

Febrero2021-Resuelto.pdf



alberto_fm_



Algorítmica y Modelos de Computación



3º Grado en Ingeniería Informática



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

Ejercicio_1. (2 puntos)

Tenemos que ejecutar un conjunto de n tareas, cada una de las cuales requiere un tiempo unitario. En un instante $T=1, 2, \dots$ podemos ejecutar únicamente una tarea. La tarea i produce unos beneficios b_i ($b_i > 0$) sólo en el caso en el que sea ejecutada en un instante anterior o igual a d_i .

Diseñar un algoritmo **voraz** para resolver el problema, aunque no se garantice la solución óptima que nos permita seleccionar el conjunto de tareas a realizar de forma que nos aseguremos que tenemos el **mayor beneficio posible**.

- Detallar :

1. **(1,5 puntos)** Las estructuras y/o variables necesarias para representar la información del problema y el método voraz utilizado (El procedimiento o función que implemente el algoritmo). Es necesario marcar en el seudocódigo propuesto a que corresponde cada parte en el esquema general de un algoritmo voraz. Si hay más de un criterio posible elegir uno razonadamente y discutir los otros. Indicar razonadamente el orden de dicho algoritmo.

2. **(0,5 puntos)** Aplicar el algoritmo implementado en el apartado anterior a la siguiente instancia:

i	1	2	3	4
b_i	50	10	15	30
d_i	2	1	2	1

1.1) Utilizaremos el esquema del algoritmo de selección de actividades,

```

algoritmo SelecActividades( b, d : array [1..N] of Integer )
    instante := 1;
    X = { 0, 0, 0, .., 0 } ;
    i-ordenado = ordenarPor(criterio(i, d, b));
    j ← 1;
    beneficio ← 0;
    mientras ( j ≤ N )
        si ( d[i-ordenado[j]] ≥ instante )
            X[i-ordenado[j]] = instante;
            beneficio = beneficio + b[i-ordenado[j]];
            instante++;
        finsi
        j++;
    finmientras
    devolver X;
falgortimo

```

Las partes del algoritmo voraz son:

CRITERIO: Ordenar el conjunto de actividades en función del valor de $d[i]$ de menor a mayor. En caso de empate elegir la actividad con mayor $b[i]$;

Otro criterio posible sería:

- Elegir la actividad con mayor beneficio

primero.

Sin embargo, este criterio no proporciona una solución óptima.

SOLUCIÓN: La forma de la solución será una N-tupla donde \emptyset significa que el elemento no se selecciona y $n \neq 0$ significa que el elemento se selecciona en el instante n .

CANDIDATOS INICIALES: Todas las actividades propuestas

Cálculo de costes:

$$T(n) = T_{ordenar} + T_{initializar} + T_{bucle-while} + 1 =$$

$$T(n) = n \log(n) + 4 + n + n + 1 =$$

$$T(n) = n \log(n) + 2n + 5 \rightarrow T(n) \in O(n \log n)$$

El algoritmo toma el orden del algoritmo de ordenación que he utilizado, ya que es el que mejor orden tiene de todo el código.

b) Aplicamos el algoritmo al ejemplo:

i	1	2	3	4
b _i	50	10	15	30
d _i	2	1	2	1

ORDENAR POR CRITERIO (i, b, d):

i	4	2	1	3
b _i	30	10	50	15
d _i	1	1	2	2

Instante = i_1

$j = 1$:

beneficio = 0;

$j \leq 4 \rightarrow \text{True}$

$d[1] = 1 \geq \text{instante} \rightarrow \text{True}$

$x[4] = 1; // x = [0, 0, 0, 1]$

beneficio = 30;

Instante = i_2

$j = 2$

beneficio = 30;

$j \leq 4 \rightarrow \text{True}$

$d[2] = 2 \geq \text{instante} \rightarrow \text{Falso}$

Instante = 2

$j = 3$

beneficio = 30

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



$j \leq 4 \rightarrow \text{True}$
 $d[3] = 2 \geq \text{Instante} \rightarrow \text{True}$
 $X[+] = 2; // X = \{2, 0, 0, 14\}$
 beneficio = 80

Instante = 3

$j=4$

beneficio = 80

$j \leq 4 \rightarrow \text{True}$

$d[4] = 2 \geq \text{Instante} \rightarrow \text{False}$

Instante = 3

$j=5$

beneficio = 80

$j \leq 4 \rightarrow \text{False}$

Daruelve $X = \{2, 0, 0, 14\}$

2)

Ejercicio_2. (1 punto)

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)
    /*El vector A[izq..der] se partitiona 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 $t_{part}(n)=n$:

1. (0.25 pto.) Calcular la complejidad del algoritmo propuesto por el método de la **ecuación característica**.
2. (0.25 pto.) Calcular la complejidad del algoritmo propuesto por el Teorema maestro.
3. (0.25 pto.) Calcular la complejidad del algoritmo propuesto por expansión de recursividad.
4. (0.25 pto.) 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.

NOTAS:

- Teorema: La solución a la ecuación $T(n) = aT(n/b) + \Theta(n^k \log^p n)$, con $a \geq 1, b > 1$ y $p \geq 0$, es:

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

- Suma de los valores de la progresión geométrica

$$\sum_{i=0}^n 2^i = 2^{n+1} - 1$$

El sistema recurrente del algoritmo es el siguiente:

$$T(n) = \begin{cases} c_1 & n \leq 1 \\ 2on + n + 2T(n/2) + c_2 & n > 1 \end{cases} \quad \rightarrow$$

$$T(n) = \begin{cases} c_1 & n \leq 1 \\ 2T(n/2) + 2in + c_2 & n > 1 \end{cases}$$

