Códigos Convolucionales

Repaso 1- objetivos de la transmisión digital de la información

- Transmisión fiable ➔ control de errores ➔ codificación de canal.
- Transmisión rápida (“ligera”) ➔ compresión ➔ codificación de fuente.
- Transmisión segura ➔ autenticidad, privacidad ➔ criptografía.

Códigos Convolucionales

- Se trata de codificación de canal: añadir redundancia para lograr una trasmisión fiable de la información.
- Los códigos convolucionales son códigos lineales, donde la suma de dos palabras de código cualesquiera también es una palabra de código.
- Se diferencian de los códigos bloque en su forma estructural y las propiedades para corregir errores.
- Tienen memoria: la codificación actual depende de los datos que se envían ahora y que se enviaron en el pasado.

Códigos Convolucionales

- Un código convolucional queda especificado por tres parámetros:
- *Números de entradas k* ← Bits de información
- *Números de salidas n* ← Secuencia codificada de información, bits de información + bits de redundancia.
- *Redundancia (registro) m*
- *Memoria de código m-1*

$$R = \frac{k}{n}$$ Tasa del código

Códigos Convolucionales - filtro FIR

- Un código convolucional es un **filtro FIR binario** (es una convolución…).
- La salida se obtiene filtrando la secuencia de entrada con un filtro FIR binario.

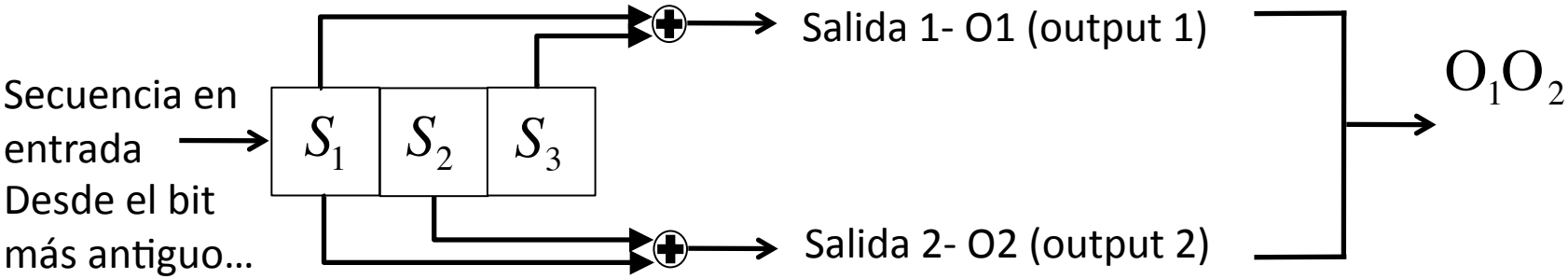
Representación

- Pueden representarse de diferentes formas:
- *Registros de desplazamiento.*
- *Filtro FIR (también hay versiones IIR).*
- *Sistema de n ecuaciones (binarias) con m incógnitas.*
- *Tabla de relación entrada/salida.*
- *Polinomios (parecido a la trasformada Zeta…).*
- *Diagrama de árbol.*
- *Diagrama de estados (maquinas de estados finitos).*
- *Diagrama Trellis.*

Ejemplo con registros de desplazamiento

- Se cosidere un código convolucional

$\boxed{k=1}$ entradas $\boxed{n=2}$ salidas $\boxed{m=3}$ registros $\boxed{m-1=2}$ memoria $\boxed{R=\frac{k}{n}=\frac{1}{2}}$ tasa



- Al principio se considera que

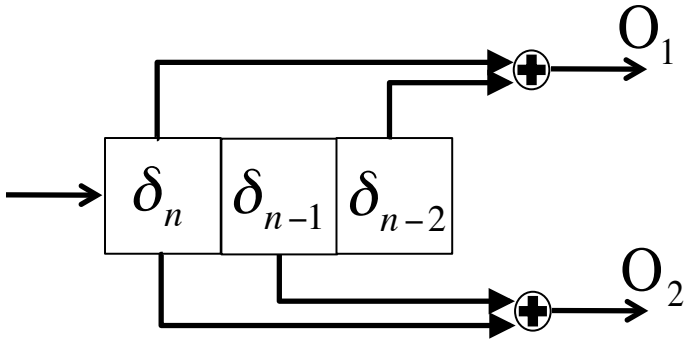
$$S_1 = 0 \quad S_2 = 0 \quad S_3 = 0$$

Ejemplo con registros de desplazamiento

- Los bits ingresan por la izquierda y salen por la derecha, pero los bits de una palabra se leen de izquierda a derecha.

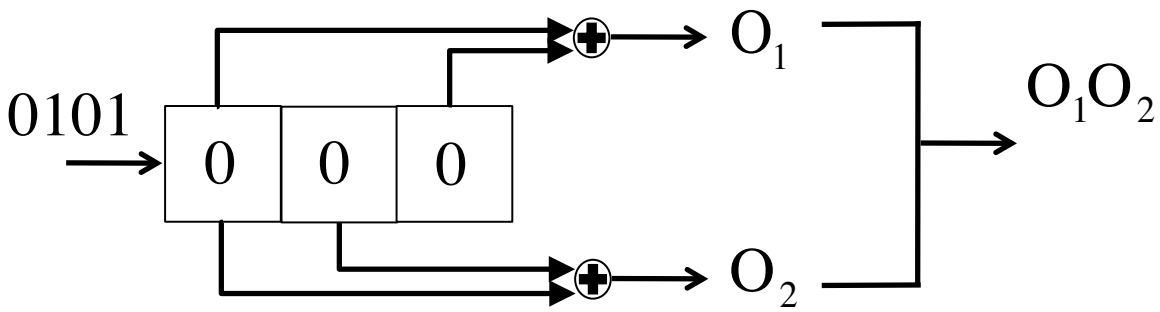
- Observación importante: si queremos codificar 1010 tenemos que dar la vuelta a la secuncia para empezar desde el más antiguo: 0101 (4 bits en entrada)

