Théorie de l'information
Algorithme de Viterbi

Julien Lavauzelle

Université Paris 8

Master 1 Mathématiques et applications — parcours ACC

13/11/2023

Plan

Code convolutif

Un **code convolutif** est la donnée de k polynômes $g^{(1)}(x), \dots, g^{(k)}(x) \in \mathbb{F}_2[x]$, de degré $\leq r-1$ et de terme constant 1. On note :

$$g^{(i)}(x) = \sum_{j=0}^{r-1} g_j^{(i)} x^j$$

et on peut associer à $g^{(i)}(x)$ la liste de ses coefficients $\boldsymbol{g}^{(i)} = (g_0^{(i)}, \dots, g_{r-1}^{(i)})$.

Un code convolutif permet d'encoder des messages binaires de **taille arbitraire**.

L'**encodage** d'un message $\boldsymbol{m} = (m_0, \dots, m_{n-1}) \in \{0,1\}^n$ produit k mots de codes $\boldsymbol{c} = (\boldsymbol{c}^{(1)}, \dots, \boldsymbol{c}^{(k)})$, de sorte que

$$c_j^{(i)} := \sum_{\ell=0}^{r-1} m_{j-\ell} g_\ell^{(i)}, \quad \forall j \geq 0$$

avec comme convention $m_\ell = 0$ si $\ell < 0$.

Remarque. Si le message est de longueur n, alors les mots de code en sortie sont nuls à partir de l'indice $n+r$. On peut donc considérer qu'ils ont longueur $n+r-1$.

Exemple

Par exemple, prenons

$$\begin{array}{rcccccc} g^{(1)}(x) & = & 1 & + & x & + & x^2 \\ g^{(2)}(x) & = & 1 & & & + & x^2 \end{array}$$

Alors on obtient les relations d'encodage :

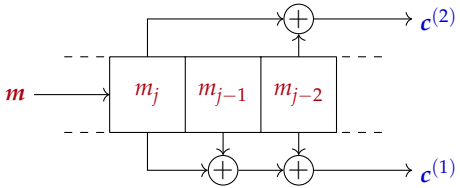
$$\begin{array}{rcccccll} c_j^{(1)} & = & m_j & + & m_{j-1} & + & m_{j-2} & \mod 2 \\ c_j^{(2)} & = & m_j & & & + & m_{j-2} & \mod 2 \end{array}$$

Si le message à encoder est $\boldsymbol{m} = (1,0,1,1)$, on obtient

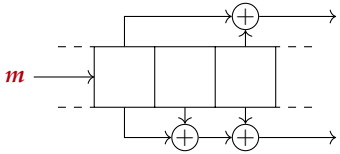
$$\begin{array}{ll} \boldsymbol{c}^{(1)} & = (1,1,0,0,0,1) \\ \boldsymbol{c}^{(2)} & = (1,0,0,1,1,1) \end{array}$$

L'encodage peut être représenté par le passage dans un registre à décalage.

$$c_j^{(1)} = m_j + m_{j-1} + m_{j-2}$$
$$c_j^{(2)} = m_j \qquad + m_{j-2}$$

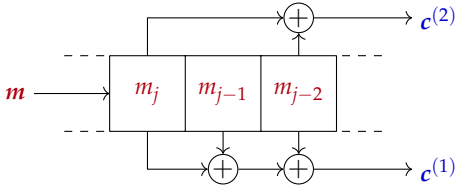


Les différentes étapes pour le message $\boldsymbol{m} = (m_0, m_1, m_2, m_3) = (1, 0, 1, 1)$:

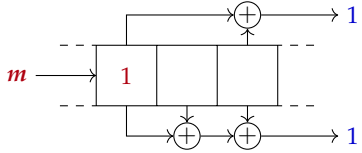


L'encodage peut être représenté par le passage dans un registre à décalage.

$$c_j^{(1)} = m_j + m_{j-1} + m_{j-2}$$
$$c_j^{(2)} = m_j \qquad + m_{j-2}$$

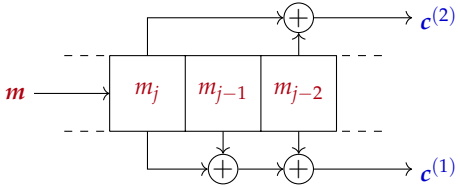


Les différentes étapes pour le message $\boldsymbol{m} = (m_0, m_1, m_2, m_3) = (1, 0, 1, 1)$:

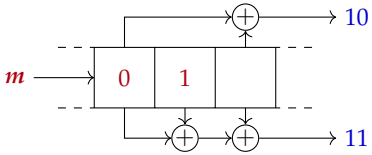


L'encodage peut être représenté par le passage dans un registre à décalage.

$$c_j^{(1)} = m_j + m_{j-1} + m_{j-2}$$
$$c_j^{(2)} = m_j \qquad + m_{j-2}$$



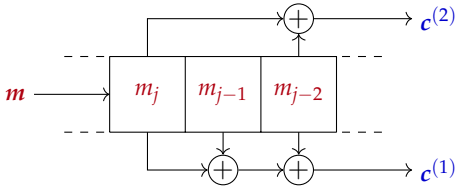
Les différentes étapes pour le message $\boldsymbol{m} = (m_0, m_1, m_2, m_3) = (1, 0, 1, 1)$:



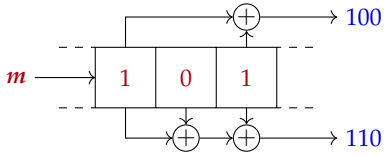
Représentation par registre à décalage

L'encodage peut être représenté par le passage dans un registre à décalage.

$$c_j^{(1)} = m_j + m_{j-1} + m_{j-2}$$
$$c_j^{(2)} = m_j \qquad + m_{j-2}$$

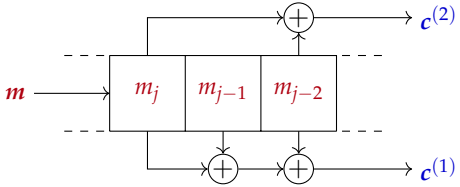


Les différentes étapes pour le message $\boldsymbol{m} = (m_0, m_1, m_2, m_3) = (1, 0, 1, 1)$:

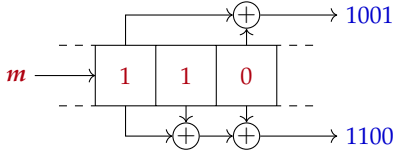


L'encodage peut être représenté par le passage dans un registre à décalage.

$$c_j^{(1)} = m_j + m_{j-1} + m_{j-2}$$
$$c_j^{(2)} = m_j \qquad + m_{j-2}$$



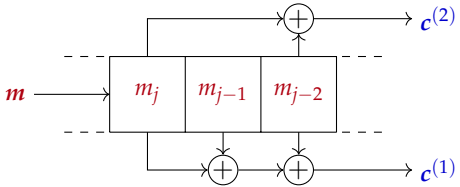
Les différentes étapes pour le message $\boldsymbol{m} = (m_0, m_1, m_2, m_3) = (1, 0, 1, 1)$:



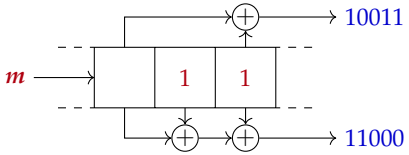
Représentation par registre à décalage

L'encodage peut être représenté par le passage dans un registre à décalage.

$$c_j^{(1)} = m_j + m_{j-1} + m_{j-2}$$
$$c_j^{(2)} = m_j \qquad + m_{j-2}$$



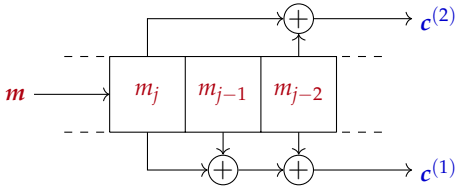
Les différentes étapes pour le message $\boldsymbol{m} = (m_0, m_1, m_2, m_3) = (1, 0, 1, 1)$:



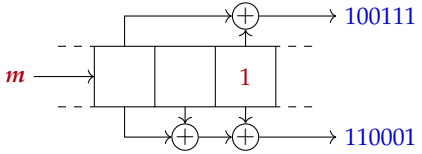
Représentation par registre à décalage

L'encodage peut être représenté par le passage dans un registre à décalage.

$$c_j^{(1)} = m_j + m_{j-1} + m_{j-2}$$
$$c_j^{(2)} = m_j \qquad + m_{j-2}$$



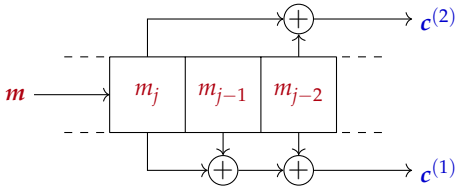
Les différentes étapes pour le message $\boldsymbol{m} = (m_0, m_1, m_2, m_3) = (1, 0, 1, 1)$:



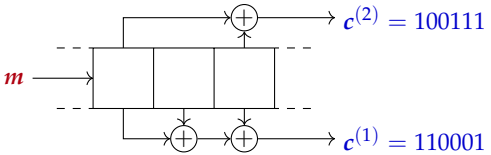
Représentation par registre à décalage

L'encodage peut être représenté par le passage dans un registre à décalage.

$$c_j^{(1)} = m_j + m_{j-1} + m_{j-2}$$
$$c_j^{(2)} = m_j \qquad + m_{j-2}$$



Les différentes étapes pour le message $\boldsymbol{m} = (m_0, m_1, m_2, m_3) = (1, 0, 1, 1)$:



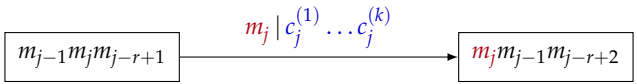
Lors de l'encodage, le registre passe par différents états, selon les bits de message qui y sont insérés.

