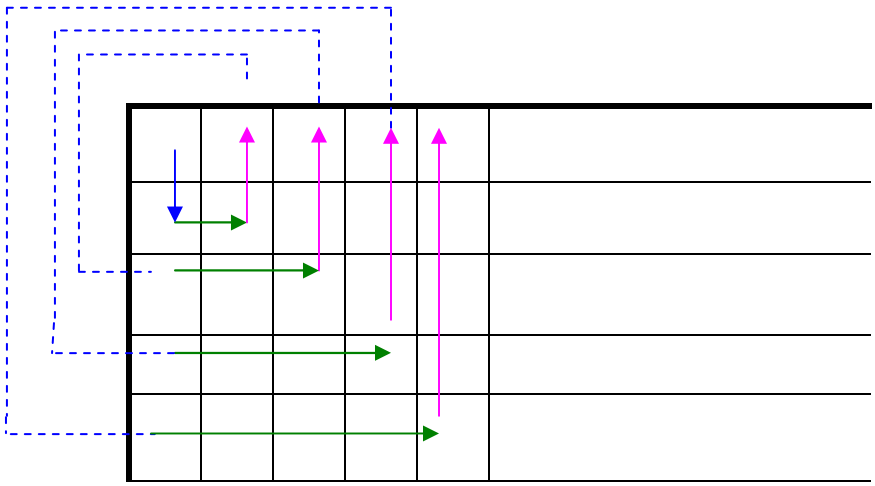
Hoja I.4

SOLUCIÓN DEL EJERCICIO 6

Se trata de definir la función de código siguiendo un orden diferente, y para distinguirla la llamamos dod2(x,y), que

Esta función de código numera la cuadrícula en el orden indicado en la figura, comenzando por **$\mathbf{dod^2(\varepsilon,\varepsilon)=\varepsilon}$.** Se debe especificar ahora cómo se calcula la posición siguiente a cada una de la cuadrícula. Para ello la dividimos en tres zonas: avance en horizontal, cuando i≥j, vertical, cuando i<j, y en espiral, cuando saltamos del borde superior al lateral izquierdo. Estos son los casos generales de la recursión.



Se definiría inductivamente como sigue:

$$\mathrm{dod}^2(\varepsilon,\varepsilon)=\varepsilon$$

$$\mathrm{dod}^2(w_{i+1},\varepsilon)=\mathrm{sig}(\mathrm{dod}^2(\varepsilon,w_i))$$

$$\mathrm{dod}^2(w_i,w_{j+1})=\begin{cases}\mathrm{sig}(\mathrm{dod}^2(w_{i+1},w_j)) & \text{si } w_i<w_j\\ \mathrm{sig}(\mathrm{dod}^2(w_i,w_j)) & \text{si } w_i\geq w_j\end{cases}$$

Es una función de codificación adecuada porque

es biyectiva;

es total: ninguna posición de la tabla queda por llenar;

es inyectiva: no hay dos palabras repetidas en la tabla, puesto que son utilizados en orden a partir de ε;

es sobreyectiva por la misma razón.

Es una función computable.

Si vamos calculando iterativamente los códigos de todos los pares hasta dar con el del buscado, podemos construir el siguiente programa que calcula dicha función:

```
FILA:=  ε; COLUMNA := ε; X0 := ε;
while X1 /= FILA  or X2 /= COLUMNA loop
        if FILA = ε then
                FILA := sig (COLUMNA);
                COLUMNA := ε;
        elsif   FILA > COLUMNA then
                COLUMNA := sig(COLUMNA);
        else    FILA := ant(FILA);
        end if;
        X0 := sig(X0);
end loop;
```

Hoja I.4

SOLUCIÓN DEL EJERCICIO 11

El tipo de datos cola, C, con las siguientes operaciones :

C_vacía?: $C \rightarrow B$ **frente:** $C \rightarrow \sum^*$

encolar: $\sum^* \times C \rightarrow C$ **desencolar:** $C \rightarrow C$

Utilizamos la misma idea que para implementar las pilas, incluyendo sus propias funciones de interpretación y representación. Así, dada cualquier cola $<x_1, x_2, \ldots, x_n]$ (donde x_1 es el elemento situado en el frente) tenemos que:

$$\Re_P(<x_1, x_2, \ldots, x_n]) = \Re_C(<x_1, x_2, \ldots, x_n])$$

Dado que se representan igual, para implementar las operaciones con colas nos será muy útil poder usar directamente las operaciones (ya implementadas) de las pilas.