- Si lo miramos como un filtro, tenemos una memoria de $\boxed{m-1=2}$



Ejemplo con registros de desplazamiento

- Supongamos que queremos enviar y codificar la secuencia 0101

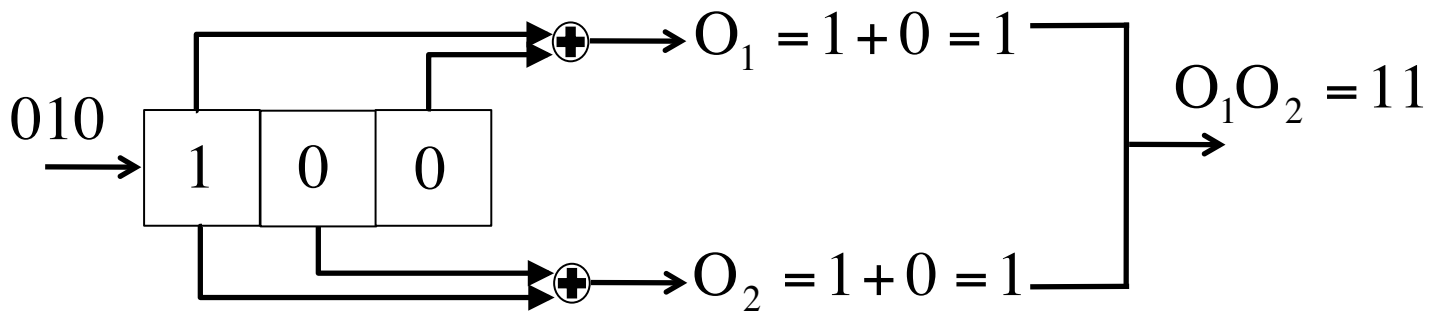


- Recordamos que al principio los registros son nulos

$$S_1 = 0 \quad S_2 = 0 \quad S_3 = 0$$

Ejemplo con registros de desplazamiento

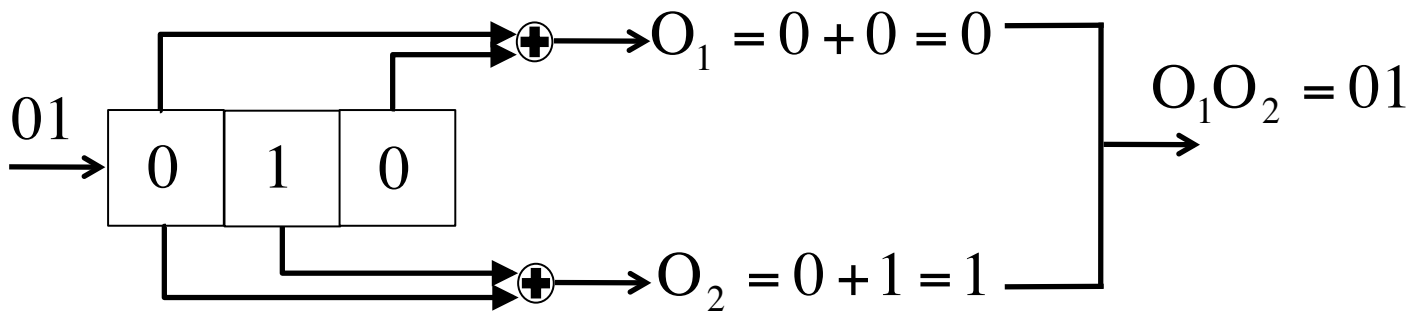
- Entra el primer bit – 1



- producimos 11

Ejemplo con registros de desplazamiento

- Desplazamos el primer bit – 1 y entra el segundo - 0

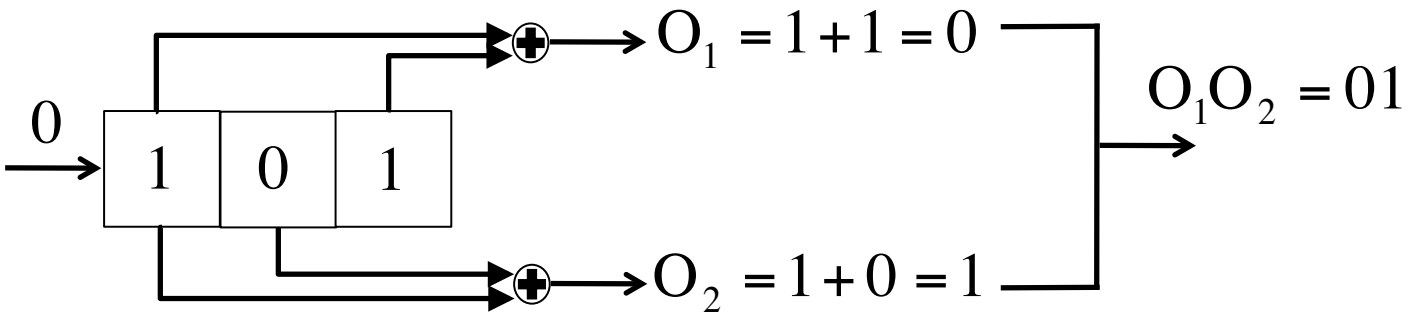


- es decir toda la salida hasta ahora es

$$....01 - 11$$

Ejemplo con registros de desplazamiento

- Desplazamos todo y entra el bit - 1

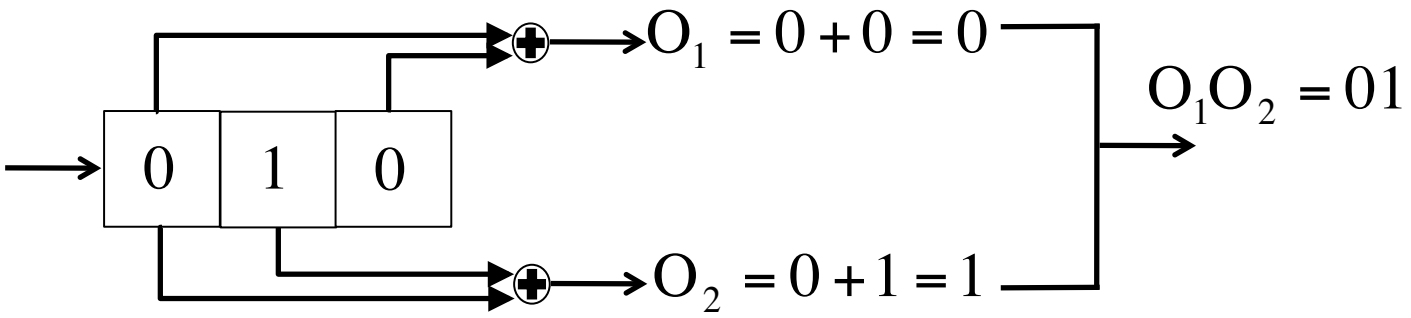


- es decir toda la salida hasta ahora es

$$....01 - 01 - 11$$

Ejemplo con registros de desplazamiento

- Finalmente



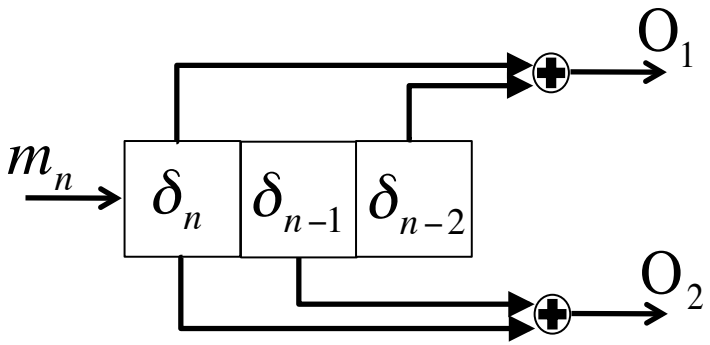
- La secuencia es codificada como (esta es la salida final de 8 bits)

$$01 - 01 - 01 - 11$$

Filtro FIR (en binario)

