

Septiembre2020-Resuelto.pdf



alberto_fm_



Algorítmica y Modelos de Computación



3º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingeniería
Universidad de Huelva

NEW

WUOLAH Print

Lo que faltaba en Wuolah



Imprimir



Ejercicio_1. (2 puntos)

Se tiene un vector de enteros no repetidos y ordenados de menor a mayor. Diseñar un algoritmo que compruebe en tiempo logarítmico si existe algún elemento del vector que coincida con su índice. Se pide:

- (0,5 puntos). Elección del esquema (voraz, vuelta atrás, divide y vencerás). Identificación del problema con el esquema.
- (1 punto). Estructura de datos. Algoritmo completo (con pseudocódigo). Aplicar a $v = \{-3, -2, 0, 2, 3, 5, 7, 9, 12\}$
- (0,5 puntos). Estudio de coste.

a) Optare por elegir un esquema divide y vencerás.

Dividir: el vector en dos partes iguales. Como está ordenado, podemos comprobar si la posición ($n/2$) es menor, igual o mayor.

- Si es igual, hemos encontrado la solución
- Si es menor, repetiremos el algoritmo con la parte izquierda.
- Si es mayor, con la parte derecha.

Nos basamos en el algoritmo de búsqueda binaria recursiva

$O(\log n)$

b)

```

funcion Ejercicio1 (v:vector, p, u: entero)
    si p > u entonces
        devolver -1;
    fin
    mitad <- ⌊(p+u)/2⌋
    si (mitad > v[mitad])
        devolver Ejercicio1 (v, mitad+1, u);
    sino si (mitad < v[mitad])
        devolver Ejercicio1 (v, p, mitad-1);
    sino
        devolver mitad;
    fin
fin
  
```

Aplicamos a $v = \{-3, -2, 0, 2, 3, 5, 7, 9, 12\}$

Ejercicio1 (-3, -2, 0, 2, 3, 5, 7, 9, 12, 0, 8)

$0 > 8? \rightarrow \text{False}$

mitad <- 4

$4 > v[4] \rightarrow \text{True}$

Ejercicio1 (-3, -2, 0, 2, 3, 5, 7, 9, 12, 5, 8)

$4 > V[4] \rightarrow \text{True}$

Ejercicio1(3, 5, 6, 7, 9, 12, 4, 5, 8)

$5 > 8 ? \rightarrow \text{NO}$

mitad $\leftarrow 6$

$6 > V[6] \rightarrow \text{NO}$
 $6 < V[6] \rightarrow \text{SI}$

Ejercicio1(3, 5, 4, 5, 5);

$5 > 5 ? \rightarrow \text{NO}$

mitad $\leftarrow 5$

$5 > V[5] \rightarrow \text{NO}$
 $5 < V[5] \rightarrow \text{NO}$

Devuelve 5

c) El coste del algoritmo vendrá dado por el siguiente sistema recurrente:

$$T(n) = \begin{cases} 2 & \text{si } n=0 \\ T\left(\frac{n}{2}\right) + 10 & \text{si } n>0 \end{cases}$$

Por teorema maestro:

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

$a=1$
 $b=2$
 $k=0$
 $p=0$

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

$$n=2^k \Leftrightarrow k=\log n$$

$$T(2^k)=t_k$$

$$T(n) = T\left(\frac{n}{2}\right) + 10 \rightarrow T(n) - T\left(\frac{n}{2}\right) = 10$$

$$\rightarrow T(2^k) - T(2^{k-1}) = 10 \rightarrow$$

$$\rightarrow t_k - t_{k-1} = 10 \rightarrow \text{No Homogénea}$$

$$\rightarrow (x-1)(x-1) = 0 \rightarrow r: 1 \text{ (doble)}$$

$$t_k = c_{11} \cdot K^0 \cdot 1^K + c_{12} \cdot K \cdot 1^K = c_{11} \cdot 1 + c_{12} \cdot \log n \cdot 1 =$$

$$= c_{11} + c_{12} \cdot \log n \rightarrow T(n) \in O(\log_2 n)$$

Por expansión de recurrencias

$$T(n) = T\left(\frac{n}{2}\right) + 10$$
$$= (T\left(\frac{n}{4}\right) + 10) + 10$$