- C_vacía: $C \rightarrow B$ X0 := P_vacía(X1);
- frente: $C \rightarrow \sum^*$ X0 := cima(X1);
- desencolar: $C \rightarrow C$ X0 := desempilar(X1);
- encolar: $\sum^* \times C \rightarrow C$

```
X0 := empilar (X1, <]);
-- X2 = <z1, ..., zn} ∧ X0 = <X1]
-- la cola X0 será construida como una pila, que tiene en su fondo el elemento encolado
if not P_vacía?(X2) then
      AUX:= X2; PILA := <];
      -- a diferencia de AUX o X0, que son colas tratadas como pilas, PILA es una pila
      -- auxiliar "verdadera" usada para volcar los elementos de X2 en X0
      while not P_vacía?(AUX) loop
            PILA := empilar(cima(AUX), PILA);
            AUX := desempilar(AUX);
      end loop;
      -- PILA = <zn, ..., z1] ∧ X0 = <X1]
      while not P_vacía?(PILA) loop
            X0 := empilar(cima(PILA), X0);
            PILA := desempilar(PILA)
      end loop;
      -- X0 = <z1, ..., zn, X1]
 end if;
```

Hoja I.4

SOLUCIÓN DEL EJERCICIO 13 - Pilas

a) Sea P el programa que resulta de expandir

X0 := empilar('ab', desempilar(X1));

Determina la palabra (no la pila) φ_P(baab)

Primero habremos de determinar la pila correspondiente a **baab** (para lo cuál nos conviene utilizar su representación numérica), después le aplicaremos las operaciones propias del programa P y finalmente devolveremos la pila resultante a su forma original como palabra.

$\Im_P(\mathbf{aab}) = 2*2^3+1*2^2+1*2^1+2*2^0 = 24$

$\mathbf{w_{24}} = cod^2(\mathbf{w_3}, \mathbf{w_3})$

En realidad no necesitamos saber qué hay en el resto de la pila. Dado que hemos de sustituir la cima por **ab** = $\mathbf{w_4}$, la palabra buscada será

$cod^2(\mathbf{w_4}, \mathbf{w_3}) = \mathbf{w_{23}}$ ya que $7*8/2 + 3 = 31$

Si deducimos ahora los símbolos de esta palabra:

Sucesión de cocientes	31	15	7	3	1
Sucesión de "restos"	1	1	1	1	

Con lo que φ_P(**aab**) = **aaaaa**

Hoja I.4 – Ejercicio 13 - Pilas

SOLUCIÓN DEL APARTADO b

```
if X2 = ε then X0 := X1;
else
    X0 := X2;
    -- ahora volcaremos X1, sabiendo que el fondo de X0 ya está ocupado
    if nonem?(X1) then
        -- X1 = <y1, …yn] ∧ X0 = X2 = <z1, …zm] ∧ n,m≥1
        -- El primer elemento a empilar debe ser el del fondo de X1, terminamos
        -- con el de la cima. Utilizamos una variable AUX intermedia para el volcado
        COPIA:= X1;  AUX:= ε;  CONT:= 0;
        while nonem?(COPIA) loop
            AUX:= cod_2(decod_2_1(COPIA), AUX) ;
            CONT:= succ(CONT);
            COPIA:= decod_2_2(COPIA);
        end loop;
        -- AUX = cod^(n+1)(sig(yn), yn-1, …y1, ε) ∧ CONT = n
        -- el contador sirve para no tener que controlar cuándo se acaba AUX
        AUX:= cod_2(ant(decod_2_1(AUX)), decod_2_2(AUX)) ;
        -- AUX = cod^(n+1)(yn, yn-1, …y1, ε)
        while CONT <> 0 loop
            X0:= cod_2(decod_2_1(AUX), X0) ;
            CONT:= pred(CONT);
            AUX:= decod_2_2(AUX);
        end loop;
        -- X0 = <y1, …yn, z1, …zm]
    end if;
end if;
```

Hoja I.4

SOLUCIÓN DEL EJERCICIO 14 - Vectores

a) Sea P el programa que resulta de expandir

X0 := modifica(X1, 2, acceso(X2, 2));

Determina la palabra (no el vector) φ_P(abaababab, bbaaabaa)

El segundo argumento es un vector al que hay que extraer la segunda componente. Empezaremos por determinar qué vector es y qué contiene en dicha componente. Primeramente obtenemos la expresión numérica, más práctica a la hora de decodificar:

$\Im_V$(**bbaaabaa**) = $2*2^7+2*2^6+1*2^5+1*2^4+1*2^3+2*2^2+1*2^1+1*2^0 = 451$

Descomponiendo la palabra en forma de pares codificados averiguamos el tamaño del vector y el contenido de su segunda componente:

$\mathbf{W_{451}}=cod^2(\mathbf{w_{13}},\mathbf{w_{16}})=cod^3(\mathbf{w_{13}},\mathbf{w_4},\mathbf{w_1})=cod^4(\mathbf{w_{13}},\mathbf{w_4},\mathbf{w_1},\mathbf{w_0})=cod^5(\mathbf{w_{13}},\mathbf{w_4},\mathbf{w_1},\boxed{\mathbf{w_0}},\mathbf{w_0})$

Deducimos que es un vector de 14 elementos, de los cuáles los primeros son las palabras $\mathbf{w_4}$, $\mathbf{w_1}$ y $\mathbf{w_0}$. Por tanto, la operación de acceso a la segunda posición nos devuelve la palabra $\mathbf{w_0}$.