1) Por ecuación característica:

$$T(n) = 2T(n/2) + 2n + C_2 \rightarrow T(n) - 2T(n/2) = 2n + C_2 \text{ (NO HOMOGENEA)}$$

$$\left[n = 2^k \leftrightarrow k = \log_2(n) \right]$$

$$T(2^k) - 2T(2^{k-1}) = 2 \cdot 2^k + C_2 \cdot 1^k \rightarrow \left[T(2^k) = t_k \right]$$

$$t_k - 2t_{k-1} = 2 \cdot 2^k + 1^k \cdot C_2 \rightarrow$$

$$(x-2)(x-2)(x-1) = 0 \rightarrow \begin{cases} r_1 = 2 & (\text{doble}) \\ r_2 = 1 & \end{cases}$$

Por tanto, la complejidad del algoritmo es:

$$\begin{aligned} t_k &= C_{11} \cdot K^0 \cdot 2^k + C_{12} \cdot K \cdot 2^k + C_3 \cdot 1^k \cdot K^0 = \\ &= 2^k C_{11} + 2^k \cdot K \cdot C_{12} + C_3 = (\text{Deshecemos los cambios}) \end{aligned}$$

$$T(n) = \log_2(n) \cdot C_{11} + n \log_2(n) C_{12} + C_3$$

Por tanto, concluimos que $T(n) \in O(n \log_2(n))$

b) Por teorema maestro:

El teorema maestro es el siguiente:

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

En el algoritmo propuesto, los valores son:

$$\left. \begin{array}{l} a = 2 \\ b = 2 \\ k = 1 \\ p = 0 \end{array} \right\} a = b^k \rightarrow 2 = 2^1 \rightarrow \boxed{T(n) \in O(n \cdot \log_2(n))}$$

c) Por expansión de recurrencias:

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + 2n + C_2 = \\ &= 2\left(2T\left(\frac{n}{4}\right) + 2 \cdot \frac{n}{2} + C_2\right) + 2n + C_2 \\ &= 2^2 T\left(\frac{n}{2^2}\right) + 2 \cdot 2n + 2C_2 + C_2 = 2^2 T\left(\frac{n}{2^2}\right) + 2n + (1+2)C_2 \\ &= 2^2 \left(2T\left(\frac{n}{2^3}\right) + 2 \cdot \frac{n}{2^2} + C_2\right) + 2n + (1+2)C_2 \\ &= 2^3 T\left(\frac{n}{2^3}\right) + 2n + 2 \cdot 2n + (1+2+2^2)C_2 \\ &= 2^3 T\left(\frac{n}{2^3}\right) + 3 \cdot 2n + (2^0 + 2^1 + 2^2)C_2 \end{aligned}$$

En general, podemos ver como se repite el patrón para el elemento i:

$$\vdots \quad n \quad \overset{i-1}{\underbrace{\qquad\qquad\qquad}} \quad .$$

En general, podemos ver como se repite el patrón para el elemento i :

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + i \cdot 2in + C_2 \cdot \sum_{j=0}^{i-1} 2^j =$$

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + i \cdot 2in + C_2 (2^i - 1)$$

Calculamos el caso base: $\left(\frac{n}{2^i} = 1\right) \rightarrow [n = 2^i \Leftrightarrow i = \log_2(n)]$
entonces...

$$T(n) = n \cdot C_1 + \log_2(n) \cdot 2in + nc_2 - C_2$$

$$T(n) = 2in \log_2(n) + (C_2 + C_1)n - C_2$$

Concluimos que $T(n) \in O(n \log_2 n)$

4) Para saber el valor n partir del cual podemos garantizar que Quicksort sea más eficiente que el algoritmo Burbuja tenemos que calcular el punto de corte de ambas funciones.

$$(2in \cdot \log_2(n) + n) - (n^2) = 0$$

- Si el resultado < 0 significa que el algoritmo quicksort es más eficiente.
- Por el contrario, si es > 0 , el algoritmo Burbuja será más eficiente.

Podemos simplificar la función para ver mejor el resultado:

$$(2in \cdot \log_2(n) + n) - n \cdot n = 0 \rightarrow$$

$$n(2i \cdot \log_2(n) + 1) - n^2 = 0$$

$$(2i \cdot \log_2(n) + 1) - n = 0$$

Comencemos probando valores arbitrarios para establecer un intervalo.

$$\begin{aligned} n = 256 &\rightarrow (2i \cdot \log_2(256) + 1) - 256 = \\ &= (2i \cdot 8 + 1) - 256 = \\ &= 169 - 256 < 0 \rightarrow \text{Es más eficiente Quicksort} \end{aligned}$$

Como sabemos que Quicksort es más eficiente que el algoritmo Burbuja para valores grandes, tenemos que probar con valores más pequeños que el que hemos elegido. (lo sabemos pq $O(n \log n) < O(n^2)$)

Es decir, tenemos un límite superior del intervalo ($?$, 256)

$$\begin{aligned} n = 128 &\rightarrow (2i \cdot \log_2(128) + 1) - 128 = \\ &= (2i \cdot 7 + 1) - 128 = \\ &= 149 - 128 > 0 \rightarrow \text{Es más eficiente Burbuja.} \end{aligned}$$

$$= 149 - 128 > 0 \rightarrow \text{Es más eficiente Burbuja.}$$

Como hemos encontrado un valor para el que burbuja es más eficiente, tenemos un intervalo donde buscan: $(128, 256)$