NEW

WUOLAH Print

Lo que faltaba en Wuolah



Imprimir



- Todos los apuntes que necesitas están aquí
- Al mejor precio del mercado, desde **2 cent.**
- Recoge los apuntes en tu copistería más cercana o recíbelos en tu casa
- Todas las anteriores son correctas



$$\begin{aligned}
 T(n) &= T\left(\frac{n}{2}\right) + 10 \\
 &= (T\left(\frac{n}{4}\right) + 10) + 10 \\
 &= (T\left(\frac{n}{8}\right) + 10) + 10 \\
 &\dots
 \end{aligned}$$

En general:

$$\begin{aligned}
 T(n) &= T\left(\frac{n}{2^i}\right) + 10 \cdot i \\
 \text{Para el caso base: } \frac{n}{2^i} = 1 \rightarrow n = 2^i \rightarrow i = \log_2 n
 \end{aligned}$$

$$T(n) = 10 \cdot \log_2 n \in O(\log_2 n)$$

*El ejercicio solo pide calcular el coste. No especifica qué método voy a usar. Yo he hecho todos para comprobar que el coste es el mismo, pero en el examen habría elegido llevado por teorema maestro (es el más sencillo y rápido)

Ejercicio 2. (3 puntos)

- Resolver el **problema de la mochila** para el caso en que **no se permite partir los objetos** (es decir, un objeto se coge entero o no se coge nada).
 - Problema de mochila: Una mochila en la que podemos meter objetos, con una capacidad de peso máximo **M**; tenemos **n** objetos, cada uno con un peso (**p_i**) y un valor o beneficio (**b_i**). El **Objetivo** es 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:**
 - (1 punto). 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)**
 - (1 punto). Resolver el problema mediante **programación dinámica**. Definir la ecuación recurrente, los casos base, las tablas. Aplicar el algoritmo al caso: **n=3, M=6, p=(2,3,4), b=(1,2,5)**
 - (1 punto). Resolver el problema **por backtracking** usando el esquema iterativo. Indicar cómo es la representación de la solución, la forma del árbol y programar las funciones genéricas del esquema correspondiente. Aplicar el algoritmo al caso: **n=3, M=6, p=(2,3,4), b=(1,2,5)**

a)

```

función MochilaVoraz( b[1..3] , p[1..3], M )
    ordenar Artículos(); // Seguir el criterio de b/p
    X = 10, 0, 0;
    peso = 0;
    i = 1;

    mientras (i ≤ n AND peso < M)
        si (peso + p[i]) ≤ M hacer
            X[i] ← 1;
            peso ← peso + p[i];
        fin
        i ← i + 1;
    fmientras

    return X;
  
```

CRITERIO: Los artículos estarán ordenados en función del mayor cociente beneficio/peso

CRITERIO: Los artículos estarán ordenados en función del mayor cociente beneficio/peso.

CANDIDATOS: Los candidatos serán todos los artículos

SOLUCIÓN: Vector X (1 : se coge el objeto; 0 : no se coge el objeto;)

Aplicamos el algoritmo al ejemplo:

$$\left(\frac{1}{2}, \frac{2}{3}, \frac{5}{4} \right) \rightarrow 0.5, 1.5, 1.25$$

1º Reordenamos los artículos en función de su cociente b/p de forma decreciente

$$b(1, 5, 2); p(2, 4, 3)$$

$$\begin{aligned} i &= 1 \\ p[1] + \text{peso} &= 2 \leq 6 \\ x[1] &\leftarrow 1 \\ &x = \langle 1, 0, 0 \rangle; \\ &\text{peso} = 2; \\ &\text{beneficio} = 1; \end{aligned}$$

$$\begin{aligned} i &= 2 \\ p[2] + \text{peso} &= 6 \leq 6 \\ x[2] &\leftarrow 1 \\ &x = \langle 1, 1, 0 \rangle; \\ &\text{peso} = 6; \\ &\text{beneficio} = 6; \end{aligned}$$

Este algoritmo no garantiza la solución óptima para este tipo de problema.