- Claramente lo que hemos hecho hasta ahora se puede expresar como

$$\begin{cases} O_1 = m_n * (\delta_n + \delta_{n-2}) \\ O_2 = m_n * (\delta_n + \delta_{n-1}) \end{cases}$$



- Se puede hacer versiones recursivas (filtros IIR).

Sistema de ecuaciones (en binario)

- También se puede expresar como

$$\begin{cases} O_1 = S_1 + S_3 \\ O_2 = S_1 + S_2 \end{cases}$$

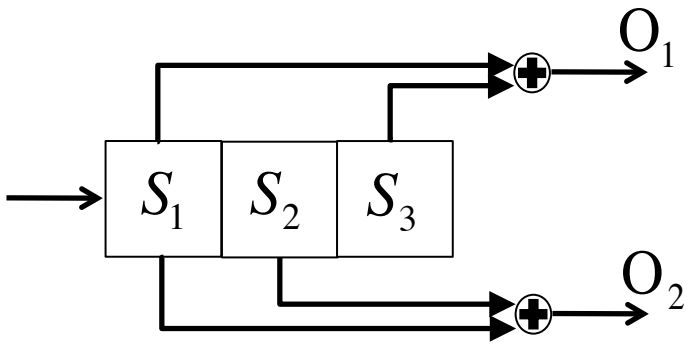


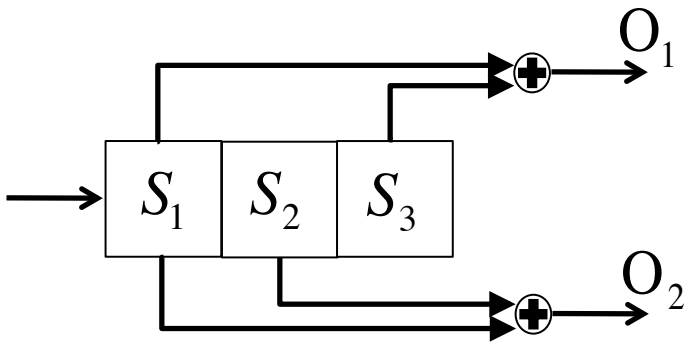
Tabla entradas/salidas

- Otra manera es

$S_1S_2S_3$	O_1O_2
000	00
100	11
010	01
001	10
110	10
011	11
101	01
111	00

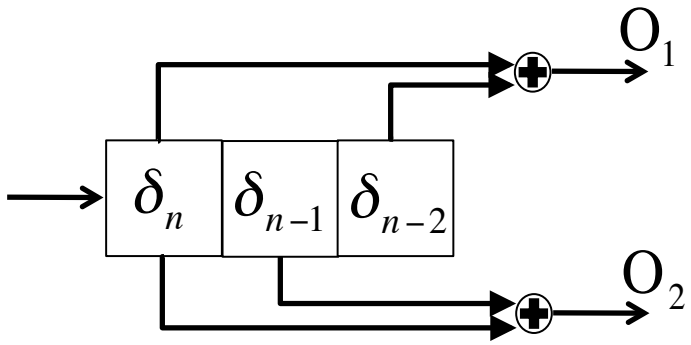
Respuesta impulsiva

$2^m = 2^3 = 8$



Máquina de estados finitos

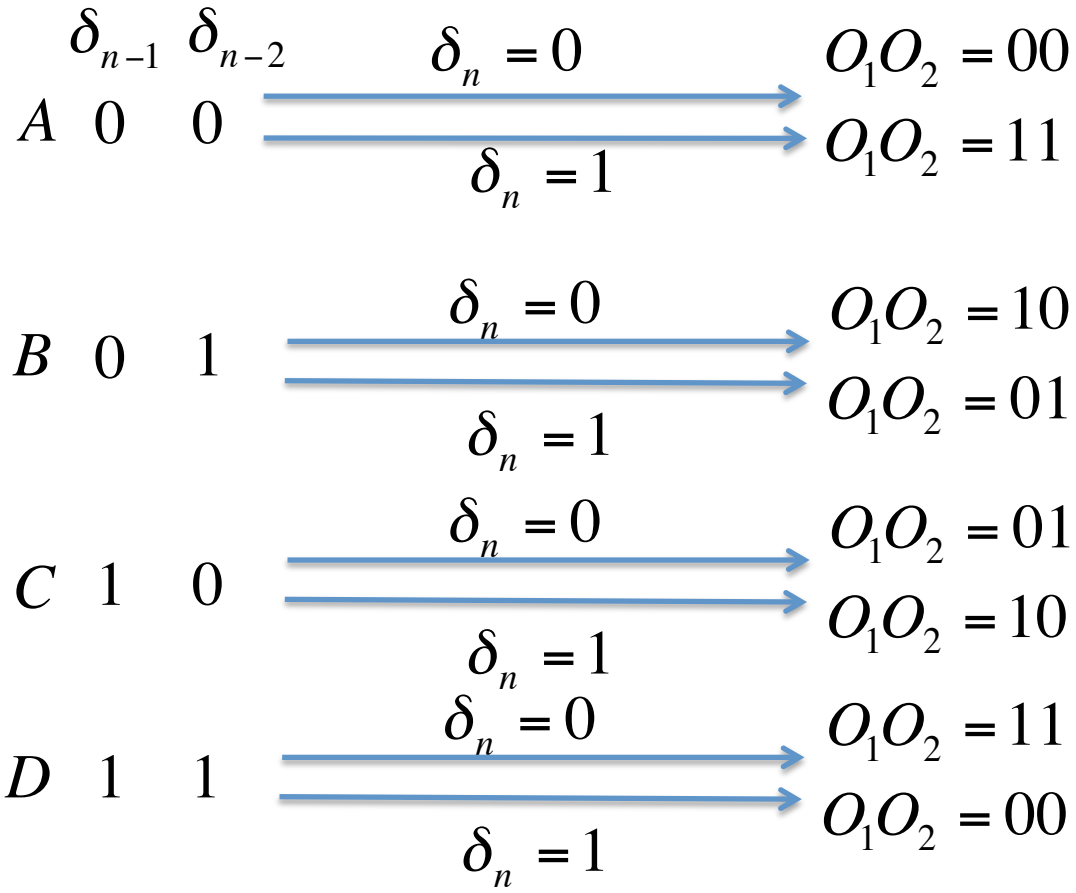
- Tenemos $2^{m-1} = 2^2 = 4$ estados (de la memoria δ_{n-1} y δ_{n-2}).