Iremos buscando el número escogiendo el número que se encuentre en la mitad del intervalo. De este modo iremos reduciendo a la mitad el espacio de búsqueda:

$$n = (128 + 256)/2 \rightarrow n = 192$$

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

$$160,28 - 192 < 0 \rightarrow \text{Es más eficiente Quicksort.}$$

- Actualizamos el intervalo: $(128, 192)$

$$n = (128 + 192)/2 \rightarrow n = 160$$

$$(21 \cdot \log_2(160) + 1) - 160 =$$

$$154,76 - 160 < 0 \rightarrow \text{Es más eficiente Quicksort.}$$

- Actualizamos el intervalo: $(128, 160)$

$$n = (128 + 160)/2 \rightarrow n = 144$$

$$(21 \cdot \log_2(144) + 1) - 144 =$$

$$151,56 - 144 > 0 \rightarrow \text{Burbuja más eficiente.}$$

- Actualizamos el intervalo: $(144, 160)$

$$n = (144 + 160)/2 \rightarrow n = 152$$

$$(21 \cdot \log_2(152) + 1) - 152 =$$

$$153,2 - 152 > 0 \rightarrow \text{Burbuja más eficiente.}$$

- Actualizamos el intervalo: $(152, 160)$

$$n = (152 + 160)/2 \rightarrow n = 156$$

$$(21 \cdot \log_2(156) + 1) - 156 =$$

$$153,99 - 156 < 0 \rightarrow \text{Quicksort más eficiente}$$

- Actualizamos el intervalo: $(152, 156)$

$$n = (152 + 156)/2 \rightarrow n = 154$$

$$(21 \cdot \log_2(154) + 1) - 154 =$$

$$(153,6) - 154 < 0 \rightarrow \text{Quicksort más eficiente}$$

- Actualizamos el intervalo: $(152, 154)$

$$n = 153$$

$$(21 \cdot \log_2(153) + 1) - 153 = 153,4 - 153 > 0 \rightarrow \text{Burbuja más eficiente}$$

Actualizamos el intervalo: $(153, 154)$

Como ya no existen números enteros entre 153 y 154, podemos concluir que para valores mayores o iguales que 154, usaremos Quicksort. mientras que, para valores menores a 154, usaremos el algoritmo de Burbuja.

* Si en el examen no dejaron usar calculadora, yo lo dejaría en el intervalo más pequeño que fuera posible calcular "directamente" (una potencia de 2) (128, 256) y le escribiría en el examen que se podría reducir el intervalo de búsqueda hasta encontrar el número exacto, pero que, para calcularlo es necesario más tiempo o una calculadora.*

Ejercicio_3. (2 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 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:**
 1. (1 punto) Resolver el problema mediante **programación dinámica**. Definir la ecuación recurrente, los casos base, las tablas y el algoritmo para rellenarlas y especificar cómo se **recomponen** la **solución** final a partir de los valores de las tablas.
 - Aplicar el algoritmo al caso: $n=3, M=5, p=(1, 1, 4), b=(2, 3, 6)$
 2. (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: $n=3, M=5, p=(1, 1, 4), b=(2, 3, 6)$

a)

La ecuación recurrente del algoritmo de la mochila sin freccionamientos es:

$$A(M, K) = \begin{cases} 0 & \text{si } K=0 \text{ ó } m=0 \\ -\infty & \text{si } K \leq 0 \text{ ó } m < 0 \\ \max(A(m, K-1), A(m, K-p_k) + b_k) & \text{otro caso} \end{cases}$$

La tabla será $A(i, j)$ donde

$$\begin{aligned} i &= N \quad (\text{nº de artículos}) \\ j &= M \quad (\text{peso máximo de la mochila}) \end{aligned}$$

El algoritmo para rellenar la tabla es el siguiente:

```
algoritmo( A[1..N][1..M], b, p : array [1..N] of Integer, M: Integer )
```

```
para i=1 hasta N hacer A[i, 0]=0;
```

```
para j=0 hasta M hacer A[0, j]=0;
```

```
para i=1 hasta N hacer
```

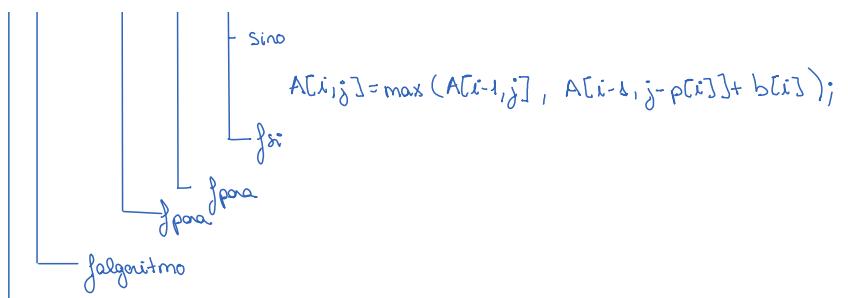
```
  para j=1 hacer hacer
```

```
    si ( j < p[i] ) hacer
```

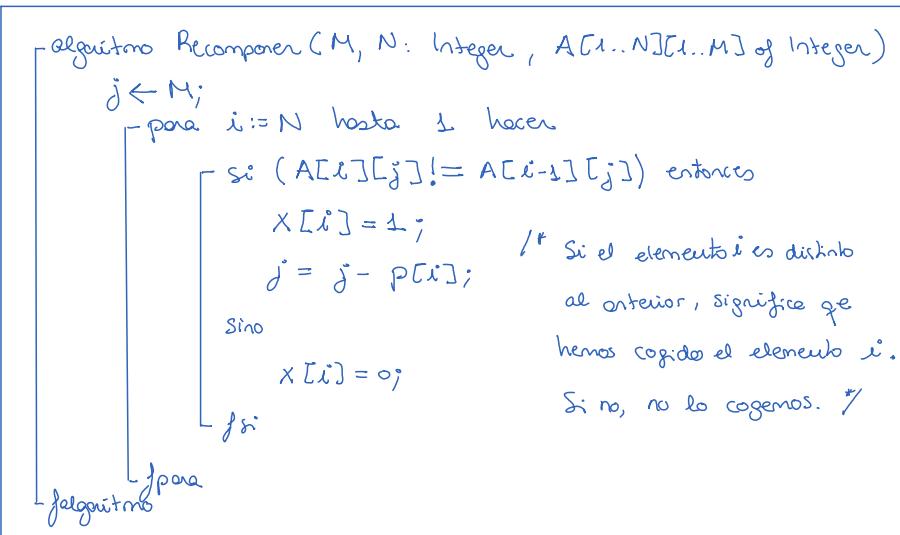
```
      A[i, j] = A[i-1, j];
```

```
    sino
```

```
      A[i, j] = max( A[i-1, j], A[i-1, j-p[i]] + b[i] );
```



Para recomponer la solución hay que aplicar el siguiente algoritmo:



Aplicemos el algoritmo al ejemplo propuesto:

j							
A	0	1	2	3	4	5	= M
i	0	0	0	0	0	0	
1	0	2	2	2	2	2	
2	0	3	5	5	5	5	
3	0	3	5	5	6	<u>9</u>	

↓
i
↑
N

Recomponemos la solución:

$$(i=3; j=5; X=(-1,-1,-1)) \leftarrow \text{INICIALMENTE}$$

$$(A[i][j] = 9) \neq (A[i-1][j] = 5) \rightarrow \text{Cogemos el elemento } i=3$$

$$X = (-1,-1,1);$$

$$j = 5 - p[i] \rightarrow j = 5 - 4 = 1$$

$$(A[i][j] = 3) \neq (A[i-1][j] = 2) \rightarrow \text{Cogemos el elemento } i=2$$

$$X = (-1,1,1,1);$$

$$j = 1 - 1 = 0$$

$(A[i, j] = 0) == (A[i-1, j] = 0) \rightarrow$ No cogemos el elemento
 $i=1$

$x = 10, 1, 14$

El beneficio se calcularía

```

funcion Beneficio( X:[1..n] of Integer, b[1..N] of Integer)
    beneficio = 0;
    para i=1 hasta N hacer
        beneficio := b[i] * X[i];
    fin para
    devolver beneficio;
finfunc.
```

2) Utilizaremos un esquema de árbol binario. Consideraremos el problema como un problema de toma de decisiones donde:

- 0: NO coge el objeto
- 1: Sí coge el objeto.
- 1: No se ha estudiado el objeto

La solución sera una N-tuple donde los valores podrán ser 0 ó 1.

Las funciones del esquema son las siguientes:

procedimiento ModularBT (solucion [1..N] of Integer)

nivel := 1;

solucion = 1-1,-1,...,-14;

Voa = -v0; Soa = Ø;

pact = 0; bact = 0

repetir

GENERAR(nivel, solucion);

si (EsSolucion(nivel)) AND

bact > Voa)

Voa := bact;

sino

si CRITERIO(nivel, solucion) entonces

nivel++;

sino

. / - - - - -

funcion GENERAR (nivel, solucion)

solucion[nivel] ++;

pact = pact + solucion[nivel] * p[nivel];

bact = bact + solucion[nivel] * b[nivel];

finfunc

funcion EsSolucion (nivel),

devolver ((nivel == N) AND (pact <= M));

finfunc

funcion Critero (nivel, solucion)

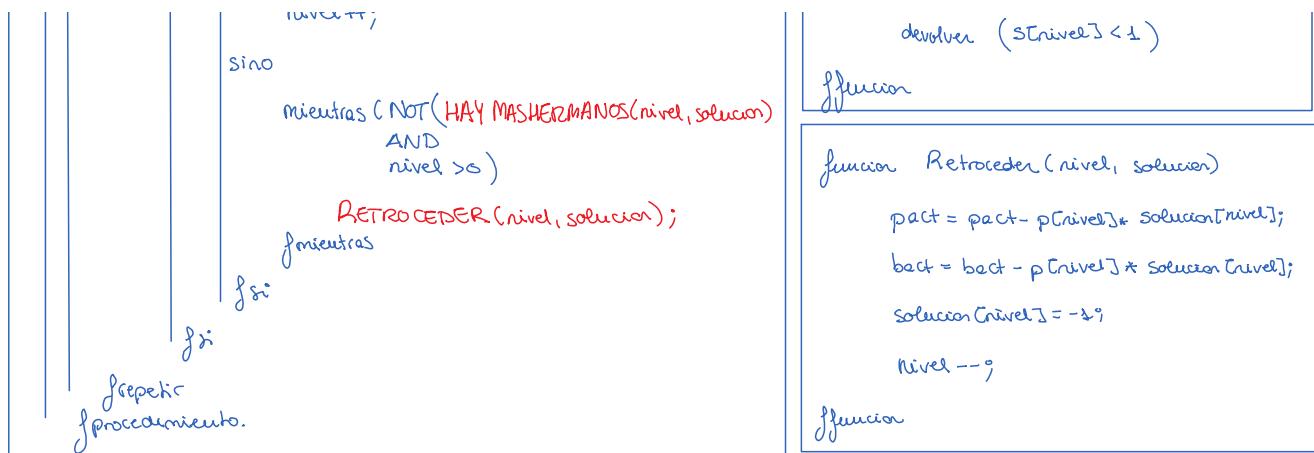
devolver (pact <= M AND nivel < N);

finfunc

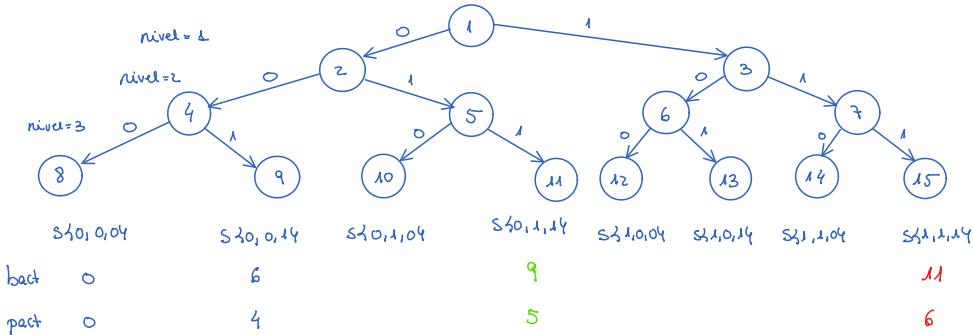
funcion HayMasHermanos (nivel, solucion)

devolver (S[nivel] < 1)

finfunc



Aplicaremos el algoritmo al ejemplo:



Ejercicio_4. (2 puntos)