Véamnos el siguiente contraejemplo:

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

Los artículos se encuentran ordenados según el criterio seleccionado:

$$b/p = \{0.5, 0.67, 1.67\}$$

$$\begin{aligned} i &= 1 \\ p[1] + \text{peso} &= 2 \leq 6 \\ x[1] &\leftarrow 1 \\ \text{peso} &\leftarrow \text{peso} + p[1]; \\ &x = \langle 1, 0, 0 \rangle \\ &\text{peso} = 2 \end{aligned}$$

$$\begin{aligned} i &= 2 \\ p[2] + \text{peso} &= 5 \leq 6 \\ x[2] &\leftarrow 1; \\ \text{peso} &\leftarrow \text{peso} + p[2]; \\ &x = \langle 1, 1, 0 \rangle \\ &\text{peso} = 5 \end{aligned}$$

$$i = 3 \quad p[3] + \text{peso} = 8 \not\leq 6$$

(en este ejemplo, la solución sería $(1+2)=3$. Sin embargo existe otra solución factible y mejor $(2+5)=7$. Por tanto, concluimos que el algoritmo no proporciona la solución óptima.

solución factible y mejor $(2+5)=7$. Por tanto, concluimos que el algoritmo no proporciona la solución óptima.

b)

función Mochila PD ($p, b: [1..3]$ of Integer, M : Integer)

devuelve $V(3, M)$;
ffunción

función $V(n, M)$: Integer

para $i=1$ hasta n hacer $V[i][0] = 0$; fpara; } CASOS BASE
para $j=0$ hasta M hacer $V[0][j] = 0$; fpara;

para $i=1$ hasta n hacer

para $j=1$ hasta M hacer

si $j < p[i]$ entonces

$$V[i][j] = V[i-1][j]$$

sino

si $V[i-1][j] \geq V[i-1][j-p[i]] + b[i]$ entonces

$$V[i][j] = V[i-1][j];$$

sino

$$V[i][j] = V[i-1][j-p[i]] + b[i];$$

fsi

fpara

fpara

ffunción

$$\left. \begin{array}{l} \\ \\ \\ \\ \\ \end{array} \right\} \max(V[i-1][j], V[i-1][j-p[i]] + b[i]);$$

	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

Si $j < p[i] \Rightarrow$

$$V[i][j] = \text{el de arriba}.$$

sino

$$V[i][j] = \max(\text{el de arriba o}$$

el de arriba $p[i]$ posiciones atrás)

La ecuación recursiva será:

$$T(K, m) = \begin{cases} 0 & \text{si } K=0 \quad \text{o} \quad m=0 \\ -\infty & \text{si } K < 0 \quad \text{o} \quad m < 0 \\ \max(T(K-1, m), T(K-1, m-p_K) + b_K) & \text{otro caso} \end{cases}$$

c)

```
funcion ModularBT( solución [1..3] : vector of Integer)
    nivel = 1;
    S = solución // solución inicial
    Voa = -∞; // valor óptimo actual
    Soa = ∅; // solución óptima actual
    pact = 0; // peso actual
    bact = 0; // beneficio actual

    repetir
        GENERAR(nivel, s);
        si ( EsSolución(s) AND bact > Voa ) entonces
            Voa = bact;
            Soa = S;
        sino si ( CRITERIO(nivel, s) ) entonces
            nivel++;
        sino
            MIENTRAS ( NOT HAYMAHERMANOS(nivel, s) AND
                        nivel > 0 ) hacer
                RETROCEDER(nivel, s);
            fMIENTRAS
        fsi
    hasta nivel == 0
}funcion
```

```
procedure GENERAR(nivel: Integer, s: array [1..3] of Integer)
    s[nivel] := s[nivel] + 1;
    pact := pact + p[nivel] * s[nivel];
    bact := bact + b[nivel] * s[nivel];
}procedure
```

```
procedure EsSolución(s: array [1..3] of Integer)
    devolver ((nivel == n) AND (pact ≤ M));
}procedure
```

```

procedure CRITERIO (nivel: Integer, s: array [1..3] of Integer)
    devolver ((nivel < n) AND (pact ≤ M));
fprocedure

```

```

procedure HAYMAMERMANOS (nivel: Integer, s: array [1..3] of Integer)
    devolver (s[nivel] < 1)
fprocedure

```

```

procedure RETROCEDER (nivel: Integer, s: array [1..3] of Integer)
    pact := pact - p[nivel] * s[nivel];
    bact := bact - b[nivel] * s[nivel];
    s[nivel] := -1;
    nivel := nivel - 1;
fprocedure

```

Aplicamos el algoritmo al ejemplo:

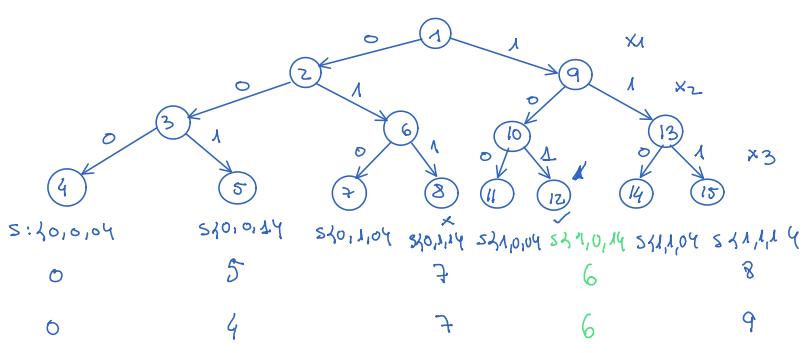
$n=3$

$M=6$

$p = (2, 3, 4)$

$b = (1, 2, 5)$

Usaré un árbol binario para representar la solución:



Ejercicio 3. (2 puntos)

Una caja fuerte dispone de una única cerradura. Para abrirla es necesario hacer girar en ella tres llaves diferentes (denominadas 1, 2 y 3), en un orden predeterminado, que se describe a continuación:

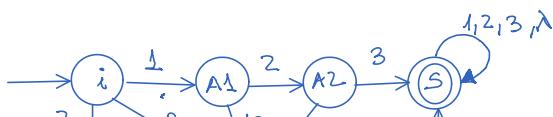
- Llave 1, seguida de llave 2, seguida de llave 3, o bien
- Llave 2, seguida de llave 1, seguida de llave 3.

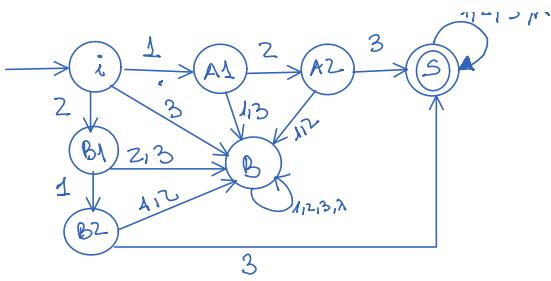
Si no se respeta este orden, la caja se bloquea, y es imposible su apertura; por ejemplo, si se hace girar la llave 1, se retira la misma, se introduce de nuevo y se hace girar. Una vez abierta la caja fuerte, la introducción de las llaves en su cerradura, en cualquier orden, no afecta al mecanismo de cierre (permanece abierta).

Considérese que las denominaciones de las llaves son símbolos del alfabeto, sobre el que define el lenguaje L cuyas palabras son las secuencias permitidas de apertura de la caja fuerte. Por ejemplo, 12323 es una palabra del referido lenguaje.

Se pide:

- (1 punto). Diseño del autómata AFND que acepta el lenguaje L.
- (0,5 puntos). Gramática que genera las palabras de L.
- (0,5 puntos). Autómata AFD mínimo equivalente del apartado a.





a) El AFND que acepta el lenguaje L será:

b) La gramática G que genera los palabras de L vendrá dada por la cuádrupla:

$$G(Q, \Sigma, P, q_0) \Rightarrow G(L) = (Q, \{A_1, A_2, B_1, B_2, S, B_4, 1, 2, 3, 4\}, P, i);$$

donde P viene dado por las siguientes producciones:

$i ::= 1A_1 + 2B_1$
$A_1 ::= 2A_2$
$A_2 ::= 3S + 3$
$B_1 ::= 1B_2$
$B_2 ::= 3S + 3$

	1	2	3	λ
i	A_1	B_1	B	
A_1	B	A_2	B	
A_2	B	B	S	
B_1	B_2	B	B	
B_2	B	B	S	
$*S$	S	S	S	S
B	B	B	B	B

c) Primero transformaré el AFND en AFD:

	1	2	3
Q_0	Q_1	Q_2	M
Q_1	M	Q_3	M
Q_2	Q_4	M	M
Q_3	M	M	Q_5^*
Q_4	M	M	Q_5^*
Q_5^*	Q_5^*	Q_5^*	Q_5^*
M	M	M	M

$$\begin{aligned} Q_0 &= CL(i) = \{1, 2\} \\ LC(f(Q_0, 1)) &= \{A_1\} = Q_1 \\ LC(f(Q_0, 2)) &= \{B_1\} = Q_2 \\ LC(f(Q_0, 3)) &= \{B\} = M \\ LC(f(Q_1, 1)) &= f(Q_1, 3) = M \\ LC(f(Q_1, 2)) &= \{A_2\} = Q_3 \\ LC(f(Q_2, 1)) &= \{B_2\} = Q_4 \\ LC(f(Q_2, 2)) &= f(Q_2, 3) = M \\ LC(f(Q_3, 1)) &= f(Q_3, 2) = M \\ LC(f(Q_3, 3)) &= \{S\} = Q_5^* \\ LC(f(Q_4, 1)) &= f(Q_4, 2) = M \\ LC(f(Q_4, 3)) &= Q_5 \\ LC(f(Q_5, 1)) &= f(Q_5, 2) = f(Q_5, 3) \end{aligned}$$

Ahora minimizaré el autómata que hemos generado. Usaremos el algoritmo de conjunto cociente.