Dans la **représentation par treillis**, on définit un **état** comme une valeur possible des $r-1$ premiers bits du registre. Il y a donc 2^{r-1} états différents.

Une étape de l'encodage :

- correspond à une **transition** entre deux états : si l'on insère un bit $b = m_j$ en début de registre, on déplace les $r-1$ premiers bits de registre ;
- produit k bits de mots de codes $c_j^{(1)}, \ldots, c_j^{(k)}$.

On représente ces deux actions par une arête entre deux nœuds d'un graphe, appelé **treillis** :



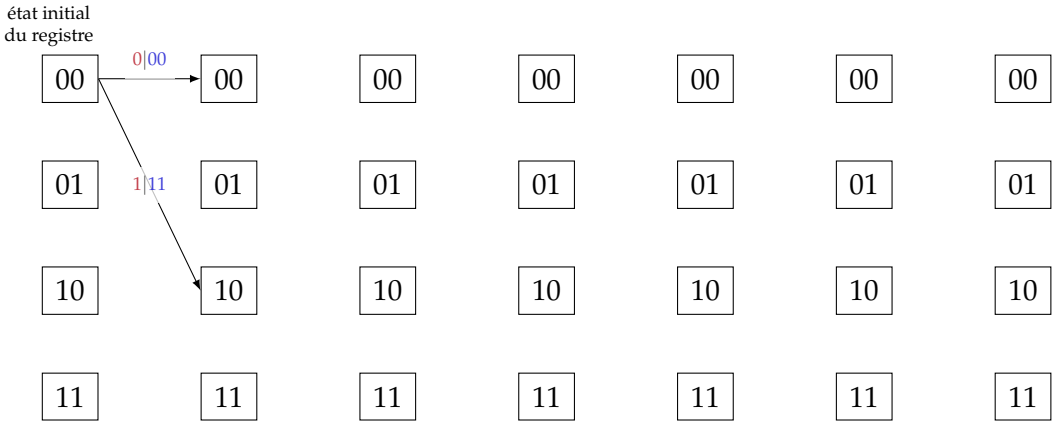
Représentation par treillis : exemple

Puis, on représente **tous** les états et **toutes** les transitions possibles lors d'un encodage.

Représentation par treillis : exemple

Puis, on représente **tous** les états et **toutes** les transitions possibles lors d'un encodage.

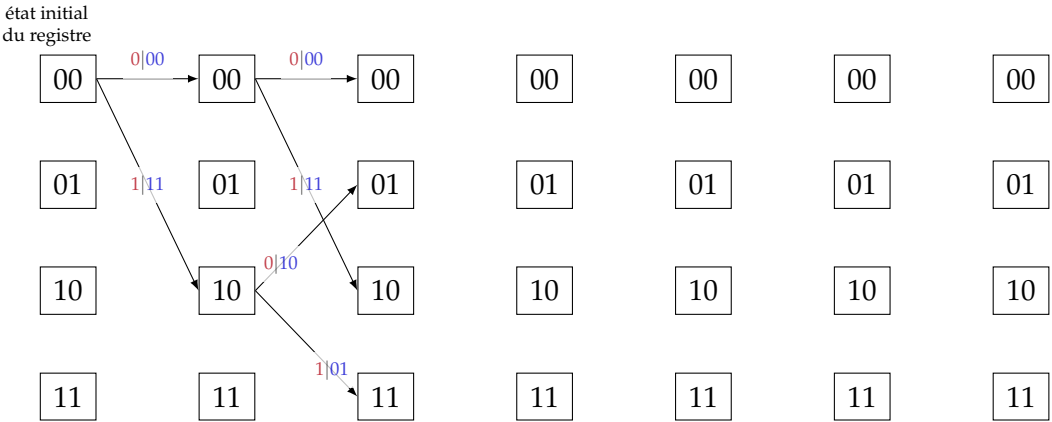
Pour l'exemple précédent ($g(1)(x) = 1 + x + x^2$ et $g^{(2)}(x) = 1 + x^2$) :



Représentation par treillis : exemple

Puis, on représente **tous** les états et **toutes** les transitions possibles lors d'un encodage.

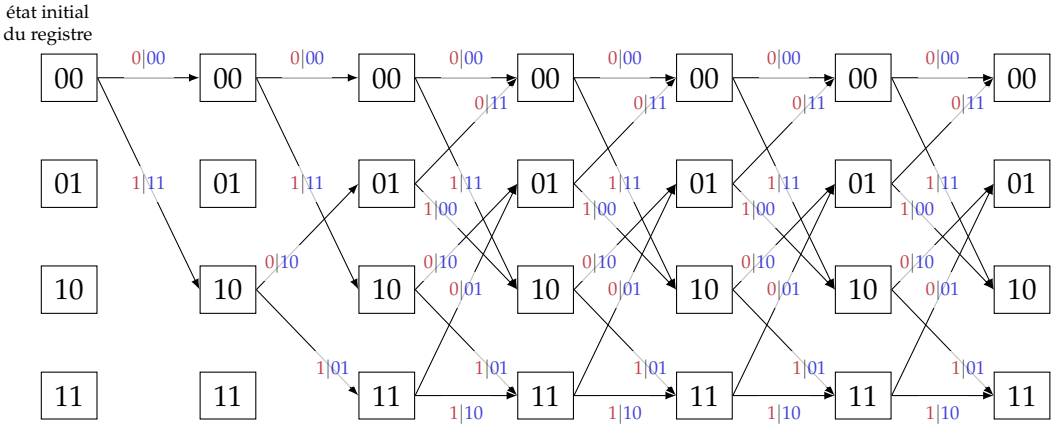
Pour l'exemple précédent ($g(1)(x) = 1 + x + x^2$ et $g^{(2)}(x) = 1 + x^2$) :



Représentation par treillis : exemple

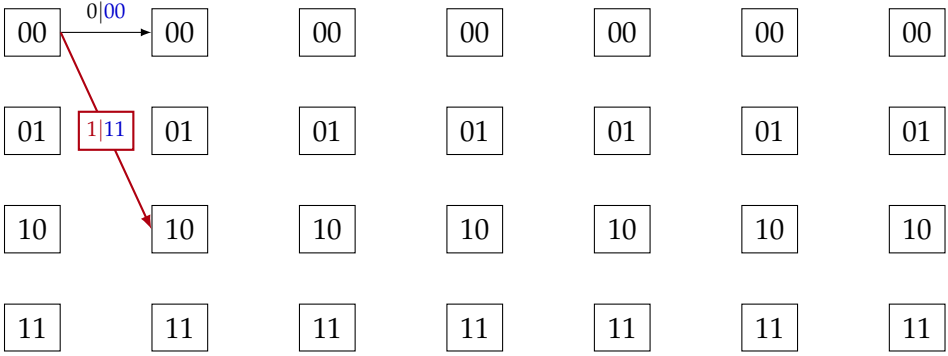
Puis, on représente **tous** les états et **toutes** les transitions possibles lors d'un encodage.

Pour l'exemple précédent ($g(1)(x) = 1 + x + x^2$ et $g^{(2)}(x) = 1 + x^2$) :



Représentation par treillis : exemple

L'encodage correspond alors à suivre un **chemin** dans ce treillis : on suit les arêtes indexées par les bits de message.