Dado el lenguaje libre de contexto: $L = \{a^{2n}b^n c / n \geq 0\}$

- (0.5 pto.) Construir una gramática LL(1) que lo genere.
- (0.25 pto.) Obtener un Autómata con Pila asociado al lenguaje que acepte el mismo lenguaje por pila vacía.
- (0.25 pto.) Analizar por el autómata anterior, teniendo en cuenta el principio de preanálisis (lectura de un símbolo de la entrada con anticipación) las entradas "aaaabbc" y "abc".
- (0.25 pto.) Con la gramática LL(1), construir la tabla de análisis LL(1) y especificar el pseudocódigo de análisis sintáctico tabular.
- (0.25 pto.) Construir las trazas correspondientes al reconocimiento de las frases: "aaaabbc" y "abc" según el pseudocódigo especificado en el punto 4 anterior.
- (0.5 pto.) Especificar el pseudocódigo de análisis sintáctico LL dirigido por la sintaxis. Analizar si son correctas sintácticamente las entradas: "aaaabbc" y "abc"

$$1) \quad L = \{a^{2n}b^n c / n \geq 0\} \rightarrow \{c, aabc, aaaabbc\}$$

La gramática viene dada por la cuádruple:

$$G = (\Sigma_T, \Sigma_{NT}, S, P) \text{ donde:}$$

$$\Sigma_T : \{S, A, B, C\}$$

$$\Sigma_{NT} : \{a, b, c\}$$

S: axioma de la gramática

P: Producciones:

$$P = \begin{cases} S \rightarrow AB \\ A \rightarrow aaAbB \mid \lambda \\ B \rightarrow C \end{cases}$$

Podemos ver que la gramática no es recursiva a izquierdas.

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

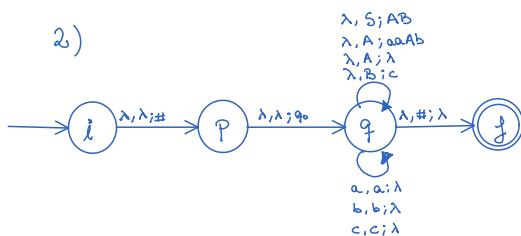
Ver en la misma ordenación, las transiciones de los símbolos siguientes

en Ø.

	PRIMERO	SIGUIENTE
S	{a, c}	{\$}
A	{a, λ}	{b, c}
B	{c}	{\$}

1. S::=AB
2. A::=aAb
3. λ
4. B::=c

2)



3)

Estado	PILA	ENTRADA	Acción	Indeterminación	Acción
i	λ	aaaabbc\$	(i, λ, λ, i, λ, #)		
p	#	aaaabbc\$	(p, λ, λ, i, q, S)		
q	S#	aaaabbc\$	(q, λ, S, q, AB)	S::=AB	
q	AB#	aaaabbc\$	(q, λ, A, q, aAb)	A::=aAb	
q	aAaB#	aaaabbc\$	(q, a, a, q, a, λ)	RECONOCER('a')	
q	aAaB#	aaaabbc\$	(q, a, a, q, λ)	RECONOCER('a')	
q	AbB#	aaaabbc\$	(q, λ, A, q, aAb)	A::=aAb	
q	aaAbB#	aaaabbc\$	(q, a, a, q, λ)	RECONOCER('a')	
q	aAbB#	abbc\$	(q, a, a, q, λ)	RECONOCER('a');	
q	AbB#	bbc\$	(q, λ, A, q, λ)	A::=λ	
q	bbB#	bbc\$	(q, b, b, q, λ)	RECONOCER('b');	
q	bB#	bc\$	(q, b, b, q, λ)	RECONOCER('b')	
q	B#	c\$	(q, λ, B, q, C)	B::=C	
q	C#	C\$	(q, C, C, q, λ)	RECONOCER('c');	
q	#	\$	(q, λ, #, q, λ)		
f	λ	λ	ACEPTAR		
—	—	—	—	—	—

Estado	PILA	ENTRADA	Acción	Indeterminación	Acción
i	λ	abc\$	i, λ, λ; p#;		
p	#	abc\$	p, λ, λ; q, S		
q	S#	abc\$	q, λ, S, q, AB	S::=AB	
q	AB#	abc\$	q, λ, A, q, aAb	A::=aAb	
q	aAaB#	abc\$	q, a, a, q, a, λ	RECONOCER('a');	
q	aAbB#	bc\$	error(simbolo);	ERROR();	

4)

Y $A \rightarrow \alpha$ hacer

Y 'a' terminal != λ ∈ PRIM(A) hacer

Tabla [A][a] = α;

fʌ

Si λ ∈ PRIM(α) entonces

Y 'b' terminal != λ ∈ SIG(A) hacer

Tabla [A][a] = λ;

jʌ

fs;

fʌ

```

procedimiento Análisis Sintáctico ()
    Apilar (#);
    Apilar (S);

    leer_símbolo();

    mientras (NOT pila-vacia) hacer
        switch cima-pila of
            case terminal:
                Si (Cima-pila == símbolo)
                    leer_símbolo();
                    Desapilar ();
                Sino
                    error_sintáctico ();
                Fin

            case No-terminal:
                Si Tabla[cima-pila][símbolo] != error() entonces
                    Desapilar ()
                    Apilar (Tabla[cima-pila][símbolo]);
                Sino
                    error_sintáctico ();
                Fin

        Finswitch
    Finmientras

    Si cima-pila != # entonces
        error_sintáctico ();
    Sino
        Escribir ('ACEPTADA');

    Fin

finprocedimiento

```

5)

PILA	ENTRADA	ACCION
λ	aaaaabbcc \$	Apilar ('#');
#	aaaaabbcc \$	Apilar (S);
S#	aaaaabbcc \$	Desapilar (); Apilar (aa);
aa#	aaaaabbcc \$	Desapilar (A); Apilar (aa);
aaAb#	aaaaabbcc \$	Desapilar (a); Leer();
aAbB#	aaaaabbcc \$	Desapilar (a); Leer();
Aab#	aaaaabbcc \$	Desapilar (A); Leer();
aaAbB#	aaaaabbcc \$	Desapilar (aa); Leer();
aaabb#	aaaaabbcc \$	Desapilar (a); Leer();
abb#	aaaaabbcc \$	Desapilar (ab); Leer();
bb#	aaaaabbcc \$	Desapilar (b); Leer();
b#	aaaaabbcc \$	Desapilar (b); Leer();
B#	aaaaabbcc \$	Desapilar (B); Apilar (c);
c#	aaaaabbcc \$	Desapilar (c); Leer();
#	aaaaabbcc \$	ACEPTADA

