Febrero 2014

miércoles, 19 de enero de 2022 8:52

Ejercicio_1. (2 puntos)

El algoritmo de ordenación MergeSort puede ser implementado por:

```
MergeSort (A, p, r) /* Ordena un vector A desde p hasta r */
    if p<r { /* Dividir en dos trozos de tamaño igual (o lo más parecido posible), es decir ⌈n/2⌉ y ⌊n/2⌋ */
        q = ⌊(p+r)/2⌋;          /* Divide */
        /* Resolver recursivamente los subproblemas */
        MergeSort (A,p,q);       /* Resuelve */
        MergeSort (A,q+1,r) ;    /* Resuelve */
        /* Combinar: mezcla dos listas ordenadas en O(n) */
        Merge (A,p,q,r);         /* Combina*/
    }
```

- El algoritmo utiliza para su implementación la función Merge que tiene un coste **$\Theta(n)$**.

Se pide:

a. (0,75 puntos). Calcular la complejidad del algoritmo propuesto por el método de la **ecuación característica**.
b. (0,5 puntos). Calcular la complejidad del algoritmo propuesto por el Teorema maestro.
c. (0,75 puntos). Comparar el algoritmo propuesto con el siguiente (Calcular la complejidad y compararlas):

```
MergeSort_bad (A, p, r) /* Ordena un vector A desde p hasta r */
    if p<r {
        MergeSort_bad (A,p,p);
        MergeSort_bad (A,p+1,r) ;
        Merge (A,p,p,r);
    }
```

$$T(n) = \begin{cases} c_1 & n = 1 \\ c_2 + 2T(\frac{n}{2}) + c_3 n & n > 1 \end{cases}$$

$n = 2^m$

a) $T(n) = c_2 + 2T(\frac{n}{2}) + c_3 n \rightarrow$ $m = \log_2 n$

$T(n) - 2T(\frac{n}{2}) = c_2 + c_3 n \rightarrow$

$T(2^m) = t_m$

$T(2^m) - 2T(2^{m-1}) = c_2 + c_3 \cdot 2^m \rightarrow$

$t_m - 2t_{m-1} = c_2 + c_3 \cdot 2^m \cdot n^0$

$\rightarrow (x-2)(x-2) = 0 \; \{ \boxed{r = 2 \text{ dobles}}$

$t_m = K_1 2^m + K_2 \cdot 2^m \cdot m \rightarrow T(2^m) = K_1 \cdot 2^m + K_2 2^m \cdot m$

$\rightarrow \boxed{T(n) = K_1 \cdot n + n \log_2 n \cdot K_2 \in O(n \log_2 n)}$

$$\boxed{T(n) = K_1 \cdot n + n \log_2 n \cdot K_2 \in O(n \log_2 n)}$$

b) $a T(\frac{n}{b}) + n^k \cdot \log^p(n)$

NOTA: El Teorema maestro es:

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

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

$$\boxed{T(n) \in O(n \cdot \log n)}$$

c)

MergeSort_bad (*A*, p, r) /* Ordena un vector *A* desde *p* hasta *r* */

```
if p<r {
        MergeSort_bad (A,p,p);
        MergeSort_bad (A,p+1,r) ;
        Merge (A,p,p,r);
    }
```

$$T(n) = \begin{cases} 1 & n \leq 1 \\ T(1) + T(n-1) + n & n > 1 \end{cases}$$

$$T(n) = T(n-1) + (\overset{n}{n+1})$$

$$T(n) - T(n-1) = 1^n n^{1}$$

$$(x-1)(x-1)^{1+1} = 0 \rightarrow (x-1)(x-1)^2 = 0 \begin{cases} r:1 \\ \text{triple} \end{cases}$$

$$T(n) = K_1 \cdot 1^n + K_2 \cdot 1^n \cdot n + K_3 \cdot 1^n \cdot n^2$$

$$= K_1 + K_2 n + K_3 \cdot n^2 \in O(n^2)$$

$$\lim_{n \to \infty} \frac{n \log n}{n^2} = \lim_{n \to \infty} \frac{\log n}{n} =$$

$$= \lim_{n\to\infty} \frac{f'(\log n)}{f'(n)} = \lim_{n\to\infty} \frac{1/n \ln(2)}{1}$$

$$= \lim_{n\to\infty} \frac{1}{n \ln(2)} = \frac{1}{\infty} = 0 \quad (n^2 > n\log n)$$