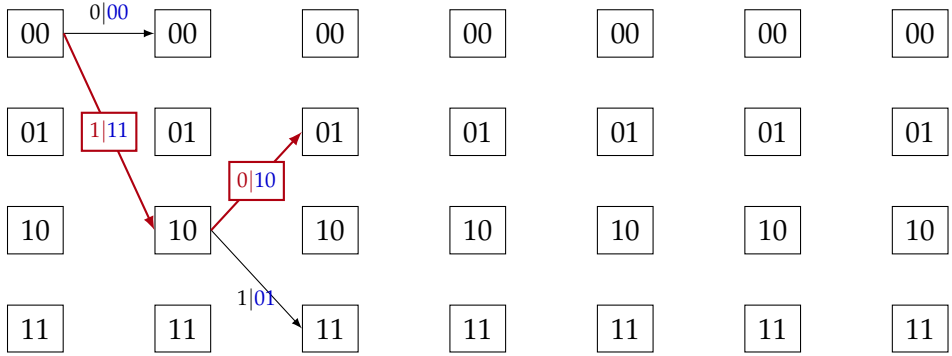
$m_0 = 1$

$\boldsymbol{c}^{(1)} = \quad 1$

$\boldsymbol{c}^{(2)} = \quad 1$

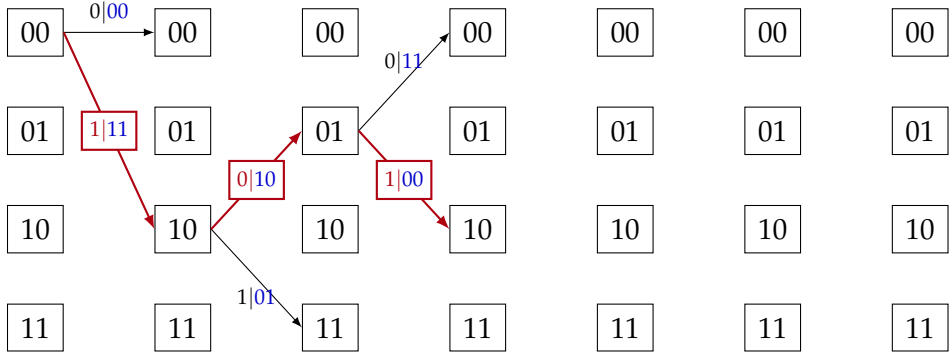
Représentation par treillis : exemple

L'encodage correspond alors à suivre un **chemin** dans ce treillis : on suit les arêtes indexées par les bits de message.



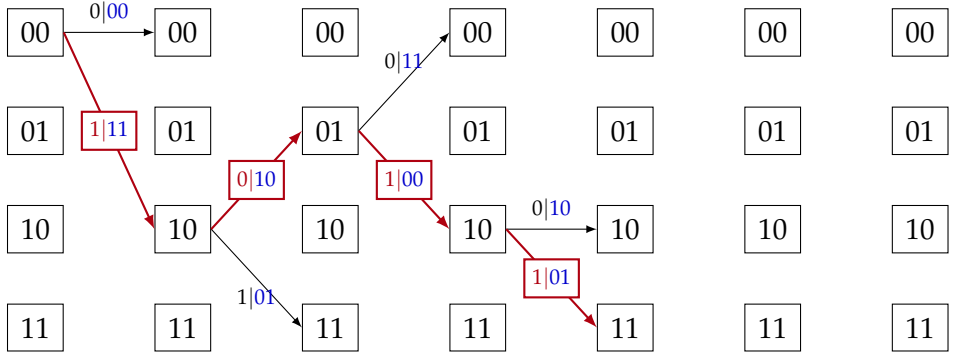
Représentation par treillis : exemple

L'encodage correspond alors à suivre un **chemin** dans ce treillis : on suit les arêtes indexées par les bits de message.

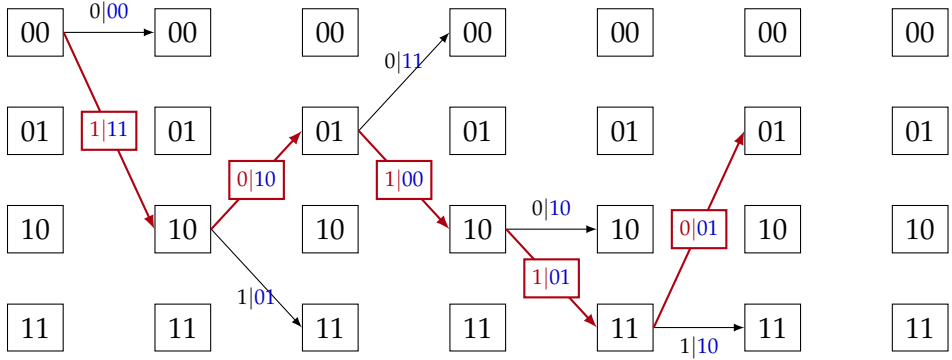


Représentation par treillis : exemple

L'encodage correspond alors à suivre un **chemin** dans ce treillis : on suit les arêtes indexées par les bits de message.

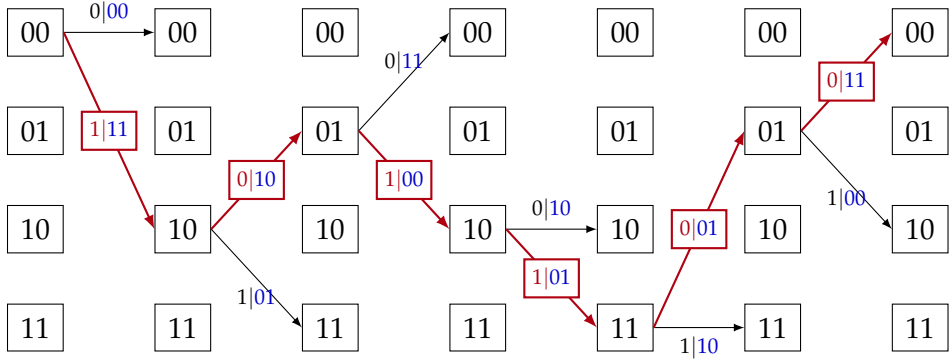


L'encodage correspond alors à suivre un **chemin** dans ce treillis : on suit les arêtes indexées par les bits de message.



Représentation par treillis : exemple

L'encodage correspond alors à suivre un **chemin** dans ce treillis : on suit les arêtes indexées par les bits de message.



Plan

Erreurs de transmission. Lors de la transmission dans le canal des mots de codes $\boldsymbol{c} = (\boldsymbol{c}^{(1)}, \dots, \boldsymbol{c}^{(k)})$ associés à un message $\boldsymbol{m}$, des erreurs peuvent survenir. On obtient alors des mots bruités $\boldsymbol{y} = (\boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(k)})$.

Distance de Hamming. On note

$$\mathrm{d}(\boldsymbol{y}^{(i)}, \boldsymbol{c}^{(i)}) = |\{j \mid y_j^{(i)} \neq c_j^{(i)}\}|$$

la distance de Hamming entre les mots $\boldsymbol{y}^{(i)}$ et $\boldsymbol{c}^{(i)}$.

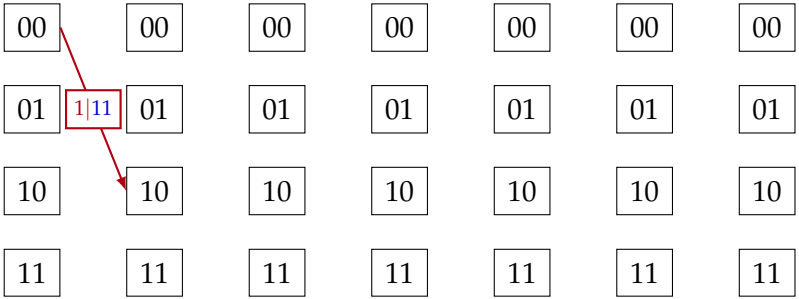
Le poids total de l'erreur est donc $e = \sum_{i=1}^{k} \mathrm{d}(\boldsymbol{y}^{(i)}, \boldsymbol{c}^{(i)})$.

Le but d'un **algorithme de décodage** est de retrouver le message $\boldsymbol{m}$ qui a donné l'encodage $\boldsymbol{c}$, à partir de sa version bruitée $\boldsymbol{y}$.

Remarque. Cela sera possible si le nombre d'erreur e est petit.

Erreurs dans la représentation par treillis

Dans le treillis, on peut calculer la distance de Hamming d'un encodage $\boldsymbol{c}$ à des mots bruités $\boldsymbol{y}$, en sommant à chaque étape le nombre de bits de sortie $c_j^{(1)}, \ldots, c_j^{(k)}$ différents de $y_j^{(1)}, \ldots, y_j^{(k)}$.



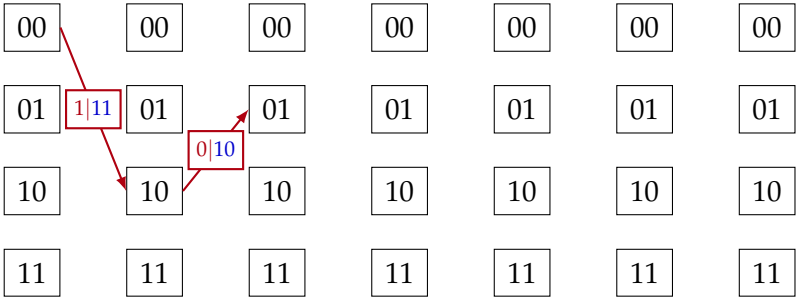
$\boldsymbol{y}^{(1)} = 1$
$\boldsymbol{y}^{(2)} = 1$

$\boldsymbol{c}^{(1)} = 1$
$\boldsymbol{c}^{(2)} = 1$

erreur $e =$ 0 +

Erreurs dans la représentation par treillis

Dans le treillis, on peut calculer la distance de Hamming d'un encodage $\boldsymbol{c}$ à des mots bruités $\boldsymbol{y}$, en sommant à chaque étape le nombre de bits de sortie $c_j^{(1)}, \ldots, c_j^{(k)}$ différents de $y_j^{(1)}, \ldots, y_j^{(k)}$.