A	0	0
B	0	1
C	1	0
D	1	1

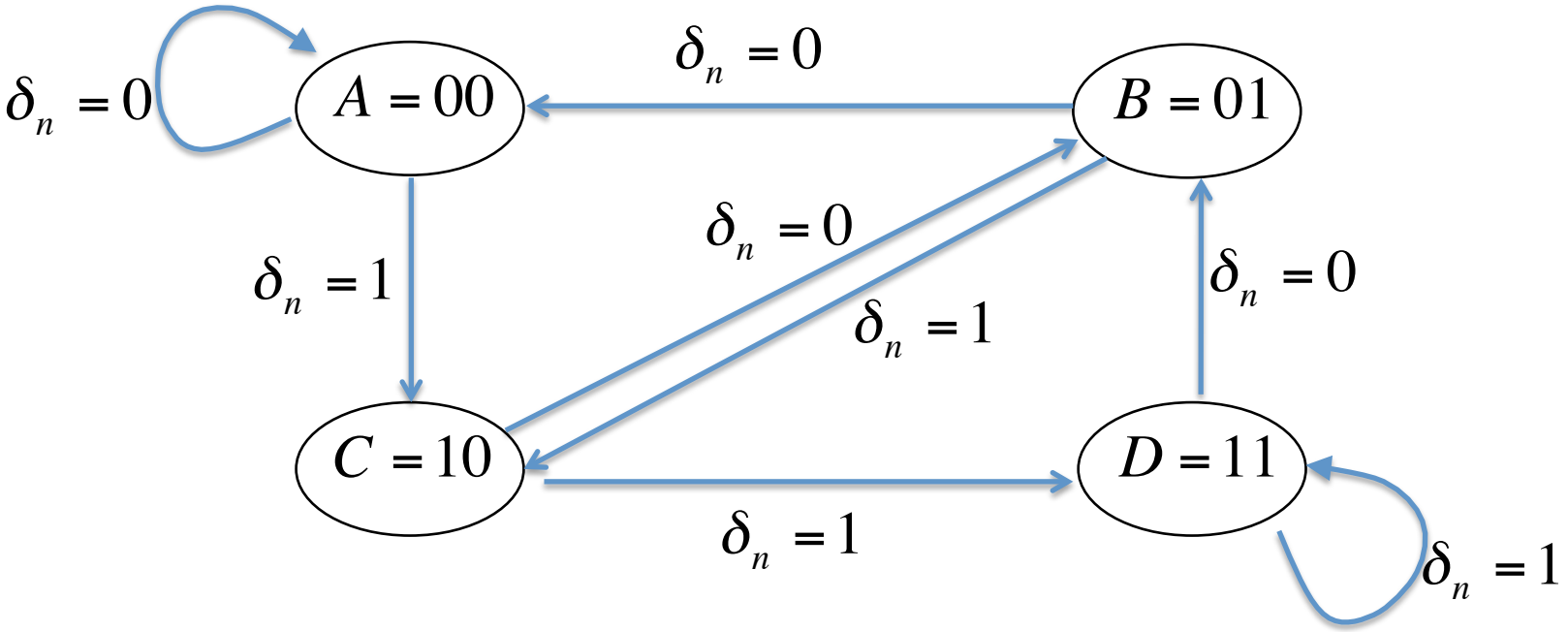
Máquina de estados finitos

- A cada estado está asociado podemos asociar unas salidas según la entrada actual δ_n .