3)

Ejercicio_3. (1,5 puntos)

Dado el autómata siguiente,

f	a	b
→0	1	3
1	0	3
2	1	4
*3	5	5
4	3	3
*5	5	3

Obtener:

a. (0.5 puntos). El A.F.D. mínimo equivalente.

b. (0.5 puntos). La expresión regular del lenguaje reconocido por el autómata del apartado anterior.

c. (0.5 puntos). La gramática de tipo 3 para el lenguaje.

$$Q/E_0 = (\overbrace{\{0,1,2,4\}}^{C1}, \overbrace{\{3,5\}}^{C2})$$

$$Q/E_1 = (\underbrace{\{0,1\}}_{C1}, \{2\}, \{4\}, \underbrace{\{3,5\}}_{C2})$$

$f(0,a) = 1$ C1
$f(0,b) = 3$ C2

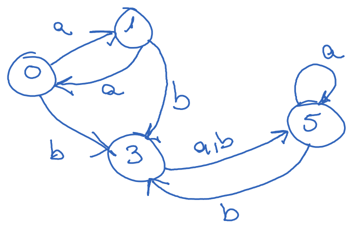
$f(1,a) = 0$ C1
$f(1,b) = 3$ C2

$f(2,a) = 1$ C1
$f(2,b) = 4$ C1

$f(3,a) = 5$ C2
$f(3,b) = 5$ C2

$f(4,a) = 3$ C2
$f(4,b) = 3$ C2

$f(5,a) = 5$ C2
$f(5,b) = 3$ C2



El estado 2 y el estado 4 son inaccesibles desde el estado inicial. Por tanto podemos eliminarlos

El autómata mínimo será:

	a	b
→ C1	C1	C2
*C2	C2	C2

b) El lenguaje aceptado por el autómata del apartado a) será:

$$\begin{cases} X_0 = aX_0 + bX_1 + b \\ X_1 = aX_1 + bX_1 + a + b \end{cases}$$

Resolvemos el sistema aplicando la regla de inferencia

$$\boxed{X = aX + b \Leftrightarrow X = a^* b}$$

$$X_1 = aX_1 + a \quad + \quad bX_1 + b$$

$$X_1 = (a+b)X_1 + (a+b) \longleftrightarrow$$

$$X_1 = (a+b)^* (a+b)$$

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

$$= a^* (b((a+b)^* \quad (a+b)) + b) =$$

$$= a^* b ((a+b)^* \quad (a+b)) + a^* b =$$

$$= a^* b (\overbrace{(a+b)^* \quad (a+b)) + \lambda} \Rightarrow (a^* = \lambda + a a^*)$$

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

$$\boxed{X_0 = a^* b(a \mid b)^*}$$

c) La gramática del lenguaje viene dada por la quíntupla:

$G(L) = (\Sigma_T, \Sigma_{NT}, S, P)$ Donde:

- $\Sigma_T = \{a, b\}$
- $\Sigma_{NT} = \{C_1, C_2\}$
- $S = C_1$
- P: Cjto de producciones formado por:

$$\boxed{\begin{array}{l} C_1 \rightarrow aC_0 \mid bC_1 \mid b \\ C_0 \rightarrow aC_1 \mid bC_1 \mid a \mid b \end{array}}$$

Ejercicio_4. (1,5 puntos)

Dado el lenguaje **(01)^n** con **n≥0**,

1) (0,75 puntos). Seleccionar, **justificando la respuesta**. el autómata que reconoce el lenguaje indicado.
   a. AF=[{0,1}, {A,B,C,F}, f, A, {F}] con f(A,0)=B, f(A, λ)= λ, f(C,0)=B, f(B,1)=C, f(B,1)= λ ×
   b. AF=[{0,1}, {A,B,C,F}, f, A, {F}] con f(A,0)=B, f(A, λ)=F, f(C,0)=B, f(B,1)=C, f(B,1)=F
   c. AF=[{0,1}, {A,B,C,F}, f, A,{F}] con f(A, B)=0, f(A,F)= λ, f(C,B)=0, f(B,C)=1, f(B,F)=1 ×
   d. AF=[{0,1}, {A,B,C,F}, f, A, {F}] con f(B,0)=A, f(F, λ)=A, f(B,0)=C, f(C,1)=B, f(F,1)=B ×
2) (0,75 puntos). Obtener el AFD mínimo equivalente del autómata seleccionado en el apartado anterior.