Ahora minimizaremos el autómata que hemos generado. Usaremos el algoritmo de conjunto cociente.

$$Q/E_0 = \{ C_1 = \{ Q_0, Q_1, Q_2, Q_3, Q_4 \}, C_2 = \{ Q_5 \}, C_3 = \{ M \} \}$$

Comprobemos si todos los estados de C_1 son indistinguibles:

$f(Q_0, 1) = Q_1 \in C_1$	$f(Q_1, 1) = M \in C_3$
$f(Q_0, 2) = Q_2 \in C_1$	$f(Q_2, 2) = Q_3 \in C_1$
$f(Q_0, 3) = M \in C_3$	$f(Q_3, 3) = M \in C_3$
$f(Q_2, 1) = Q_4 \in C_1$	$f(Q_3, 4) = M \in C_3$
$f(Q_2, 2) = M \in C_3$	$f(Q_3, 2) = M \in C_3$
$f(Q_2, 3) = M \in C_3$	$f(Q_3, 3) = Q_5 \in C_2$
$f(Q_4, 1) = M \in C_3$	Monto en el mismo conjunto los estados que son indistinguibles. Y creo un nuevo conjunto.
$f(Q_4, 2) = M \in C_3$	
$f(Q_4, 3) = Q_5 \in C_2$	

$$E/Q_1 = (C_1 = \{ Q_0, Q_1, Q_2 \}, C_2 = \{ Q_5 \}, C_3 = \{ M \}, C_4 = \{ Q_3, Q_4 \})$$

$$\begin{aligned} f(Q_0, 1) &= c_1 & f(Q_0, 2) &= c_1 & f(Q_0, 3) &= c_3 \\ f(Q_1, 1) &= c_3 & f(Q_1, 2) &= c_4 & f(Q_1, 3) &= c_3 \\ f(Q_2, 1) &= c_4 & f(Q_2, 2) &= c_3 & f(Q_2, 3) &= c_3 \end{aligned} \quad \left. \begin{array}{l} \text{No coinciden, por lo que hay que crear un nuevo conj.} \\ \text{y monto en el mismo conjunto.} \end{array} \right\}$$

$$\begin{aligned} f(Q_3, 1) &= c_3 & f(Q_3, 2) &= c_3 & f(Q_3, 3) &= c_2 \\ f(Q_4, 1) &= c_3 & f(Q_4, 2) &= c_3 & f(Q_4, 3) &= c_2 \end{aligned} \quad \left. \begin{array}{l} \text{Coinciden, por lo que los monto en el mismo conjunto.} \\ \text{y monto en el mismo conjunto.} \end{array} \right\}$$

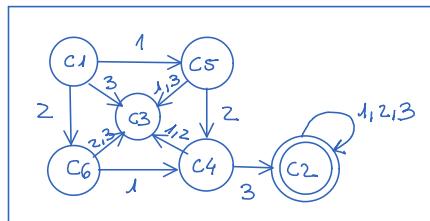
$$E/Q_2 = (C_1 = \{ Q_0 \}, C_2 = \{ Q_5 \}, C_3 = \{ M \}, C_4 = \{ Q_3, Q_4 \}, C_5 = \{ Q_1, Q_2 \})$$

$$\begin{aligned} f(Q_1, 1) &= c_3 & f(Q_1, 2) &= c_4 & f(Q_1, 3) &= c_3 \\ f(Q_2, 1) &= c_4 & f(Q_2, 2) &= c_3 & f(Q_2, 3) &= c_3 \end{aligned}$$

$$E/Q_3 = (C_1 = \{ Q_0 \}, C_2 = \{ Q_5 \}, C_3 = \{ M \}, C_4 = \{ Q_3, Q_4 \}, C_5 = \{ Q_1 \}, C_6 = \{ Q_2 \})$$

El AFD mínimo equivalente será:

	1	2	3
1	C ₅	C ₆	C ₃
2	C ₂	C ₂	C ₂
3	C ₃	C ₃	C ₃
4	C ₃	C ₃	C ₂
5	C ₃	C ₄	C ₃
6	C ₄	C ₃	C ₃



Ejercicio_4. (3 puntos)

Considérese la siguiente gramática:

$$\begin{aligned} S &\rightarrow S = A \mid A \\ A &\rightarrow A + B \mid A - B \mid B \\ B &\rightarrow (S) \mid a \mid b \end{aligned}$$

- (0,5 puntos). Eliminar la recursividad por la izquierda y comprobar si es LL(1).
- (0,5 puntos). Con la gramática equivalente LL(1), especificar un autómata con pila que acepte el mismo lenguaje por pila vacía. Analizar por el autómata anterior, teniendo en cuenta el principio de preanálisis (lectura de un símbolo de la entrada con anticipación) la entrada "a=b".
- (1 punto). Con la gramática equivalente LL(1), construir la tabla de análisis LL(1) y especificar el pseudocódigo de análisis sintáctico tabular. Construir la traza correspondiente al reconocimiento de la frase: "a-b" según el pseudocódigo especificado.
- (1 punto). Especificar el pseudocódigo de análisis sintáctico dirigido por la sintaxis para la gramática obtenida LL(1) y realizar el análisis para la entrada "a=b".

a) Primero eliminamos la recursividad a izquierdo:

$$\begin{aligned} S &\rightarrow S = A \mid \left. \begin{array}{l} A \\ A \rightarrow A + B \mid \left. \begin{array}{l} A - B \mid \left. \begin{array}{l} B \\ B \rightarrow (S) \mid \left. \begin{array}{l} a \mid b \end{array} \right. \end{array} \right. \end{array} \right. \end{array} \right\} S \rightarrow AS' \\ &\quad S' \rightarrow = AS' \mid \lambda \end{aligned}$$

$$\begin{aligned} A &\rightarrow BA' \\ A' &\rightarrow +BA' \mid -BA' \mid \lambda \\ B &\rightarrow (S) \mid \left. \begin{array}{l} a \mid b \end{array} \right. \end{aligned}$$

Para comprobar que una gramática es LL(1), la intersección de sus símbolos directores debe ser vacía. Calcularemos los conjuntos PRIMERO y SIGUIENTE de gramática:

	PRIMERO	SIGUIENTE
S	{c, a, b}	{), \$}
S'	{=, λ }	{), \$}
A	{c, a, b}	{=,), #}
A'	{+, -, λ }	{c, a, b}
B	{c, a, b}	{c, a, b}

1.	$S \rightarrow AS'$
2.	$S' \rightarrow =AS'$
3.	$=$
4.	$A \rightarrow BA'$
5.	$A' \rightarrow +BA'$
6.	$-BA'$
7.	λ
8.	$B \rightarrow (S)$
9.	a
10.	b

$$1) PRIM(=AS') \cap SIG(S') = \{=\} \cap \{\}, \$ \cap \emptyset = \emptyset$$

$$2) PRIM(+BA') \cap PRIM(-BA') \cap SIG(A') =$$

$$\{+\} \cap \{-\} \cap \{c, a, b\} = \emptyset$$

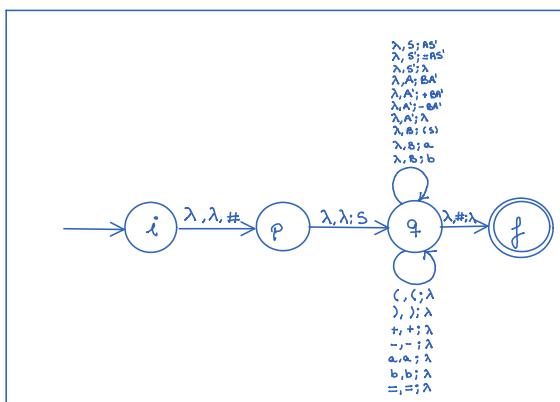
$$3) PRIM((S)) \cap PRIM(a) \cap PRIM(b) = \{(\} \cap \{a\} \cap \{b\} = \emptyset$$

Como las intersecciones son vacías significa que la gramática es LL(1).

b) Un autómata de pila viene dado por la siguiente:

$$AP = (\Sigma, \Gamma, Q, q_0, A_0, f, F)$$

Generaremos un AP que acepte la cadena cuando la pila esté vacía:



ESTADO	PILA	ENTRADA	ACCIÓN	INDETERMINACIÓN	ACCIÓN
i	λ	$a = b \$$	$(i, \lambda, \lambda; \#, p)$		
p	$\#$	$a = b \$$	$(p, \lambda, \lambda; S, q)$		
q	$S\#$	$a = b \$$	$(q, \lambda, S; AS', q)$		$S \rightarrow AS'$
q	$AS'\#$	$a = b \$$	$(q, \lambda, A; BA', q)$		$A \rightarrow BA'$
q	$BA'S'\#$	$a = b \$$	$(q, \lambda, B; a, q)$	$B \xrightarrow{S} (S)$ $B \xrightarrow{a} a \leftarrow \text{Ejecución}$ $B \xrightarrow{b} b$	$B \rightarrow a$
q	$aA's'\#$	$a = b \$$	(q, a, a, λ, q)		RECONOCER('a');
q	$A's'\#$	$= b \$$	$(q, \lambda, A', \lambda, q)$	$A' \xrightarrow{a} BA'$ $A' \xrightarrow{=} B'A'$ $A' \xrightarrow{\lambda} \lambda \leftarrow \text{Ejecución}$	$A' \rightarrow \lambda$
q	$S'\#$	$= b \$$	$(q, \lambda, S'; AS', q)$		$S' \rightarrow AS'$
q	$= AS'\#$	$= b \$$	$(q, \lambda, =; \lambda, q)$		RECONOCER('=');
q	$AS'\#$	$b \$$	$(q, \lambda, A; BA', q)$		$A \rightarrow BA'$
q	$BA'S'\#$	$b \$$	$(q, \lambda, B; b, q)$	$B \xrightarrow{S} (S)$ $B \xrightarrow{b} b \leftarrow \text{Ejecución}$	$B \rightarrow b$
q	$bAS'\#$	$b \$$	(q, b, b, λ, q)		RECONOCER('b');
q	$A's'\#$	$\$$	$(q, \lambda, A'; \lambda, q)$	$A' \xrightarrow{b} BA'$ $A' \xrightarrow{\lambda} \lambda$ $A' \xrightarrow{A} A \leftarrow \text{Ejecución}$	$A' \rightarrow \lambda$
q	$S'\#$	$\$$	$(q, \lambda, S'; \lambda, q)$	$S' \xrightarrow{b} AS'$ $S' \xrightarrow{\lambda} \lambda \leftarrow \text{Ejecución}$	$S' \rightarrow \lambda$
q	$\#$	$\$$	$(q, \lambda, \#; \lambda, f)$		
f	λ	λ	ACEPTAR		

c) La tabla se obtiene mediante el siguiente algoritmo.

```

para todo  $A \rightarrow \alpha$ 
  para todo terminal  $t = \lambda \in PRIM(\alpha)$ 
    Tabla[A][a] =  $\alpha;$ 
  fin para
  si  $\lambda \in PRIM(\alpha)$ 
    para todo terminal  $t = \lambda \in S(b(\alpha))$ 
      Tabla[A][a] =  $\lambda;$ 
    fin para
  fin si
fin para
  
```

```

procedure ANALISIS_MENUAL()
  Apilar( $\#$ );
  Apilar( $S$ );
  Leer(Símbolo);
  while (NOT Pila_Vacia()) hacer
    . .
  
