminal != λ ∈ PRIM(α)
        T[A][a] = α
    f∀
    ∀ 'b' terminal != λ ∈ SIG(A)
        T[A][a] = λ
    f∀
f∀
```

El algoritmo para el análisis sintáctico Tabular:

```
algoritmo AnalisisTabular()
    Apilar(#);
    Apilar(S);
    leer_simbolo();

  mientras (NOT pila_vacia) hacer
      switch cima_pila of
          terminal:
              si (simbolo == cima_pila);
                  Desapilar();
                  leer(simbolo)
              sino
                  error_sintáctico();
              fsi
          break;
          no terminal:
              si (Tabla[cima][simbolo] != error)
                  Desapilar();
                  leer();
                  Apilar(Tabla[cima][simbolo]);
              sino
                  error_sintactico();
              fsi
          break;
      fswitch
  fmientras

  si (cima != '#') entonces
      error_sintáctico();
  sino
      Escribir('ACEPTADA');
  fsi
falgoritmo.
```

e)

PILA	ENTRADA	ACCIÓN
λ	(a%(a%.a)) $	Apilar(#)
#	(a% (a%.a)) $	Apilar(S);
S#	(a%.(a%.a)) $	Desapilar(S); Apilar((L))
(L)#	(a%.(a%.a)) $	Desapilar('(') ; leer ('(');
L)#	a%.(a%.a)) $	Desapilar(L); Apilar(SL');
SL')#	a%(a%.a)) $	Desapilar(S); Apilar('a');
aL')#	a%.(a%.a)) $	Desapilar(a); Leer(a)
L')#	%.(a%.a)) $	Desapilar(L') ;Apilar(%.SL')
%.SL')#	%.(a%.a)) $	Desapilar(%.); leer(%.)
SL')#	(a%.a)) $	Desapilar(S); Apilar((L));
(L)L')#	(a%.a)) $	Desapilar((); Leer('(');
L)L')#	a%.a)) $	Desapilar(L); Apilar(SL')
SL')L')#	a%.a)) $	Desapilar(S); Apilar(a)
aL')L')#	a%.a)) $	Desapilar(a); leer(a)
L')L')#	%.a)) $	Desapilar(L'); Apilar(%.SL')
%SL')L')#	%.a)) $	Desapilar(%.); Leer(%.)
SL')L')#	a)) $	Desapilar(S); Apilar(a)
aL')L')#	a)) $	Desapilar(a); leer(a)
L')L')#	)) $	Desapilar(L')
)L')#	)) $	Desapilar('('); Leer('(')
L')#	) $	Desapilar(L');
)#	) $	Desapilar(')'); leer(')');
#	$	cima == #
#	$	ACEPTADA

f)

```
Program   ProgramaPrincipal()
      SLA = leer_simbolo();
        S();

      si  SLA != '$'  entonces

          error_sintáctico();

      fsi
fprogram
```

```
procedure Reconocer(simbolo: s)

      si (SLA == s) entonces

          leer_simbolo();
      sino
          error_sintáctico();
      fsi
fprocedure
```

```
funcion  S()

   switch  SLA  of

      case '(' :
                Reconocer('(');
                L();
                Reconocer(')');

      case 'a' :
                Reconocer('a');

      default: error_sintáctico();
   fswitch
ffuncion
```

```
funcion  L()
   switch  SLA  of

      case '(', 'a' :
                 S();
                 L'();
      default: error_sintáctico();

   fswitch
ffuncion
```

```
funcion  L'()

   switch  SLA  of

      case '%.' :
                 Reconocer('%.');
                 S();
```

```
        case '%' :
                    Reconocer('%');
                    S();
                    L'();
        case ')' : /* nada */

        default :  error_sintáctico()

    }switch
}funcion
```