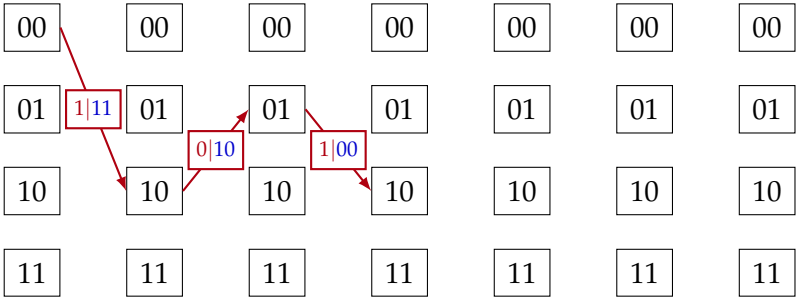
$\boldsymbol{y}^{(1)} = 1 \quad 1$
$\boldsymbol{y}^{(2)} = 1 \quad 1$

$\boldsymbol{c}^{(1)} = 1 \quad 1$
$\boldsymbol{c}^{(2)} = 1 \quad 0$

erreur $e = 0 + 1 +$

Erreurs dans la représentation par treillis

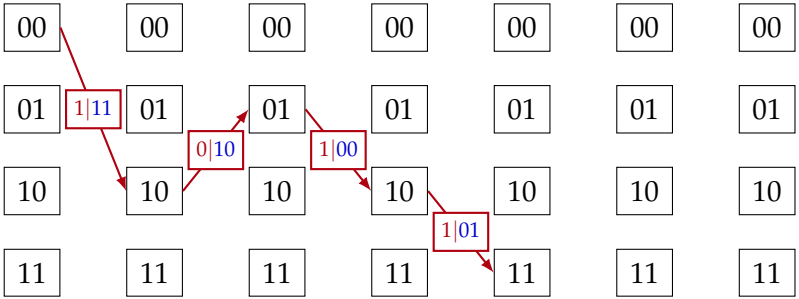
Dans le treillis, on peut calculer la distance de Hamming d'un encodage $\boldsymbol{c}$ à des mots bruités $\boldsymbol{y}$, en sommant à chaque étape le nombre de bits de sortie $c_j^{(1)}, \ldots, c_j^{(k)}$ différents de $y_j^{(1)}, \ldots, y_j^{(k)}$.



$$
\begin{array}{lccccccc}
\boldsymbol{y}^{(1)} = & 1 & & 1 & & 0 & & \\
\boldsymbol{y}^{(2)} = & 1 & & 1 & & 0 & & \\
 & & & & & & & \\
\boldsymbol{c}^{(1)} = & 1 & & 1 & & 0 & & \\
\boldsymbol{c}^{(2)} = & 1 & & 0 & & 0 & & \\
 & & & & & & & \\
\text{erreur } e = & 0 & + & 1 & + & 0 & + &
\end{array}
$$

Erreurs dans la représentation par treillis

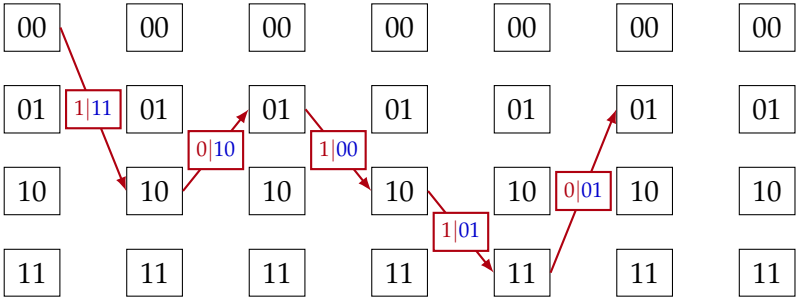
Dans le treillis, on peut calculer la distance de Hamming d'un encodage $\boldsymbol{c}$ à des mots bruités $\boldsymbol{y}$, en sommant à chaque étape le nombre de bits de sortie $c_j^{(1)}, \ldots, c_j^{(k)}$ différents de $y_j^{(1)}, \ldots, y_j^{(k)}$.



$$
\begin{array}{lccccccccc}
\boldsymbol{y}^{(1)} = & 1 & & 1 & & 0 & & 1 & & \\
\boldsymbol{y}^{(2)} = & 1 & & 1 & & 0 & & 1 & & \\
\boldsymbol{c}^{(1)} = & 1 & & 1 & & 0 & & 0 & & \\
\boldsymbol{c}^{(2)} = & 1 & & 0 & & 0 & & 1 & & \\
\text{erreur } e = & 0 & + & 1 & + & 0 & + & 1 & + &
\end{array}
$$

Erreurs dans la représentation par treillis

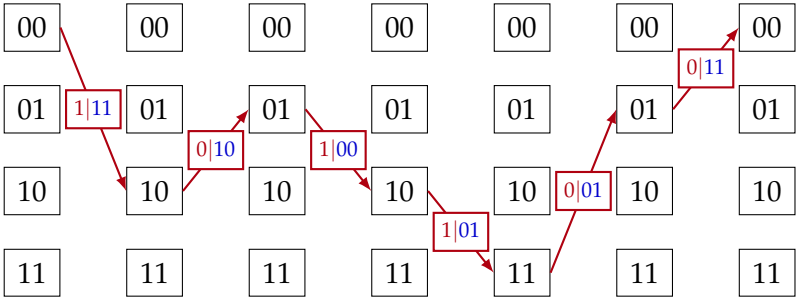
Dans le treillis, on peut calculer la distance de Hamming d'un encodage $\boldsymbol{c}$ à des mots bruités $\boldsymbol{y}$, en sommant à chaque étape le nombre de bits de sortie $c_j^{(1)}, \ldots, c_j^{(k)}$ différents de $y_j^{(1)}, \ldots, y_j^{(k)}$.



$\boldsymbol{y}^{(1)} =$	1		1		0		1		0
$\boldsymbol{y}^{(2)} =$	1		1		0		1		1
$\boldsymbol{c}^{(1)} =$	1		1		0		0		0
$\boldsymbol{c}^{(2)} =$	1		0		0		1		1
erreur $e =$	0	+	1	+	0	+	1	+	0	+

Erreurs dans la représentation par treillis

Dans le treillis, on peut calculer la distance de Hamming d'un encodage $\boldsymbol{c}$ à des mots bruités $\boldsymbol{y}$, en sommant à chaque étape le nombre de bits de sortie $c_j^{(1)}, \ldots, c_j^{(k)}$ différents de $y_j^{(1)}, \ldots, y_j^{(k)}$.



$\boldsymbol{y}^{(1)} =$	1		1		0		1		0		1	
$\boldsymbol{y}^{(2)} =$	1		1		0		1		1		1	
$\boldsymbol{c}^{(1)} =$	1		1		0		0		0		1	
$\boldsymbol{c}^{(2)} =$	1		0		0		1		1		1	
erreur $e =$	0	+	1	+	0	+	1	+	0	+	0	= 2

Algorithme de Viterbi

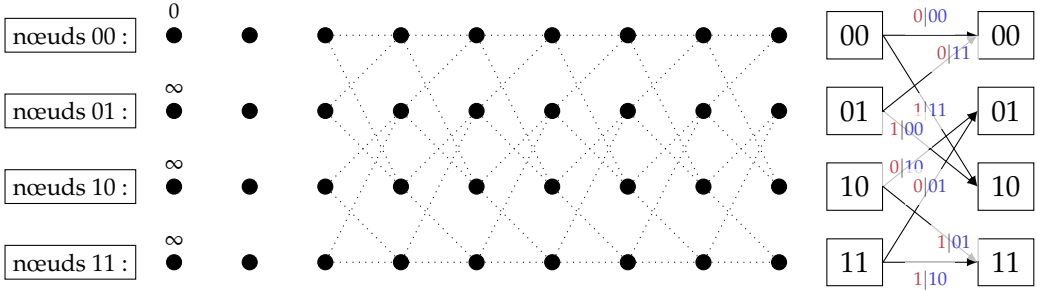
L'algorithme de Viterbi est un algorithme **glouton** et **dynamique** de calcul d'encodage le plus proche (en distance de Hamming) de $\boldsymbol{y}$.

À chaque pas de temps, on tient à jour une liste des **chemins de poids minimaux** menant de l'état initial $(m_{-1} \dots m_{-r+1}) = (0 \dots 0)$ à l'état courant $(m_j \dots m_{j-r+2})$.

Si à l'instant final $j = n + r$, le poids du chemin minimal associé à l'état $(0 \dots 0)$ est le plus petit, alors le décodage à **réussi**. Les arêtes de ce chemin minimal fournissent les bits de message.

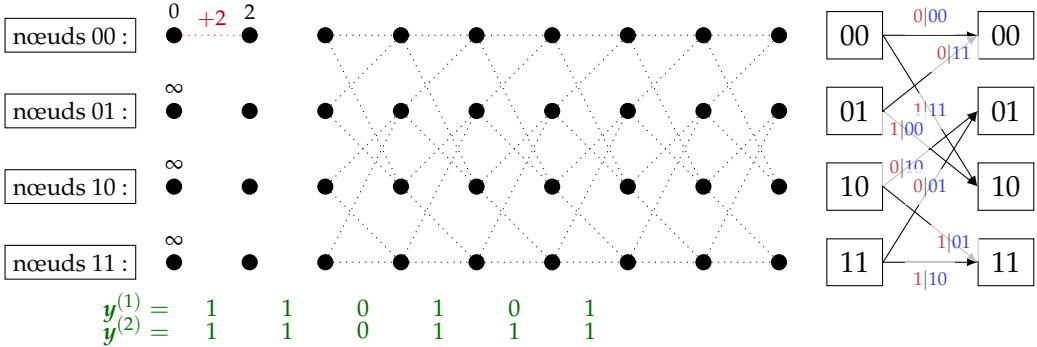
Sinon, le décodage a **échoué**.