1)

a) No es correcta porque el estado de llegada de una transición no puede ser λ. Tiene que ser un estado de Σ_{NT}.

c) No es correcta porque el estado de destino de una transición no puede ser un símbolo. Tiene que ser un estado de Σ_{NT}

d) No es correcta porque no hay ninguna transición que tenga como origen el estado inicial (A).

2) El AF del apartado anterior es:

	0	1	λ
→ A	{B}		{F}
B		{C,F}	
C	{B}		
D			
E			
*F			

Como los estados D y E no se utilizan, los podemos eliminar.

Empezaremos a transformar el AFND-λ a AFD.

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

$CL(f(Q_0, 0)) = \{B\} = Q_1$

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

$CL(f(Q_1, 0)) = CL(\{\emptyset\}) = Q_2$

$CL(f(Q_1, 1)) = CL(\{C, F\}) = \{C, F\} = Q_3^*$

$CL(f(Q_2, 0)) = \emptyset$

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

$CL(f(Q_3, 0)) = CL(\{B\}) = \{B\} = Q_1$

$CL(f(Q_3, 1)) = CL(\emptyset) = \emptyset = Q_2$

AFD	0	1
→ *Q_0	Q_1	Q_2
Q_1	Q_2	Q_3
Q_2	Q_2	Q_2
*Q_3	Q_1	Q_2

Una vez hemos creado el AFD a partir del AFND, lo minimizamos con el método de conjunto/cociente:

$$Q/E_0 = (C_0 = \{Q_2, Q_1\},\ C_1 = \{Q_0, Q_3\})$$

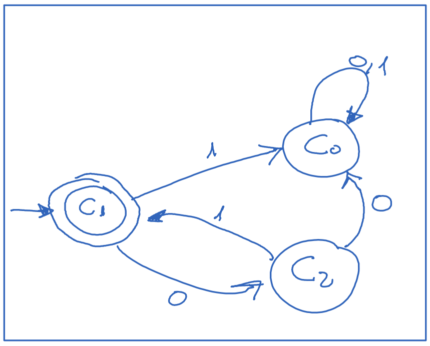
$$\left.\begin{array}{ll} f(Q_2, 0) = Q_2 & f(Q_1, 0) = Q_2 \\ f(Q_2, 1) = Q_2 & f(Q_1, 1) = Q_3 \end{array}\right\}$$ Como Q_2 y Q_1 son distinguibles → Creamos un nuevo conjunto

$$Q/E_1 = (C_0 = \{Q_2\},\ C_1 = \{Q_0, Q_3\},\ C_2 = \{Q_1\})$$

$$\left.\begin{array}{ll} f(Q_0, 0) = Q_1 & f(Q_3, 0) = Q_1 \\ f(Q_0, 1) = Q_2 & f(Q_3, 1) = Q_2 \end{array}\right\}$$ Como no son distinguibles no creamos otro conjunto

Por tanto, el AFD mínimo será:

	0	1
C_0	C_0	C_0
→ * C_1	C_2	C_0
C_2	C_0	C_1



Ejercicio_5. (2 puntos)

Dada la gramática:

$S \rightarrow S = A \mid A$

$A \rightarrow A := B \mid B$

$B \rightarrow (S) \mid a \mid b$

a. (0.5 puntos). Comprobar si es LL(1), eliminar la recursividad a la izquierda y obtener la gramática LL(1) equivalente.

b. (0.5 puntos). Convertir la gramática del apartado anterior en un autómata con pila que acepte el mismo lenguaje por pila vacía.

c. (0.5 puntos). Analizar, teniendo en cuenta el principio de preanálisis (lectura de un símbolo de la entrada con anticipación) la entrada "**(a)**".

d. (0.5 puntos). Implementar el seudocódigo de análisis descendente dirigido por la sintaxis para la gramática obtenida LL(1).

a)

Eliminar Recursividad a izquierda

$$S \to S = A \mid A \quad \begin{cases} S \to AS' \\ S' \to = AS' \\ \quad \mid \lambda \end{cases}$$

$$A \to A := B \mid B \quad \begin{cases} A \to BA' \\ A' \to := BA' \mid \lambda \end{cases}$$

$$B \to (S) \mid a \mid b$$

Por tanto, la gramática equivalente será:

1) $S \to AS'$
2) $S' \to =AS' \mid$
3) λ
4) $A \to BA'$
5) $A' \to :=BA' \mid$
6) λ
7) $B \to (S) \mid$
8) $a \mid$
9) b

La condición para que una gramática sea LL(1) es:

"Para cada par de reglas de la gramática con el mismo antecedente, la intersección de sus símbolos directores es vacía"

$$PRIM(=AS') \cap \underset{SIG(S')}{\overset{\shortparallel}{PRIM(\lambda)}} =$$

$$PRIM(:=BA') \cap SIG(A') =$$

$$PRIM((S)) \cap PRIM(a) \cap PRIM(b) = \emptyset$$

Calculamos los conjuntos PRIMERO Y SIGUIENTE:

	PRIMERO	SIGUIENTE
S	$\{(, a, b\}$	$\{\$,)\}$
S'	$\{=, \lambda\}$	$\{\$,)\}$
A	$\{(, a, b\}$	$\{=, \$,)\}$
A'	$\{:=, \lambda\}$	$\{=, \$,)\}$
B	$\{(, a, b\}$	$\{:=, =, \$,)\}$

Comprobamos las condiciones anteriores:

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

$$PRIM(:=BA') \cap SIG(A') =$$

$$\{:=\} \cap \{=, \$,)\} = \emptyset$$

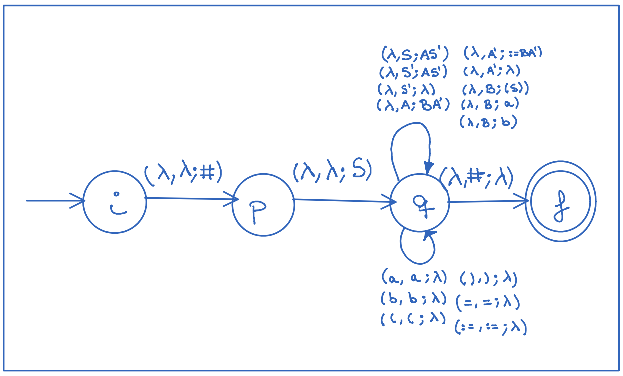
- $PRIM((S)) \cap PRIM(a) \cap PRIM(b) =$
$$\{(\} \cap \{a\} \cap \{b\} = \emptyset$$

Como las condiciones se cumplen, entonces la gramática es LL(1)

b) Un autómata con pila (AP) viene definido por la séptupla:

$$AP = (\Sigma, \Gamma, Q, A_0, q_0, f, F) \text{ donde:}$$

- Σ : alfabeto que acepta el autómata
- Γ : alfabeto de la pila
- Q : Conjunto de estados
- A_0 : Símbolo inicial de la pila (#)
- q_0 : Estado inicial.
- f : funciones de transición
- F : Conjunto de Estados Finales



c)

Estado	Pila	Entrada	Acción	Indeterminación	Acción
i	λ	(a)$	$(i,\lambda,\lambda;p,\#)$		
p	#	(a)$	$(p,\lambda,\lambda;q,S)$		
q	S#	(a)$	$(q,\lambda,S;q,AS')$		$S\to AS'$
q	AS'#	(a)$	$(q,\lambda,A;q,BA')$		$A\to BA'$
q	BA'S'#	(a)$	$(q,\lambda,B;q,(S))$	$B\to(S)$ ← $B\to a$ $B\to b$	$B\to(S)$
q	(S)A'S'#	(a)$	$(q,(,(;q,\lambda)$		Reconocer('(');
q	S)A'S'#	a)$	$(q,\lambda,S;AS')$		$S\to AS'$
q	AS')A'S'#	a)$	$(q,\lambda,A;q,BA')$		$A\to BA'$
q	BA'S')A'S'#	a)$	$(q,\lambda,B;q,a)$	$B\to(S)$ $B\to a$ ← $B\to b$	$B\to a$
q	aA'S')A'S'#	a)$	$(q,a,a;q,\lambda)$		Reconocer('a');
q	A'S')A'S'#	)$	$(q,\lambda,A';q,\lambda)$	$A'\to:=BA'$ $A'\to\lambda$ ←	$A'\to\lambda$
q	S')A'S'#	)$	$(q,\lambda,S';q,\lambda)$	$S'\to AS'$ $S'\to\lambda$ ←	$S'\to\lambda$
q	)A'S'#	)$	$(q,),);q,\lambda)$		Reconocer(')');
[illegible]	[illegible]	$	[illegible]	$A'\to:=BA'$	[illegible]

q	)A'S'#	)$	(q,),);q,λ)		Reconocer(')');
q	A'S'#	$	(q,λ,A';q,λ)	A'→:=BA' A'→λ ←	A'→λ
q	S'#	$	(q,λ,S';q,λ)	S'→AS' S'→λ	S'→λ
q	#	$	(q,λ,#;f,λ)		
f	λ	λ	ACEPTAR		

d) La gramática:

1) S → AS'
2) S' → =AS' |
3) λ
4) A → BA'
5) A' → :=BA' |
6) λ
7) B → (S) |
8) a |
9) b

con los conjuntos PRIMERO y SIGUIENTE:

	PRIMERO	SIGUIENTE
S	{(, a, b}	{$,)}
S'	{=, λ}	{$,)}
A	{(, a, b}	{=, $,)}
A'	{:=, λ}	{=, $,)}
B	{(, a, b}	{:=, =, $,)}

El pseudocódigo del análisis descendente dirigido por la sintaxis para la gramática LL(1) es:

```
Símbolo SLA;

Programa_Principal;
{
   SLA = leer_símbolo();
   S();
   Si SLA != $ entonces
      Error();
}
```

```
Procedimiento Reconocer( Símbolo terminal){

   Si (SLA == terminal)
        leer_símbolo();
   Sino
        error_sintáctico();
}
```

```
Procedimiento S (){
  /* S→ AS' */

   A();
   S'();
}
```

```
Procedimiento A()
 /* A→BA' */

   B();
   A'();
```

```
Procedimiento S'(){

 /*S' →=AS' | λ*/
    CASE SLA OF
```

```
Procedimiento A'(){
 /* A'→:=BA' | λ */
   CASE SLA OF
```

```
Procedimiento A'(){
  /* A'->:=BA' | λ */
  CASE SLA OF
    ':=': Reconocer(':=');
          B();
          A'();
    '=',')','$': /* */

    else: error_sintáctico
  END CASE;
}
```

```
Procedimiento B()
  /* B-> (S) | a | b */

  CASE SLA OF

    '(' : Reconocer('(');
          S();
          Reconocer(')');

    'a' : Reconocer('a');
    'b' : Reconocer('b');
  else: error_sintáctico();
  END CASE;
}
```

```
/* S' -> =AS' | λ */
  CASE SLA OF

    '=':    Reconocer('=');
            A();
            S'();

    ')', '$' : /**/

    else:   error_sintáctico();
  END CASE;
```

Para aprobar una asignatura el/la estudiante tiene que hacer en total **n** tareas (exámenes, prácticas, trabajos, etc). Para cada una de ellas estima que le llevará cierto tiempo, $\mathbf{t_i}$. Como puede realizarlas a lo largo del curso, las quiere repartir entre las convocatorias de junio, septiembre y diciembre. En cada convocatoria puede sacar **M** unidades de tiempo como **máximo**. Se supone que todas las tareas se deben hacer y que no se pueden fraccionar (cada tarea va a una sola convocatoria). El **objetivo** es conseguir un reparto haciendo las tareas cuanto antes, no dejarlas para el final, es decir, minimizar el tiempo dedicado en la convocatoria de diciembre. En caso de empate en diciembre, minimizar el tiempo en la de septiembre.

a. (1 punto). Diseñar un algoritmo **voraz** para resolver el problema aunque no se garantice siempre la solución óptima. Proponer y contrastar dos criterios de selección.
Aplicar el algoritmo al caso: **n** = 5, **M** = 16, **t** = {7, 5, 3, 5, 6}.

b. (1 punto). Resolver el problema mediante **programación dinámica**. Definir la ecuación recurrente, los casos base, las tablas y el algoritmo para rellenarlas. No hay que aplicar el ejemplo.

c. (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.