El primer argumento también es un vector, que deberemos modificar:

$\Im_V$(**abaababab**) $=1*2^8+ 2*2^7+1*2^6+1*2^5+2*2^4+1*2^3+2*2^2+1*2^1+2*2^0 = 660$

$\mathbf{w_{660}}=cod^2(\mathbf{w_5},\mathbf{w_{30}})=cod^3(\mathbf{w_5},\mathbf{w_5},\mathbf{w_2})=cod^4(\mathbf{w_5},\mathbf{w_5},\mathbf{w_0},\mathbf{w_1})=cod^5(\mathbf{w_5},\mathbf{w_5},\mathbf{w_0},\boxed{\mathbf{w_1}},\mathbf{w_0})$

Deducimos que es un vector de 6 elementos, de los cuáles los primeros son las palabras $\mathbf{w_5}$, $\mathbf{w_0}$ y $\mathbf{w_1}$, y que los cinco restantes están codificados en la palabra $\mathbf{w_0}$. Por tanto, el valor $\mathbf{w_1}$ debe ser sustituido por $\mathbf{w_0}$ para realizar la operación de modificación, reconstruyéndose el vector modificado de la siguiente forma:

$cod^5(\mathbf{w_5},\mathbf{w_5},\mathbf{w_0},\boxed{\mathbf{w_0}},\mathbf{w_0})=cod^4(\mathbf{w_5},\mathbf{w_5},\mathbf{w_0},\mathbf{w_0})=cod^3(\mathbf{w_5},\mathbf{w_5},\mathbf{w_0})= cod^2(\mathbf{w_5},\mathbf{w_{15}})= \mathbf{w_{225}}$

Si deducimos ahora los símbolos de esta palabra

Sucesión de cocientes	225	111	55	27	13	6	2
Sucesión de "restos"	1	2	1	1	1	2	

Con lo que φ_P(**abaababab**, **bbaaabaa**) = **bbaaaba**

Hoja I.4 – Ejercicio 14 - Vectores

SOLUCIÓN DEL APARTADO b

```
N:=decod_2_1(X1); VECTOR:=decod_2_2(X1);
-- X1= (z0, ..., zn) ∧ VECTOR = cod^{n+1}(z0, ..., zn)
if X2>N or X3>N then X0:= ⊥⊥;
elsif X2 = X3 then X0:= X1;
else
-- copiamos los índices X2 y X3 en I y J de forma que I<J
      if  X2 < X3  then  I:= X2;  J:= X3;
      else  I:= X3;  J:= X2;
      end if;
      MEN:= decod(N+1, I+1, VECTOR);
      MAY:= decod(N+1, J+1, VECTOR);
      -- VECTOR = cod^{n+1}(z0, …, zi,…, zj, …, zn) ∧ MEN = zi ∧ MAY = zj
      -- Ahora reconstruimos el vector (de atrás adelante)
      -- Primero los valores entre J y N
      if  J = N  then  NVECT:= MEN;
      else
            NVECT:= decod(N+1, N+1, VECTOR);
            for  K  in reverse  J+1..N-1  loop
                  NVECT := cod_2(decod(N+1, K+1, VECTOR), NVECT);
            end loop;
            NVECT := cod_2(MEN, NVECT);
      end if;
      -- NVECT= cod^{n-j+1}(zi, zj+1, zj+2, …, zn)
      -- Ahora los valores entre I y J-1
      for  K  in reverse  I+1..J-1  loop
            NVECT := cod_2(decod(N+1, K+1, VECTOR), NVECT);
      end loop;
      NVECT := cod_2(MAY, NVECT);
      -- NVECT= cod^{n-i+1}(zj, zi+1, zi+2, …, zj-2, zj-1,zi, zj+1, zj+2, …, zn)
      -- Para terminar los valores entre 0 y I-1
      for  K  in reverse  0..I-1  loop
            NVECT := cod_2(decod(N+1, K+1, VECTOR), NVECT);
      end loop;
      -- NVECT= cod^{n+1}(z0, …, zi-2, zi-1,zj, zi+1, zi+2, …, zj-2, zj-1,zi, zj+1, zj+2, …, zn)
      X0:= cod_2(N, NVECT);
end if;
```