Au temps $j = 0$, on associe un poids 0 à l'état initial, et un poids infini aux autres états.



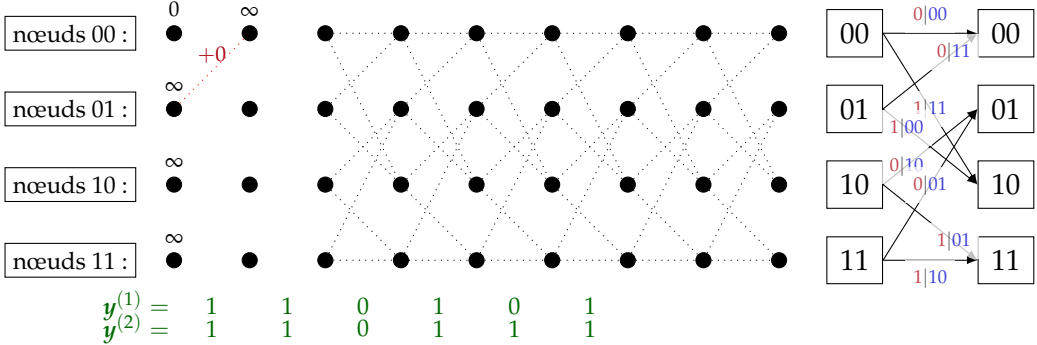
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



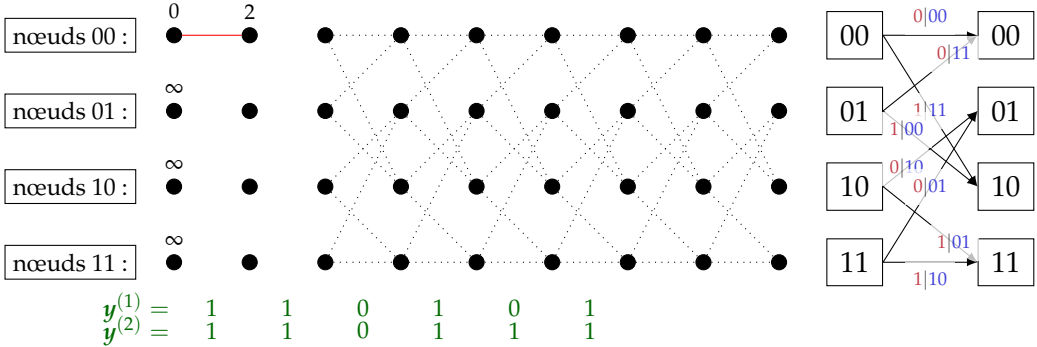
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



Au temps $j \geq 1$. Pour chaque état i :

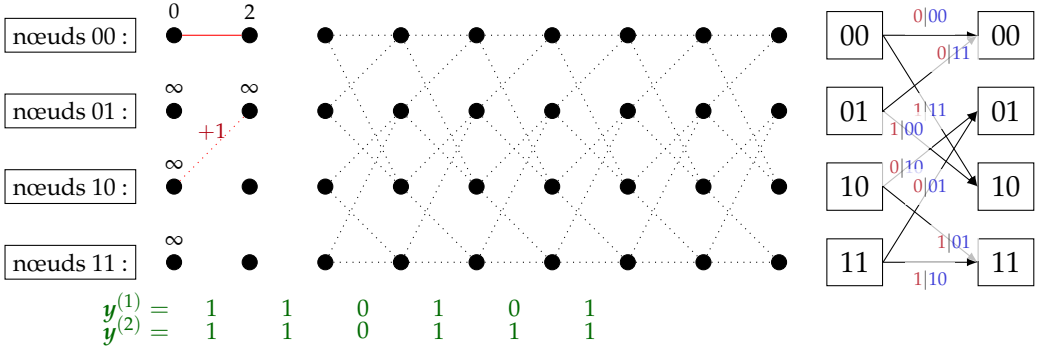
1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



$$\begin{array}{lcccccc} \boldsymbol{y}^{(1)} = & 1 & 1 & 0 & 1 & 0 & 1 \\ \boldsymbol{y}^{(2)} = & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$

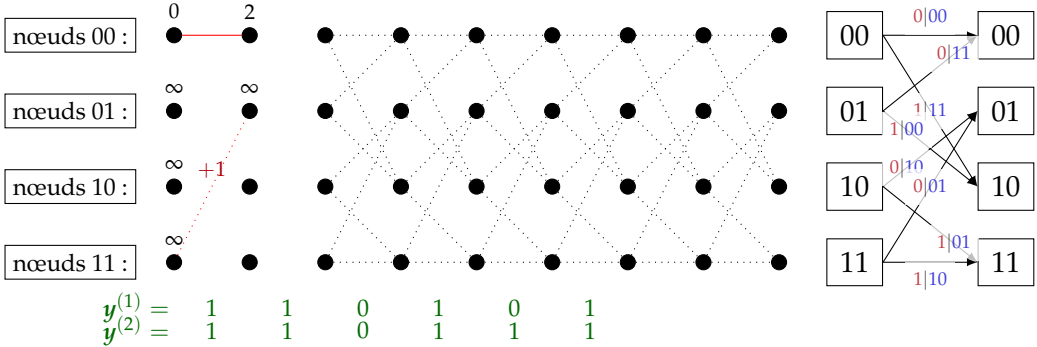
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



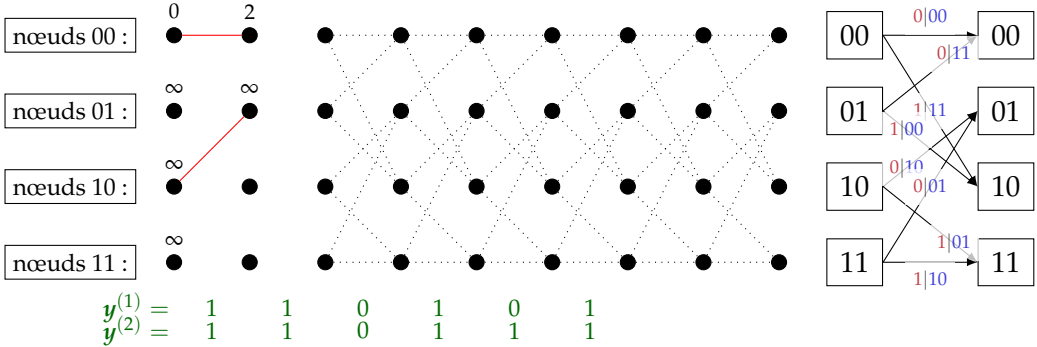
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



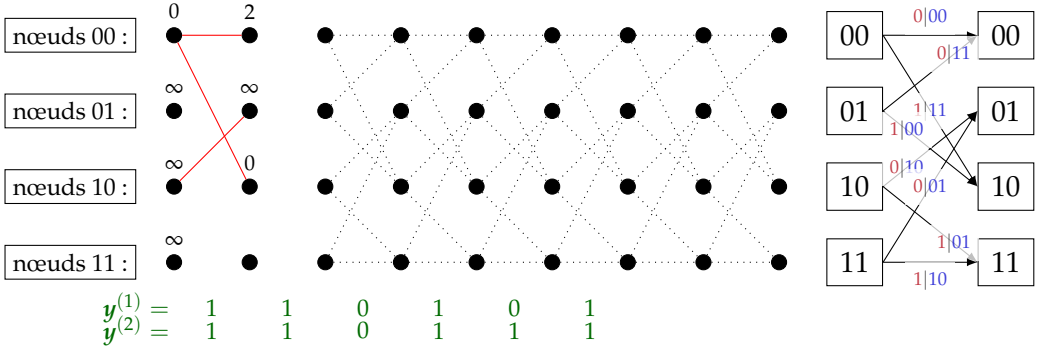
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



Au temps $j \geq 1$. Pour chaque état i :

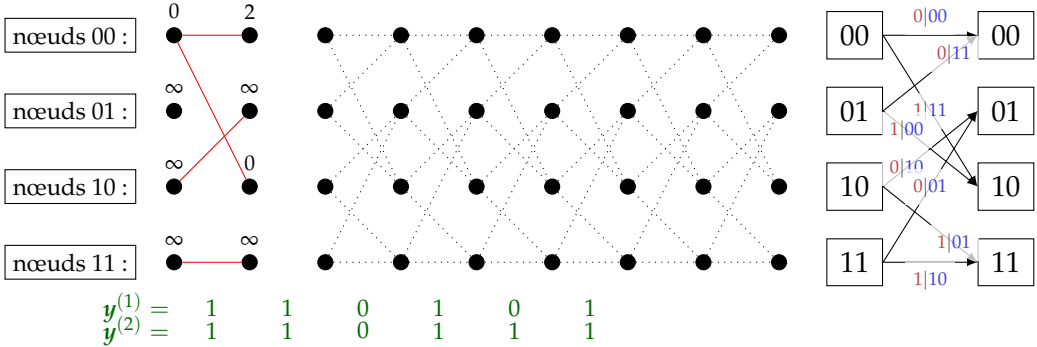
1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



$$\begin{array}{lcccccc} \boldsymbol{y}^{(1)} = & 1 & 1 & 0 & 1 & 0 & 1 \\ \boldsymbol{y}^{(2)} = & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$