PILA	ENTRADA	ACCION
λ	a b c \$	Apilar ('#');
#	a b c \$	Apilar (S);
S#	a b c \$	Desapilar (); Apilar (aa);
AB#	a b c \$	Desapilar (A); Apilar (aa);
aaAbB#	a b c \$	Desapilar (a); Desapilar (a); Leer();
abb#	a b c \$	Desapilar (ab); Leer();
bb#	a b c \$	Desapilar (b); Leer();
b#	a b c \$	Desapilar (b); Leer();
B#	a b c \$	Desapilar (B); Apilar (c);
c#	a b c \$	Desapilar (c); Leer();
#	a b c \$	error_sintáctico();

6)

```

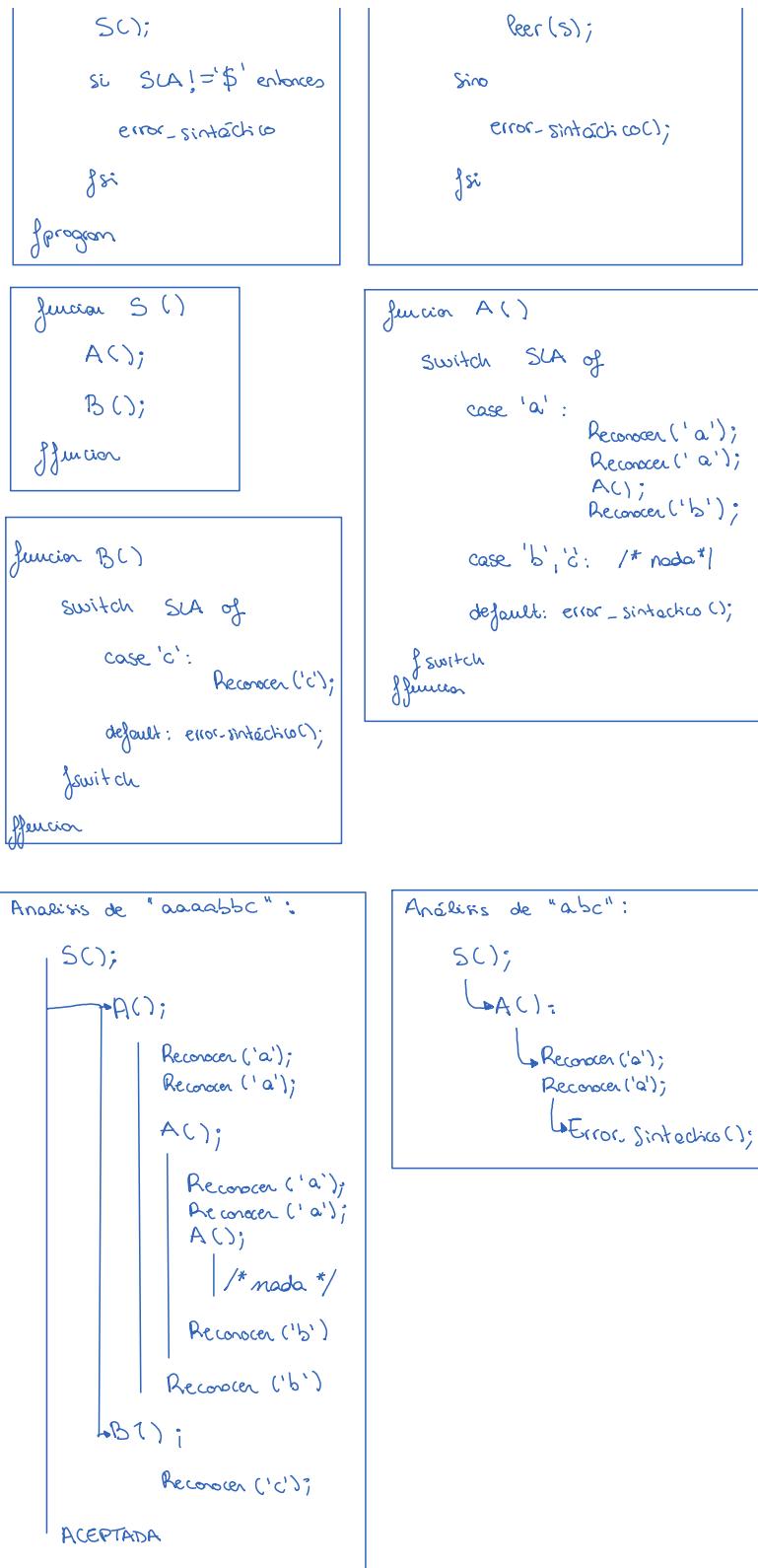
program Programa_Principal
    SLA = leer_símbolo();
    SL();
    Si SLA != '$' entonces

```

```

procedimiento Reconocer(símbolo: s)
    Si SLA == s entonces
        Leer(s);
    Sino

```



Ejercicio_5. (2 puntos)

Una puerta blindada dispone de una única cerradura. Para abrirla es necesario hacer girar en ella tres llaves diferentes (denominadas a, b y c), en un orden predeterminado, que se describe a continuación:

- Llave a, seguida de llave b, seguida de llave c, o bien
- Llave b, seguida de llave a, seguida de llave c.