```

```

www.vanillavm.com

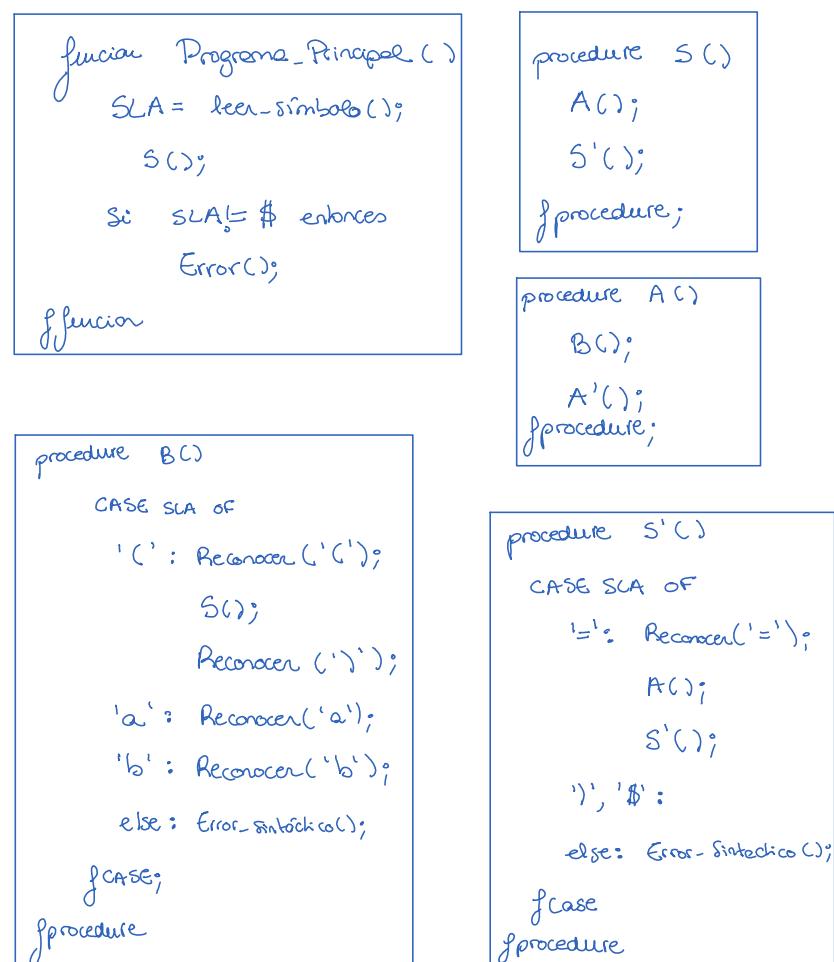
while(NOT Pila-Vacia()) hacer
    switch cima-pila
        case terminal :
            Si cima-pila == simbolo →
                Desapilar(simbolo);
                Leer(simbolo);
            Sino
                error-sintactico();
            Fin
        case NO-terminal :
            Si Tabla[cima-pila][simbolo] != error →
                Desapilar(cima-pila);
                Apilar(Tabla[cima-pila][simbolo]);
            Sino
                error-sintactico();
            Fin
    fswitch
fwhile
    Si cima == # →
        Desapilar(#);
        Escribir(ACEPTADA);
    Sino
        error-sintactico();
    Fin
fprocedure.

```

La traza correspondiente al reconocimiento de $a - b$ será:

Pila	Entrada	Acción
λ	a - b \$	Añadir (#);
#	a - b \$	Añadir (\$);
S#	a - b \$	$S \Rightarrow AS'$
AS#	a - b \$	$A \Rightarrow BA'$
BA'S#	a - b \$	$B \Rightarrow a$
aA'S#	a - b \$	Leer (a);
A'S#	- b \$	$A' \Rightarrow -BA'$
-BA'S#	- b \$	Leer ('-');
BA'S#	b \$	$B \Rightarrow b$
bA'S#	b \$	leer (b),
A'S#	\$	Desapilar (A');
S#	\$	Desapilar (S');
#	\$	Desapilar ('#')
λ	\$	ACEPTAR

c) Pseudocódigo del análisis sintáctico dirigido por la la sintaxis:



```

procedure Reconocer( terminal )
    si (terminal == $A) entonces
        leer_simbolo();
    sino
        error_sintactico();
    fini
fin procedure

```

```

procedure A'()
CASE SCA OF
    '+' : Reconocer('+');
    BC();
    A'();
    '-' : Reconocer('-');
    BC();
    A'();
    '=' , '$' :
else : Error_Sintactico();
fin case
fin procedure

```

Realizamos el análisis de "a=b" utilizando el pseudocódigo anterior:

```

Programa_Principal()
SLA = a ;
S(); → A(); → B(); → Reconocer('a'); →
→ A'(); → /* nada */ → S'(); → Reconocer('='); →
→ A(); → B(); → Reconocer('b'); → A'(); → /* nada */
→ S'(); → /* nada */ → (SLA == $) → Reconocida .

```

paso	SLA	Acción	
1	a	S();	
2	a	A();	
3	a	B();	
4	a	Reconocer('a');	
5	=	A'();	
6	=		
7	=	S'();	
8	=	Reconocer('=');	
9	b	A();	
10	b	B();	
11	b	Reconocer('b');	
12	\$	A'();	
13	\$		
14	\$	S'();	
15	\$		
16	\$	(SLA == \$)	RECONOCIDA