Au temps $j \geq 1$. Pour chaque état i :

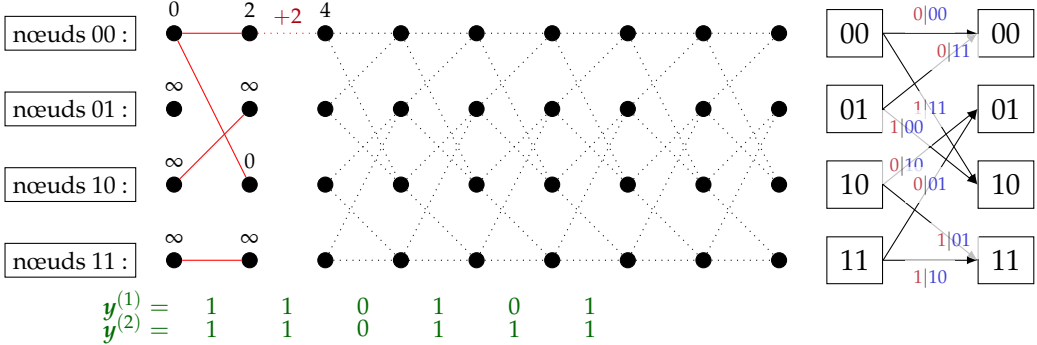
1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



$$\begin{array}{lcccccc} \boldsymbol{y}^{(1)} = & 1 & 1 & 0 & 1 & 0 & 1 \\ \boldsymbol{y}^{(2)} = & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$

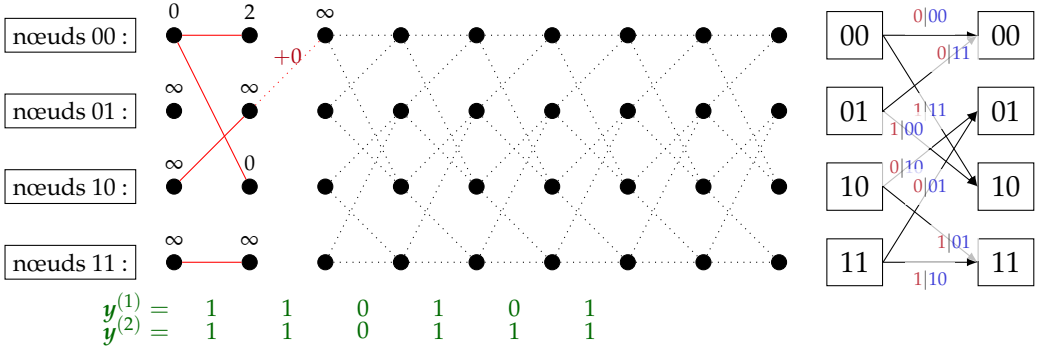
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



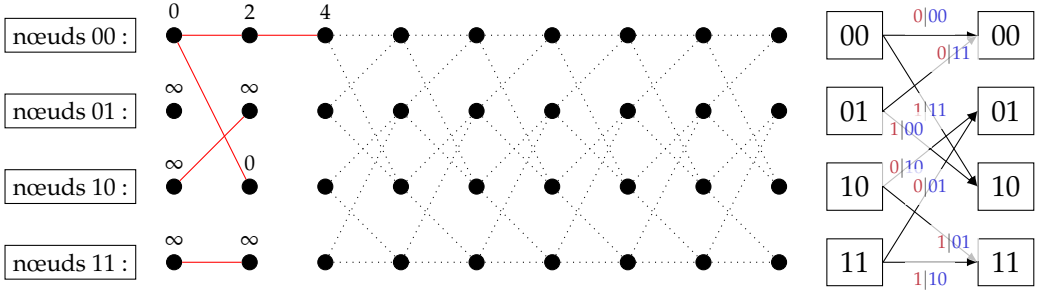
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



Au temps $j \geq 1$. Pour chaque état i :

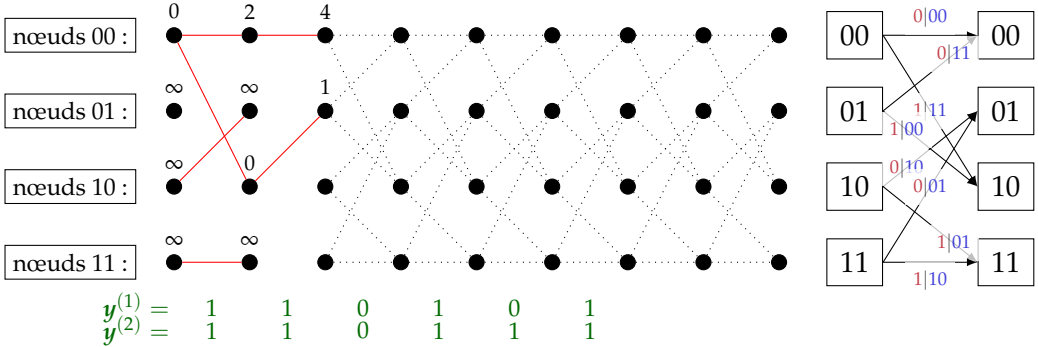
1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



$$\begin{array}{lcccccc} \boldsymbol{y}^{(1)} = & 1 & 1 & 0 & 1 & 0 & 1 \\ \boldsymbol{y}^{(2)} = & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$

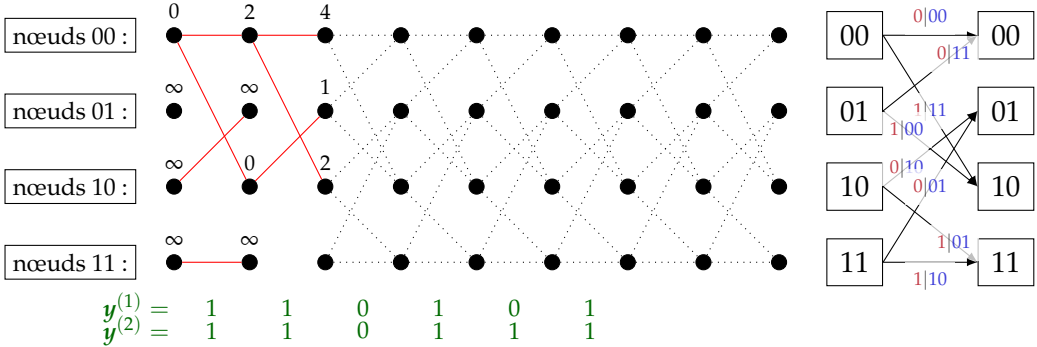
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



Au temps $j \geq 1$. Pour chaque état i :

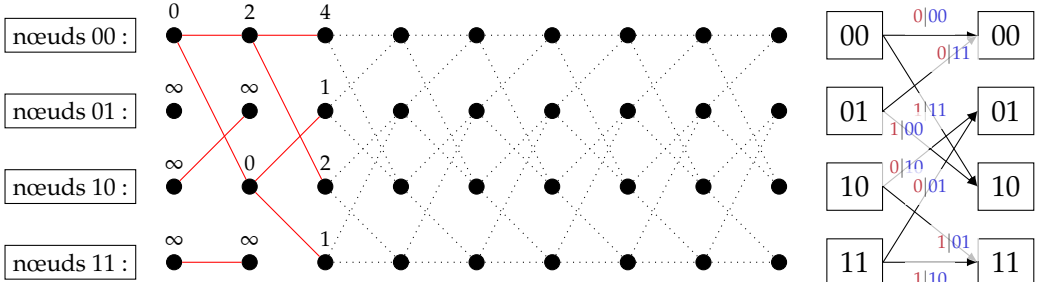
1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



Algorithme de Viterbi

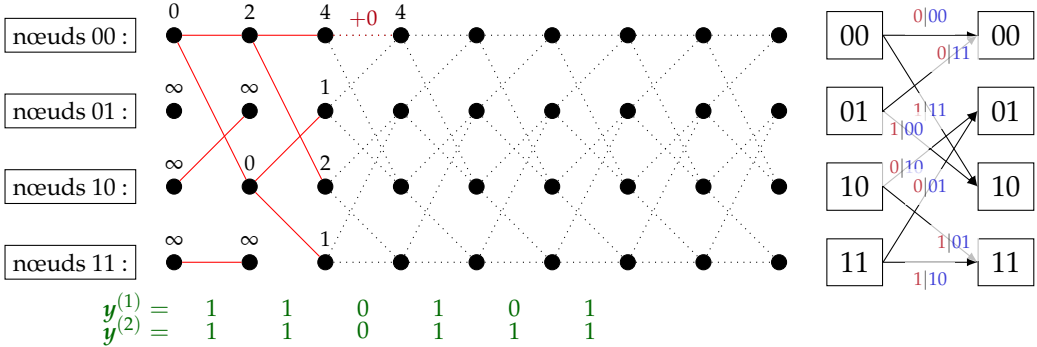
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