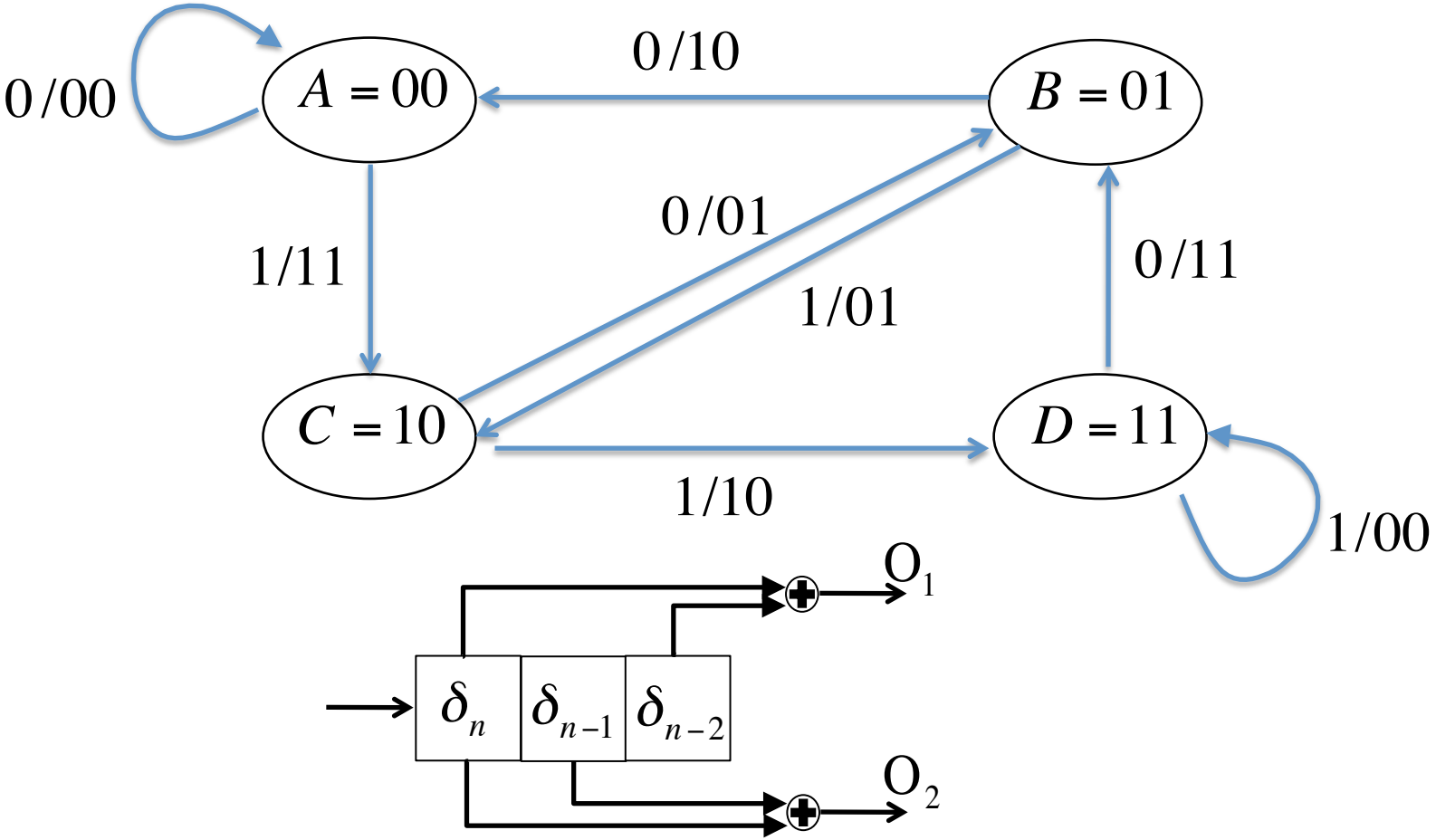
Máquina de estados finitos

- Además según δ_n , al paso siguente tenemos transiciones de estados.



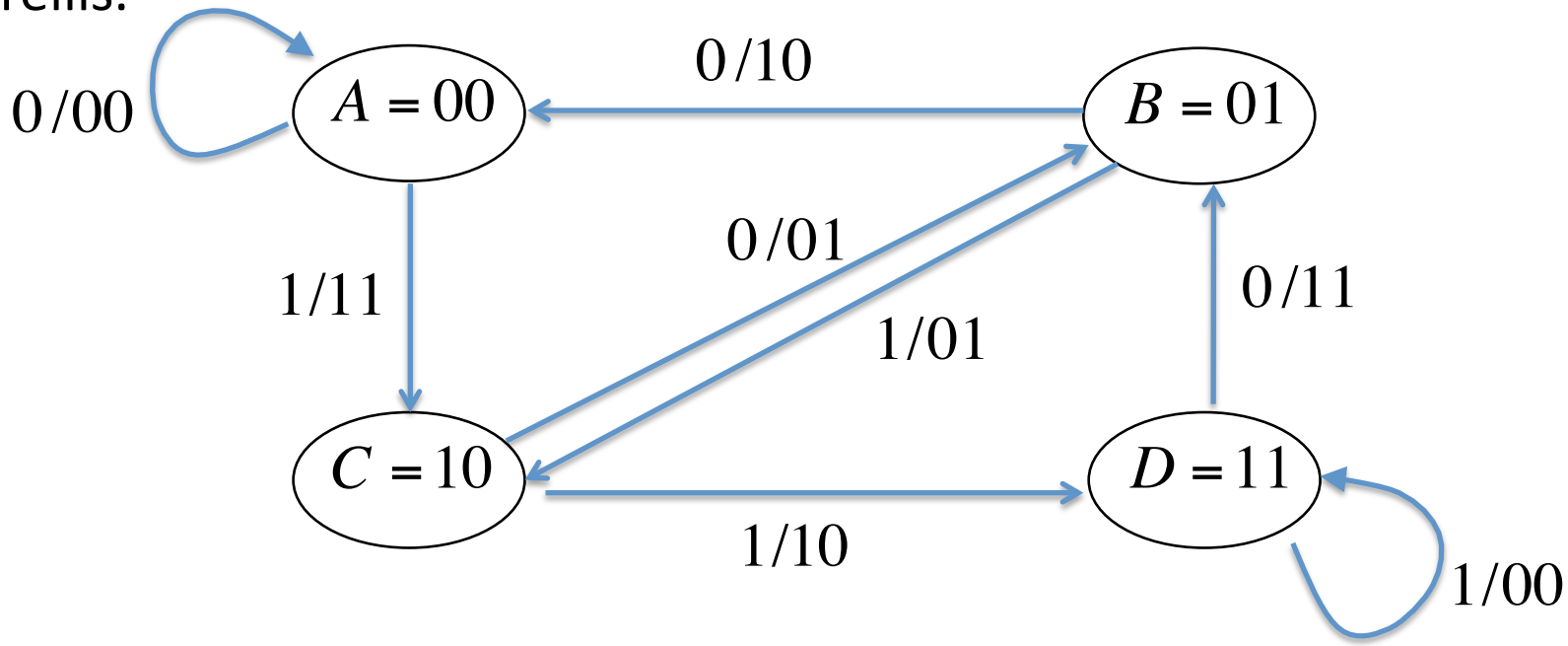
Máquina de estados finitos

- Se suele juntar toda la información en un unico grafo, poniendo también las correspondientes salidas.