a) Como criterio de selección utilizaré la actividad que termine antes.

```
funcion Ejercicio2Voraz( t: array [1..N] of Integer, M: Integer )
    J[1..N], S[1..N], D[1..N];
    tordenado = ordenarPorCriterio();
    J ← {1}; tJ = t[1];
    S ← ∅ ; tS = 0;
    D ← ∅ ; tD = 0;
    para i = 2 hasta N hacer
        si ( t[i] + tJ ≤ M ) entonces
            J ← J ∪ {i};
        sino
            si ( t[i] + tS ≤ M ) entonces
                S ← S ∪ {i};
            fsi
        sino
            si ( t[i] + tD ≤ M ) entonces
                D ← D ∪ {i};
            fsi
        sino
            Escribir ("No se puede hacer la actividad", i );
            error();
        fsi
    fpara
```

Aplicamos el algoritmo a los datos: $n = 5$, $M = 16$, $t = \{7, 5, 3, 5, 6\}$

1) t Ordenado $= \{3, 5, 5, 6, 7\}$;

$\begin{cases} J \leftarrow \{1\}; \quad S \leftarrow \emptyset; \quad D \leftarrow \emptyset; \\ tJ = 3; \end{cases}$

$\begin{cases} J \leftarrow \{1, 2\} \quad S \leftarrow \emptyset \quad D \leftarrow \emptyset \\ tJ = 8; \end{cases}$

$\begin{cases} J \leftarrow \{1, 2, 3\} \\ tJ = 13; \end{cases}$

$\begin{cases} J = \{1, 2, 3\} \quad S \leftarrow \{4\} \\ tJ = 13; \quad tS = 6; \end{cases}$

$\begin{cases} S = \{1, 2, 3\} \quad S \leftarrow \{4, 5\} \quad D \in \emptyset \\ tJ = 13; \quad tS = 13; \quad tD = 0; \end{cases}$

$$\begin{cases} S = \{1,2,3\} & S \leftarrow \{4,5\} & D \leftarrow \emptyset \\ tJ = 13; & tS = 13; & tD = 0; \end{cases}$$

b)

$$A(i,j,s,d) \begin{cases} 0 & \text{si} \quad i = 0 \\ +\infty & \text{si} \quad j<0,\ s<0,\ d<0 \\ \min\{A(i-1, j-t_i, s, d) + A(i-1, j, s-t_i, d), \\ \quad t_i \cdot M + A(i-1, j, s, d-t_i)\} \end{cases}$$

$\left.\begin{matrix} 0 \text{ si } i=0 \\ +\infty \text{ si } j<0, s<0, d<0 \end{matrix}\right\}$ Casos Base

Tabla T:

array[(0...N), (0...M), (0..M), (0..M)]

El algoritmo será:

```
fucion Ejercicio2PD()

    para i=0 hasta N hacer
        para j=0 hasta M hacer
        para S = 0 hasta M hacer
        para D = 0 hasta M hacer

            si (i==0) entonces T[i,j,s,d]=0;
            sino
            T[i,j,s,d] = min(T(i-1,j-t_i,s,d)+T(i-1,j,s-t_i,d),
                                  t_i*M+T[i-1,j,s,d-t_i]);

            fsi
        fpara
       fpara
      fpara
    fpara
       Escribir("Tiempo Diciembre:", T[n,M,M,M] / M);
       Escribir("Tiempo septiembre", T[n,M,M,M] mod M);

fucion
```

c) La solución por Backtracking se representa con un vector s.

$S = \{ s_1, s_2 \dots s_N \}$ donde s_i puede tomar los valores $\{1, 2, 3\}$ en función de a qué convocatoria se asigne.

El arbol sería ternario, ya que puedes tomar 3 decisiones distintas para cada elemento

Las funciones genéricas que se utilizan son:

operación Inicializar
    voa:= +∞
    nivel:= 1
    s:= (0, 0, ..., 0)
    conv:= (0, 0, 0, 0)

operación Generar (nivel, s)
    conv[s[nivel]]:= conv[s[nivel]] - t[nivel]
    s[nivel]:= s[nivel] + 1
    conv[s[nivel]]:= conv[s[nivel]] + t[nivel]

operación Solución (nivel, s)
    devolver (nivel==n) AND (conv[s[nivel]]<=M)

operación Criterio (nivel, s)
    devolver (nivel<n) AND (conv[s[nivel]]<=M)

operación MasHermanos (nivel, s)
    devolver s[nivel] < 3

operación Retroceder (nivel, s)
    conv[s[nivel]]:= conv[s[nivel]] – t[nivel]
    s[nivel]:= 0
    nivel:= nivel - 1

operación Valor (s)
    devolver conv[3]*M + conv[2]