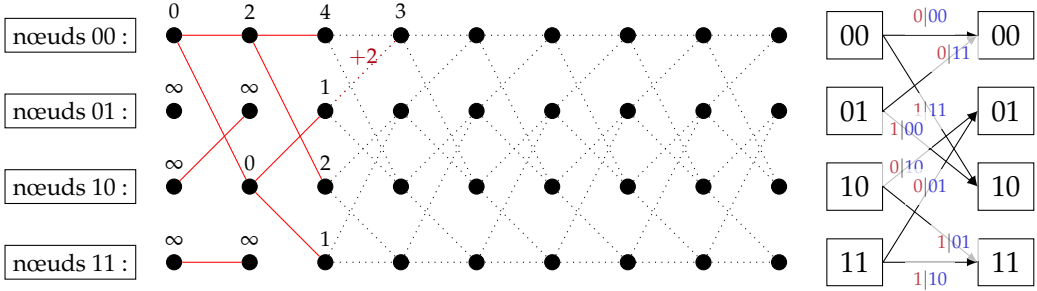
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



Au temps $j \geq 1$. Pour chaque état i :

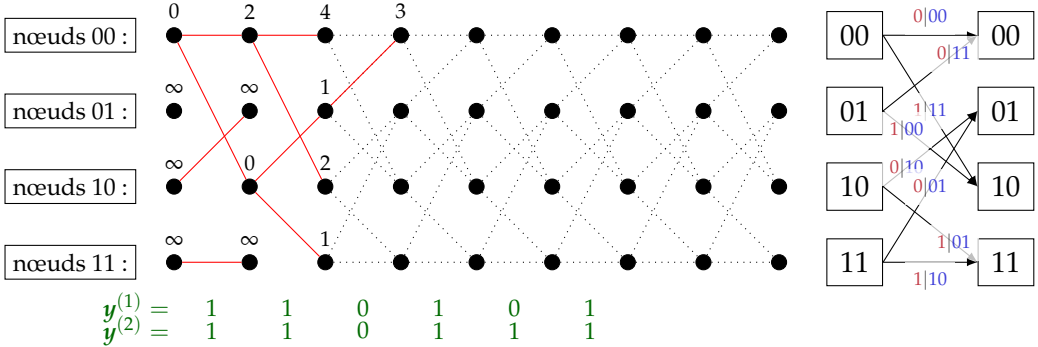
1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



$$\begin{array}{lcccccc} \boldsymbol{y}^{(1)} = & 1 & 1 & 0 & 1 & 0 & 1 \\ \boldsymbol{y}^{(2)} = & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$

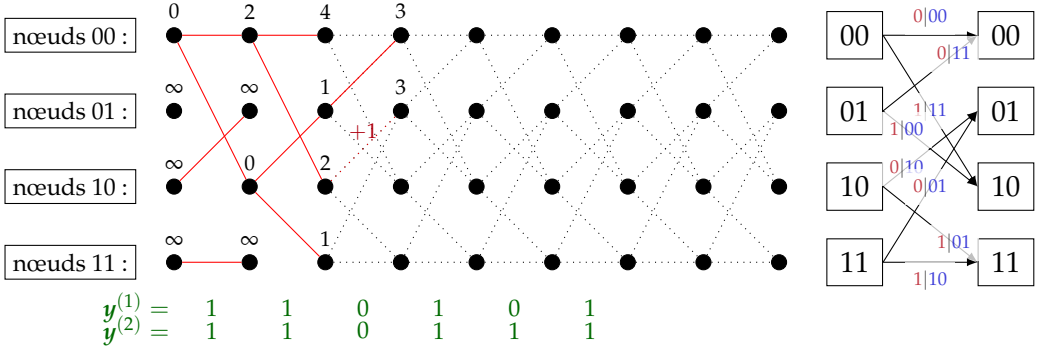
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



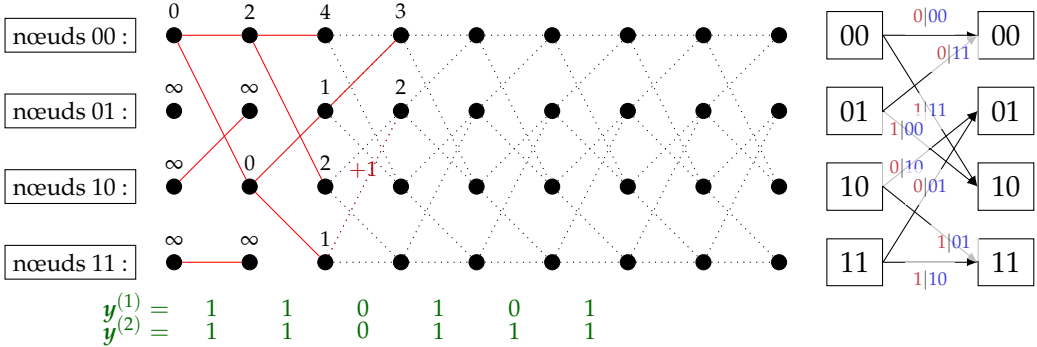
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



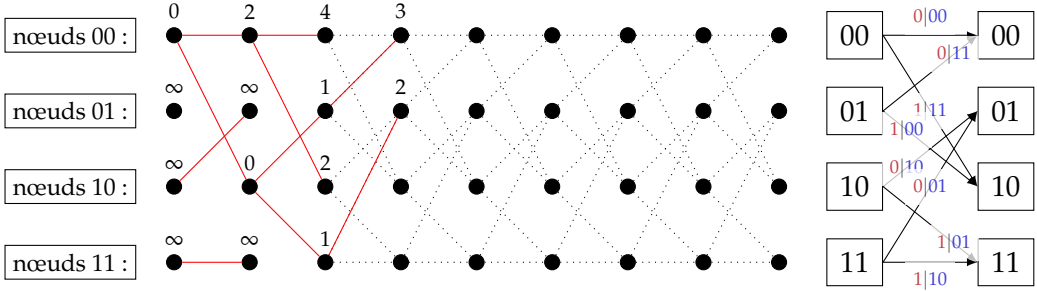
Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



Au temps $j \geq 1$. Pour chaque état i :

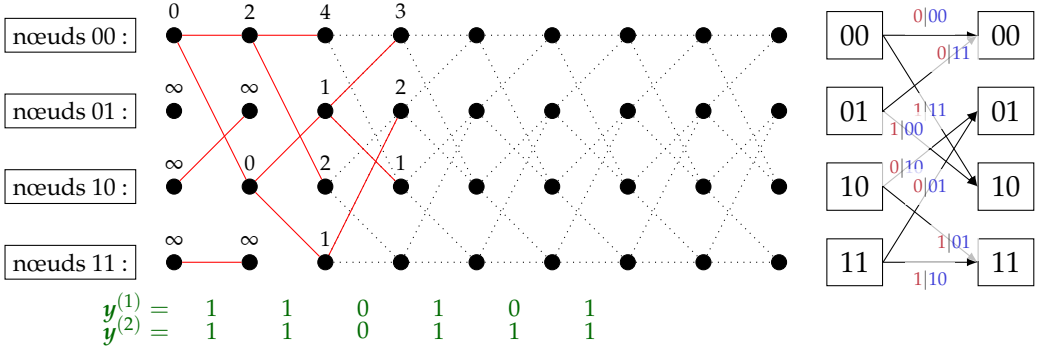
1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



$$\begin{array}{lcccccc} \boldsymbol{y}^{(1)} = & 1 & 1 & 0 & 1 & 0 & 1 \\ \boldsymbol{y}^{(2)} = & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$

Au temps $j \geq 1$. Pour chaque état i :

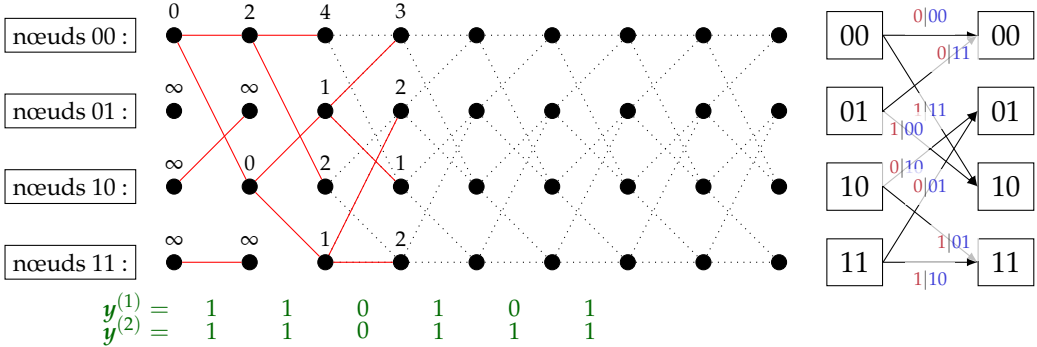
1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



$$\begin{array}{lcccccc} \boldsymbol{y}^{(1)} = & 1 & 1 & 0 & 1 & 0 & 1 \\ \boldsymbol{y}^{(2)} = & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$