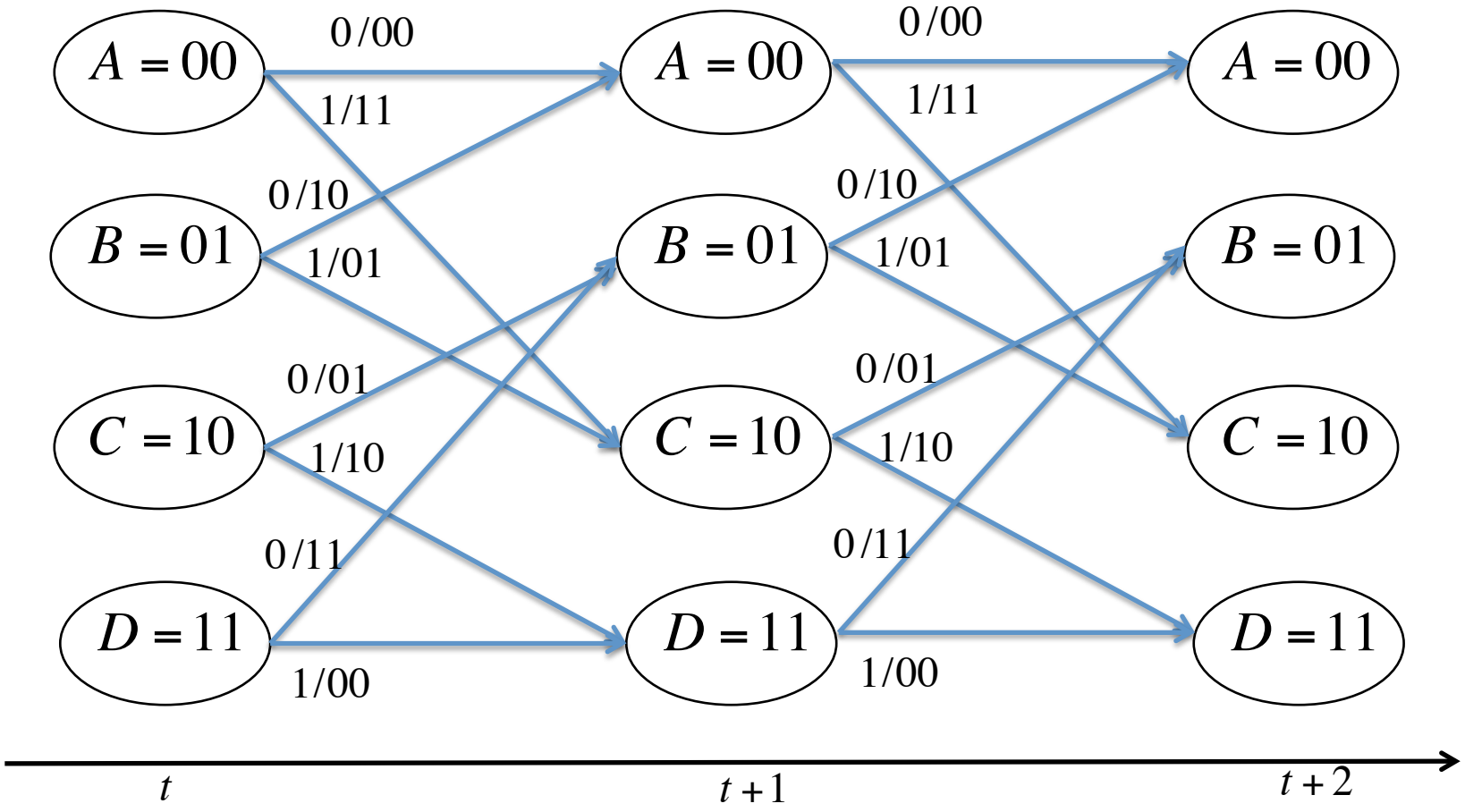
Máquina de estados finitos

- Nota que una *máquina de estados finitos* puede verse caso particular de *Cadena de Markov Ocultas (CMO)* donde las transiciones y la emisiones de las salidas son deterministas (las densidades de probabilidad son deltas).
- De hecho como la CMO pueden representarse también como un Trellis.