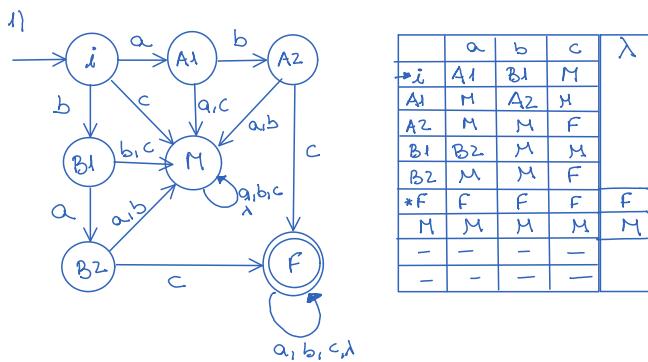
Si no se respeta este orden, la caja se bloquea, y es imposible su apertura; por ejemplo, si se hace girar la llave 1, se retira la misma, se introduce de nuevo y se hace girar.

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 de un alfabeto, sobre el que define el lenguaje L cuyas palabras son las secuencias permitidas de apertura de la caja fuerte. Por ejemplo, abcbc es una palabra del referido lenguaje.

Se pide:

1. (0.5 ptos.) Diseño del autómata AFND que acepta el lenguaje L.
2. (0.5 pto.) Gramática que genera las palabras de L.
3. (0.5 pto.) AFD mínimo equivalente del apartado 1.
4. (0.5 pto.) La expresión regular del lenguaje reconocido por el autómata del apartado 3)



2) La gramática que genera las palabras de L viene dada por la cuádruple: $G(\Sigma_{NT}, \Sigma_T, S, P)$ donde

$$\Sigma_{NT} = \{ i, 1.1, 1.2, 2.1, 2.2, 3, b \}$$

$$\Sigma_T = \{ a, b, c \}$$

$$S = i$$

P son las producciones:

$$\begin{aligned}
 P = & \left| \begin{array}{l}
 i \rightarrow aA_1 \quad | \quad bB_1 \\
 A_1 \rightarrow bA_2 \\
 A_2 \rightarrow cF \quad | \quad c \\
 B_1 \rightarrow aB_2 \\
 B_2 \rightarrow cF \quad | \quad c \\
 F \rightarrow af \quad | \quad bf \quad | \quad cf \quad | \quad a \quad | \quad b \quad | \quad c
 \end{array} \right.
 \end{aligned}$$

3) AFD equivalente

	a	b	c
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

$$Q_0 = CL(i) = \lambda i \cup$$

$$f'(Q_0, a) = \lambda A_1 \cup = Q_1$$

$$f'(Q_0, b) = \lambda B_1 \cup = Q_2$$

$$f'(Q_0, c) = \lambda M \cup = M$$

$$f'(Q_1, a) = M$$

$$f'(Q_1, b) = \lambda A_2 \cup = Q_3$$

$$f'(Q_1, c) = M$$

...

Minimizamos el AFD con el algoritmo de conjuntos/cociente:

$$E/Q_0 = (C_0 = \{ Q_0, Q_1, Q_2, Q_3, Q_4 \}, C_1 = \{ Q_5 \}, C_2 = \{ M \})$$

Son distinguibles
↓
nuevos
cjt.

$$\left\{
 \begin{array}{l}
 f'(Q_0, a) = C_0; \quad f'(Q_0, b) = C_0; \quad f'(Q_0, c) = C_2 \\
 f'(Q_1, a) = C_2; \quad f'(Q_1, b) = C_0; \quad f'(Q_1, c) = C_2 \\
 f'(Q_2, a) = C_0; \quad f'(Q_2, b) = C_2; \quad f'(Q_2, c) = C_2 \\
 f'(Q_3, a) = C_2; \quad f'(Q_3, b) = C_2; \quad f'(Q_3, c) = C_1 \\
 f'(Q_4, a) = C_2; \quad f'(Q_4, b) = C_2; \quad f'(Q_4, c) = C_1
 \end{array}
 \right.$$

$$E/Q_1 = \{C_0 = \{Q_0\}, C_1 = \{Q_5\}, C_2 = \{M\}, C_3 = \{Q_1, Q_2, Q_3, Q_4\}\}$$

Distingüibles

$$\begin{cases} f'(Q_1, a) = c_2; f'(Q_1, b) = c_3; f'(Q_1, c) = c_2 \\ f'(Q_2, a) = c_3; f'(Q_2, b) = c_2; f'(Q_2, c) = c_2 \\ f'(Q_3, a) = c_2; f'(Q_3, b) = c_2; f'(Q_3, c) = c_1 \\ f'(Q_4, a) = c_2; f'(Q_4, b) = c_2; f'(Q_4, c) = c_1 \end{cases}$$

$$E/Q_2 = \{C_0 = \{Q_0\}, C_1 = \{Q_5\}, C_2 = \{M\}, C_3 = \{Q_1\}, C_4 = \{Q_2, Q_3, Q_4\}\}$$

Distingüible

$$\begin{cases} f'(Q_2, a) = c_4; f'(Q_2, b) = c_2; f'(Q_2, c) = c_2 \\ f'(Q_3, a) = c_2; f'(Q_3, b) = c_2; f'(Q_3, c) = c_1 \\ f'(Q_4, a) = c_2; f'(Q_4, b) = c_2; f'(Q_4, c) = c_1 \end{cases}$$

$$E/Q_3 = \{C_0 = \{Q_0\}, C_1 = \{Q_5\}, C_2 = \{M\}, C_3 = \{Q_1\}, C_4 = \{Q_2\}$$

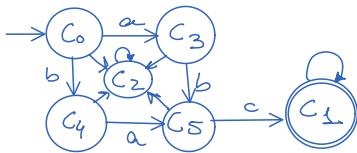
$$C_5 = \{Q_3, Q_4\}$$

No Distingüibles

$$\begin{cases} f'(Q_3, a) = c_2; f'(Q_3, b) = c_2; f'(Q_3, c) = c_1 \\ f'(Q_4, a) = c_2; f'(Q_4, b) = c_2; f'(Q_4, c) = c_1 \end{cases}$$