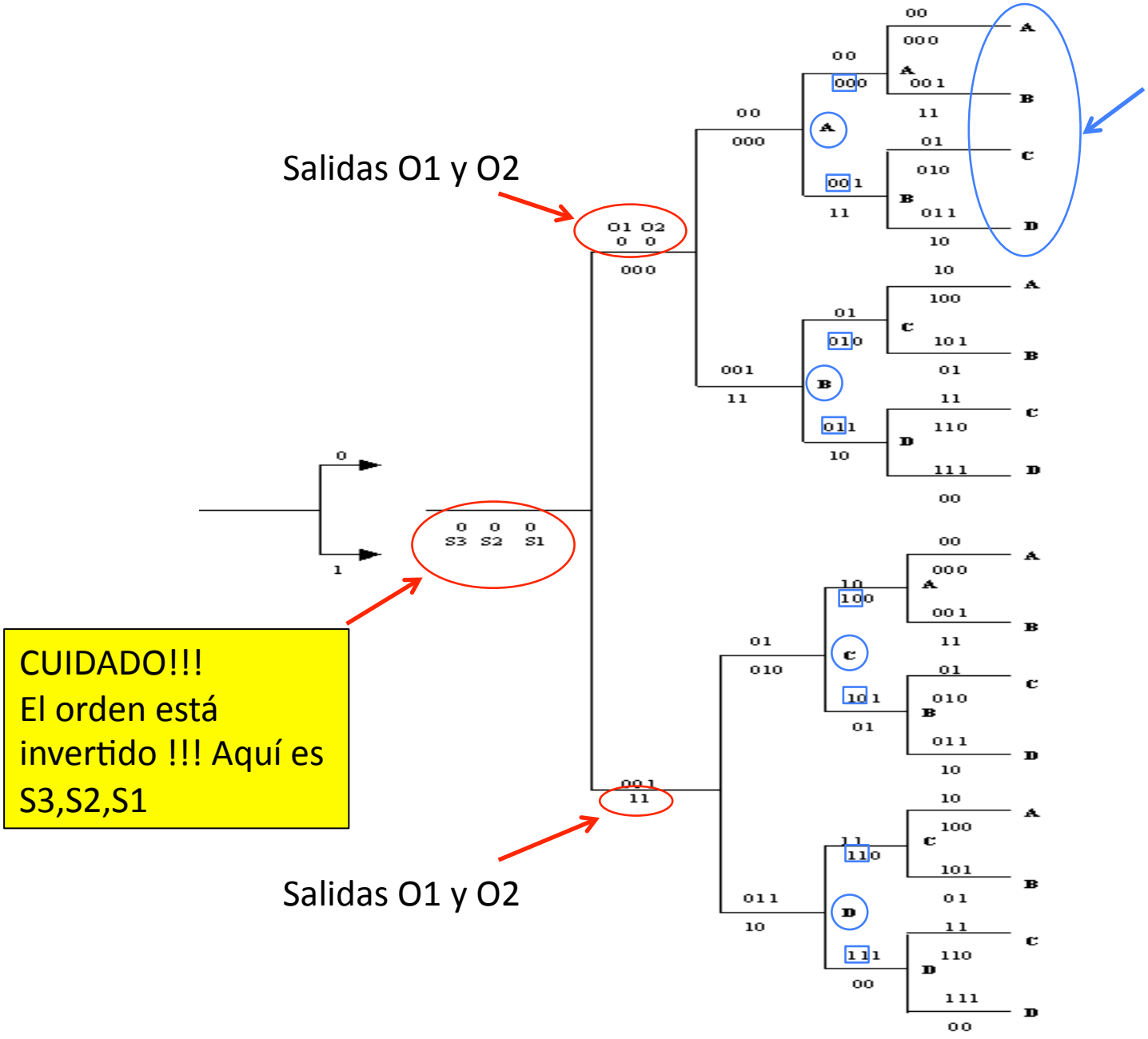


Trellis

- También se representa así



Árbol



$$A = 00$$

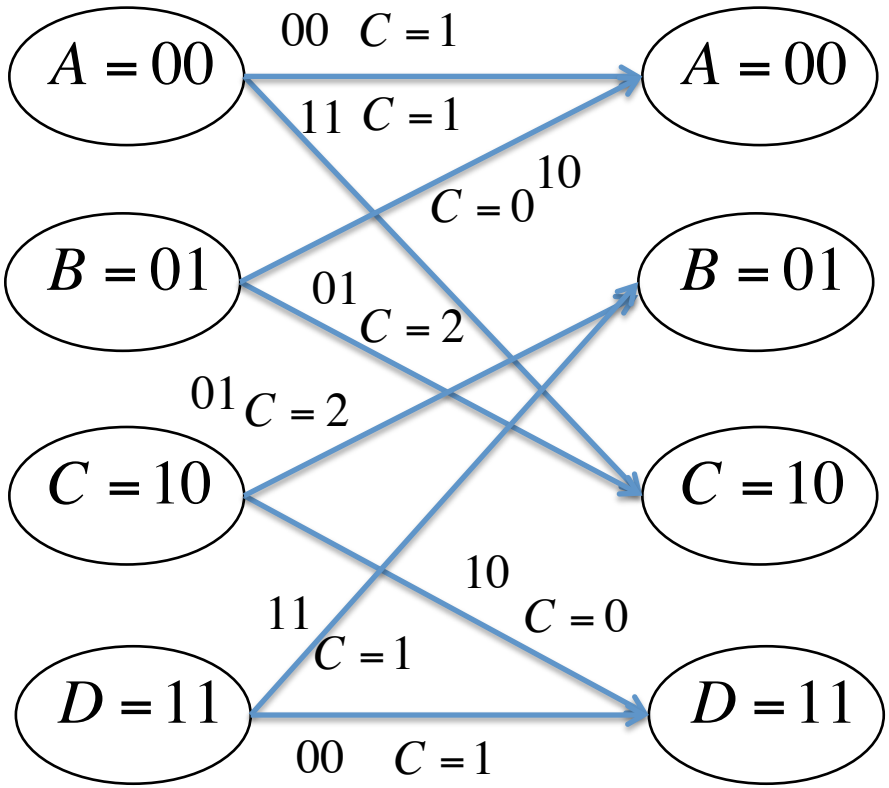
$$B = 01$$

$$C = 10$$

$$D = 11$$

Trellis- Costes distancia de Hamming

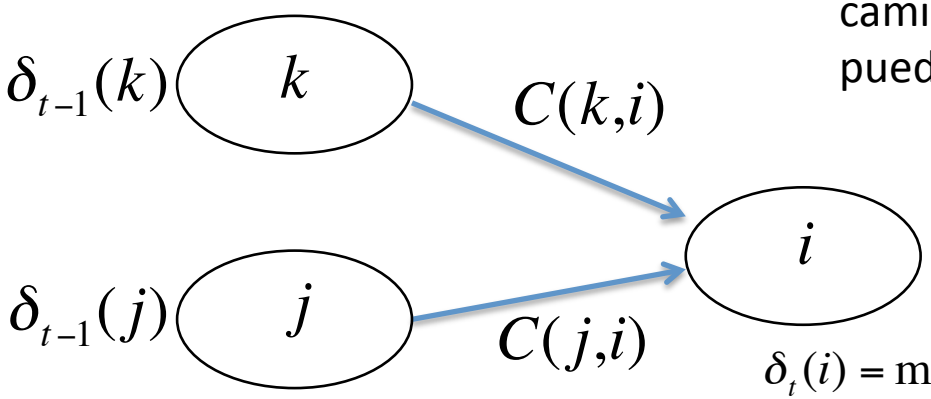
- Si recibimos por ejemplo 10, los coste se calculan como distancia de Hamming



Algoritmo de Viterbi

- Se calculan metricas acumuladas y se guarda el mejor nodo precedente

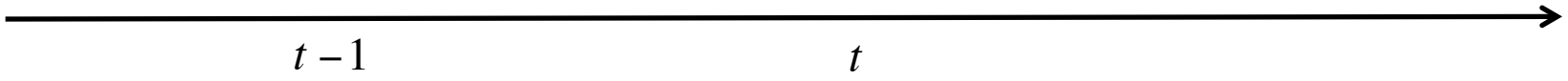
En este ejemplo hay solo 2 caminos… en realidad pueden ser muchos más.



$$\delta_t(i) = \min\left\{\delta_{t-1}(k) + C(k,i); \delta_{t-1}(j) + C(j,i)\right\}$$

$$\psi_t(i) = \arg\min\left\{\delta_{t-1}(k) + C(k,i), \delta_{t-1}(j) + C(j,i)\right\}$$

$\psi_t(i) =$ o k o j en este caso



Algoritmo de Viterbi

- Si los dos caminos (pueden ser más) tiene misma metrica acumulada

$$\delta_{t-1}(k) + C(k,i) = \delta_{t-1}(j) + C(j,i)$$

- Elijo uno de los dos nodos precedentes a caso

$$\psi_t(i) = \text{o } k \text{ o } j \text{ elegido a caso}$$

Algoritmo de Viterbi

- Al principio:

$$\boxed{\delta_0(i) = \text{Costeinicial}(estado = i)}$$

$$\boxed{\psi_0(i) = 0}$$

$$1 \le i \le N$$

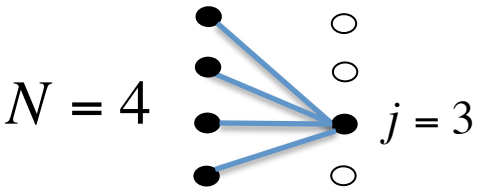
- Iteramos:

$$\boxed{\delta_t(j) = \left(\min_i \left[\delta_{t-1}(i) + C(i,j)\right]\right)}$$

$$1 \le j \le N$$

$$1 \le t \le T$$

$$\boxed{\psi_t(j) = \arg\min_i \left[\delta_{t-1}(i) + C(i,j)\right]}$$



Algoritmo de Viterbi

- luego cuando hemos llegado al final $t = T$, decidimos el estado

$$\boxed{\hat{e}_T = \arg\min_i \left[\delta_T(i) \right]}$$

- Y vamos hacia atrás

$$\boxed{\hat{e}_{T-1} = \psi_T(\hat{e}_T)}$$

$$\boxed{\hat{e}_t = \psi_{t+1}(\hat{e}_{t+1})} \qquad t = T-1, T-2, \ldots, 0$$

- Finalmente tenemos una secuencia de estados

$$\boxed{\hat{e}_T, \hat{e}_{T-1}, \hat{e}_{T-2}, \ldots, \hat{e}_0} \qquad \boxed{\hat{x}_T, \hat{x}_{T-1}, \hat{x}_{T-2}, \ldots, \hat{x}_0}$$