Por tanto, el AFD mínimo equivalente es:

	a	b	c
C ₀	C ₃	C ₄	C ₂
*C ₁	C ₁	C ₁	C ₁
C ₂	C ₂	C ₂	C ₂
C ₃	C ₂	C ₅	C ₂
C ₄	C ₅	C ₂	C ₂
C ₅	C ₂	C ₂	C ₁



4) La expresión regular será la siguiente:

$$\begin{cases} x_0 = ax_3 + bx_4 \\ x_1 = ax_1 + bx_1 + cx_1 + a + b + c \\ x_3 = bx_5 \\ x_4 = ax_5 \\ x_5 = cx_1 + c \end{cases}$$

* Los ESTADOS MUERTOS o ESTADOS DE ABSORCIÓN NO SE PONEN EN LAS ECUACIONES.
SI LO PONES NO SALE BIEN :*

$$x_1 = (a+b+c)x_1 + (a+b+c) \Leftrightarrow x_1 = (a+b+c)^*(a+b+c) \Rightarrow$$

$$\Rightarrow x_1 = (a+b+c)^*$$

$$x_5 = c(a+b+c)^* + c$$

$$x_4 = a (c(a+b+c)^* + c)$$

$$x_3 = b (\dot{c}(a+b+c)^* + c)$$

$$x_0 = a(b(c(a+b+c)^* + c) + b(a(c(a+b+c)^* + c)) \rightarrow$$

$$\rightarrow x_0 = abc(a+b+c)^* + abc + bac(a+b+c)^* + bac$$

Ejercicio_6. (1 punto)

Dado el lenguaje $(01)^n$ con $n \geq 0$,

- (0.5 pto.) Marcar el autómata que reconoce el lenguaje indicado justificando la respuesta.
- (0.5 pto.) Obtener el AFD mínimo equivalente del autómata seleccionado.
 - AF = $\{0,1\}, \{A,B,C,F\}, f, A, \{F\}$ con $f(A,0)=B, f(A,\lambda)=\lambda, f(C,0)=B, f(B,1)=\lambda, f(F,0)=F$
 - AF = $\{0,1\}, \{A,B,C,F\}, f, A, \{F\}$ con $f(A,0)=B, f(A,\lambda)=\lambda, f(C,0)=B, f(B,1)=C, f(F,1)=F$
 - AF = $\{0,1\}, \{A,B,C,F\}, f, A, \{F\}$ con $f(A,B)=0, f(A,F)=\lambda, f(C,B)=0, f(B,C)=1, f(B,F)=1$
 - AF = $\{0,1\}, \{A,B,C,F\}, f, A, \{F\}$ con $f(B,0)=A, f(F,\lambda)=A, f(B,0)=C, f(C,1)=B, f(F,1)=B$

a) El autómata que reconoce el lenguaje indicado es el ②

porque:

El ① transiciones hacia λ , cuando λ no es un estado de AF.

El ③ utiliza símbolos no terminales (estados) como símbolos terminales.

El ④ no tiene ninguna transición que comience en el estado inicial A.

b)

AFND descrito:

	0	1	λ
A	$\lambda B \dashv$	\emptyset	$\lambda F \dashv$
B	\emptyset	$\lambda C, F \dashv$	\emptyset
C	$\lambda B \dashv$	\emptyset	\emptyset
F	\emptyset	\emptyset	\emptyset

AFD equivalente

	0	1
$\overset{*}{Q}_0$	Q_1	Q_2
Q_1	Q_2	Q_3
Q_2	Q_2	Q_2
$\overset{*}{Q}_3$	Q_1	Q_2

$$Q_0 = CL(A) = \{A, F\}$$

$$f'(Q_0, 0) = \lambda B \dashv = Q_1$$

$$f'(Q_0, 1) = \emptyset = Q_2$$

$$f'(Q_1, 0) = \emptyset = Q_2$$

$$f'(Q_1, 1) = \lambda C, F \dashv = Q_3$$

$$f'(Q_2, 0) = \lambda B \dashv = Q_1$$

$$f'(Q_2, 1) = \emptyset = Q_2$$

	0	1
$\overset{*}{Q}_0$	Q_1	Q_2
Q_1	Q_2	Q_3
Q_2	Q_2	Q_2
$\overset{*}{Q}_3$	Q_1	Q_2

Para minimizar el autómata usaremos

el algoritmo del conjunto cociente:

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

$$f'(Q_1, 0) = C_0 ; f'(Q_1, 1) = C_1 \quad \left. \right\} \text{ Distinguibles}$$

$$f'(Q_2, 0) = C_0 ; f'(Q_2, 1) = C_0 \quad \left. \right\} \text{ Indistinguibles}$$

$$f'(Q_3, 0) = C_0 ; f'(Q_3, 1) = C_0 \quad \left. \right\} \text{ Indistinguibles}$$

$$\left. \begin{array}{l} f'(Q_3, 0) = C_0 ; f'(Q_3, 1) = C_0 \\ f'(Q_0, 0) = C_0 ; f'(Q_0, 1) = C_0 \end{array} \right\} \text{Indistinguibles}$$

$$E/Q_A = (C_0 = \{Q_1\}, C_1 = \{Q_3, Q_0\}, C_2 = \{Q_2\})$$

$$\left. \begin{array}{l} f'(Q_3, 0) = C_0 ; f'(Q_3, 1) = C_2 \\ f'(Q_0, 0) = C_0 ; f'(Q_0, 1) = C_2 \end{array} \right\} \text{Indistinguibles}$$

Por tanto, el AFD mínimo equivalente será:

	0	1
C ₁	C ₀	C ₂
C ₀	C ₂	C ₁
C ₂	C ₂	C ₂

$$AF = (Q_0, 14, \{C_0, C_1, C_2\}, f, C_1, \{C_2\})$$

donde f :

$$f = \left\{ \begin{array}{l} f(C_0, 0) = C_2 \\ f(C_0, 1) = C_1 \\ f(C_1, 0) = C_0 \\ f(C_1, 1) = C_2 \\ f(C_2, 0) = C_2 \\ f(C_2, 1) = C_2 \end{array} \right\}$$