Algoritmo de Viterbi

- Conocida la secuencia de estados estimada

$$\hat{e}_T, \hat{e}_{T-1}, \hat{e}_{T-2}, \ldots, \hat{e}_0$$

$$\hat{e}_0, \hat{e}_1, \hat{e}_2, \ldots, \hat{e}_T$$

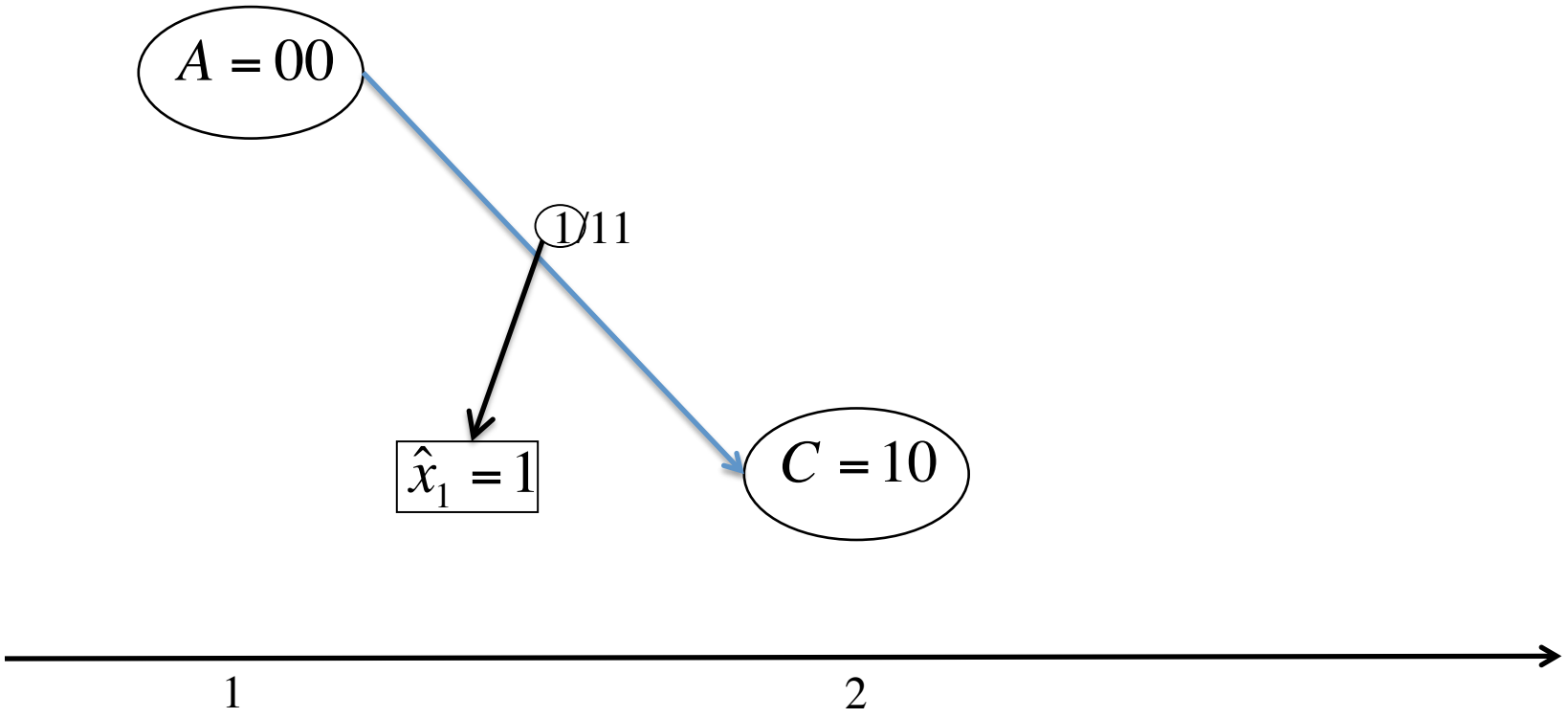
- Es posible encontrar el mensaje que generó dicha secuencia

$$\hat{x}_0, \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_T$$

- Esta palabra código decodificada!!!

Algoritmo de Viterbi

- Por ejemplo



Canal (BSC – Gaussiano)

- Los costes son siempre las log-verosilimilitudes.
- Canal BSC: resultar ser la distancia de Hamming.
- Canal Gaussiano: resultar ser la distancia Euclidea (al cuadrado).

Salida Blanda-Dura

- El algoritmo de Viterbi proporciona una *salida dura* (decidimos directamente la palabra código que con más probabilidad ha sido enviado) en decodificación. Esto es porque el algoritmo de Viterbi es un algoritmo de optimización (sirve a minimizar o maximizar costes o probabilidades).

- Para lograr una *salida blanda* (es decir, hallar todas las probabilidades o costes) hay que utilizar otro algoritmos (por el ejemplo el BCJR, que sirven para calcular de modo eficaz costes o probabilidades). Dado que se calculan todos los costes/probabilidades (de cada bit) luego se puede minimizar/ maximizar y decidir (es decir, hallar una salida dura).

Curvas BER para la practica

- SIN codificación: DADA una SNR se envían N bits (de mensaje) se pasan por el canal. Se reciben N bits, y se ve cuanto errores hay en recepción (bit por bit). Para calcular la BER se repite varias veces y se promedia. Todo tiene que repetirse por diferente valores de SNR.

- CON codificación: DADA una SNR se codifican N bits, se envían N+R, que pasarán por el canal. Se decodifican los N+R bits, así que tenemos otra vez N bits. Se calculan los errores (para cada bit) respecto a los bits enviados. Se repite todo varias veces y se promedia. Todo tiene que repetirse por diferente valores de SNR.

Curvas BER para la practica

- Realmente, con codificación, habría que tenerse en cuenta que utilizamos el canal más veces (gastamos más potencia), y esto afectaría al SNR y a la curva BER (que empeoraría). Pero para esta practica podéis no tenerlo en cuenta.