Au temps $j \geq 1$. Pour chaque état i :

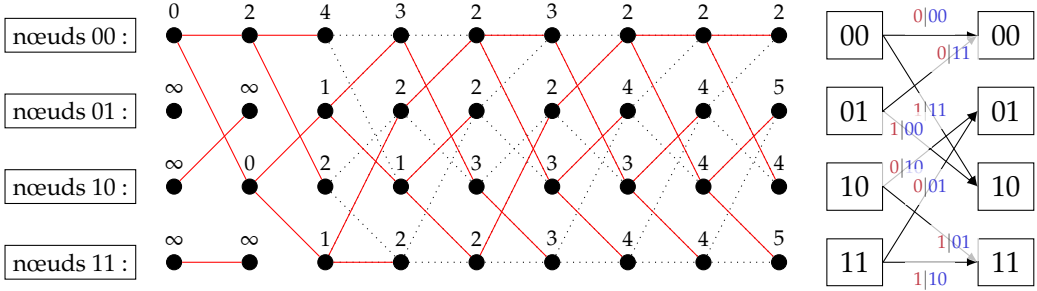
1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)



$$\begin{array}{lcccccc} \boldsymbol{y}^{(1)} = & 1 & 1 & 0 & 1 & 0 & 1 \\ \boldsymbol{y}^{(2)} = & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$

Au temps $j \geq 1$. Pour chaque état i :

1. on cherche les deux états qui précédent i
2. on calcule les suppléments d'erreur associées à ces transitions, que l'on ajoute ensuite au poids de ces états
3. on met à jour le poids de i avec la valeur minimale des poids des deux chemins
4. on garde en mémoire la transition de poids minimal (cf. arête rouge)

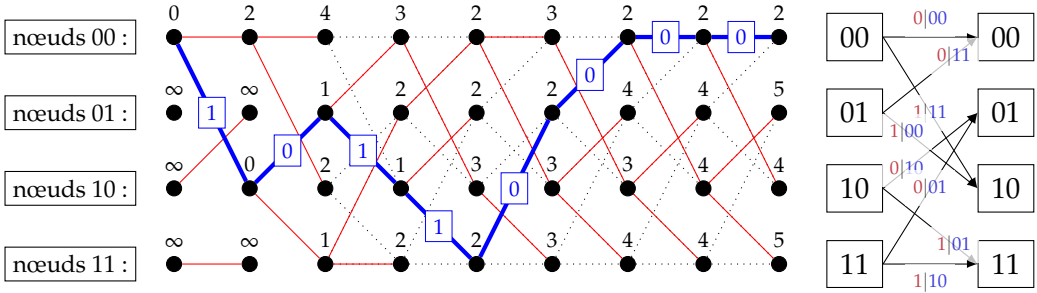


$$\begin{array}{lcccccc} \boldsymbol{y}^{(1)} = & 1 & 1 & 0 & 1 & 0 & 1 \\ \boldsymbol{y}^{(2)} = & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$

Algorithme de Viterbi

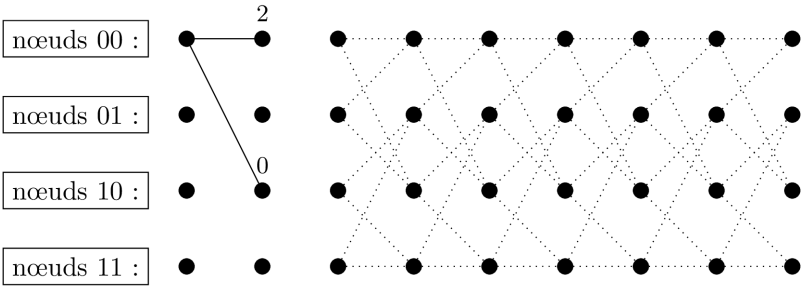
Une fois le processus terminé, **si l'état** $(0,\dots,0)$ **correspond à un poids d'erreur minimal**, on remonte ce chemin de poids minimal pour retrouver le message $\boldsymbol{m}$.

Dans l'exemple, on avait les mots de code $\begin{cases} \boldsymbol{c}^{(1)} = 1\ 1\ 0\ 0\ 0\ 1 \\ \boldsymbol{c}^{(2)} = 1\ 0\ 0\ 1\ 1\ 1 \end{cases}$ donc le poids de l'erreur était bien 2.

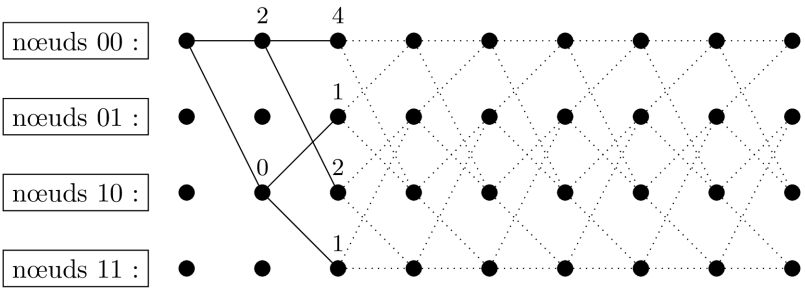


$$\begin{matrix} \boldsymbol{y}^{(1)} = & 1 & 1 & 0 & 1 & 0 & 1 \\ \boldsymbol{y}^{(2)} = & 1 & 1 & 0 & 1 & 1 & 1 \end{matrix}$$

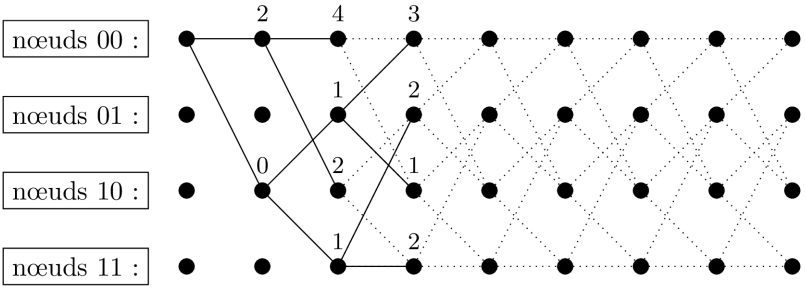
Déroulé complet de l'algorithme pour $\boldsymbol{y} = (110101), (110111)$.



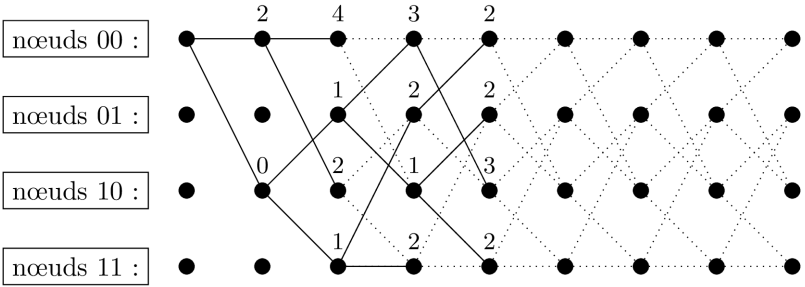
Déroulé complet de l'algorithme pour $\boldsymbol{y} = (110101), (110111)$.



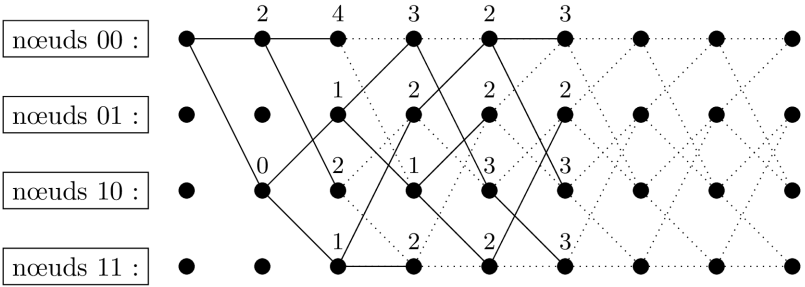
Déroulé complet de l'algorithme pour $\boldsymbol{y} = (110101), (110111)$.



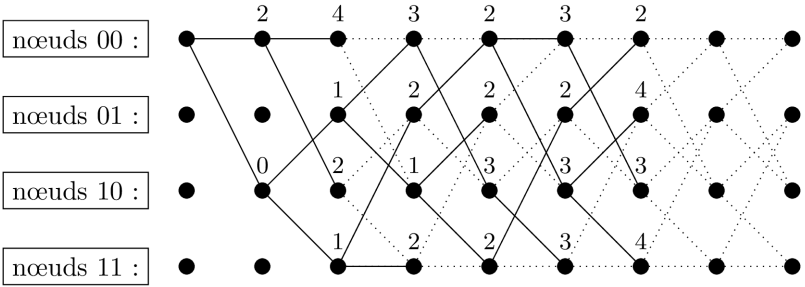
Déroulé complet de l'algorithme pour $\boldsymbol{y} = (110101), (110111)$.



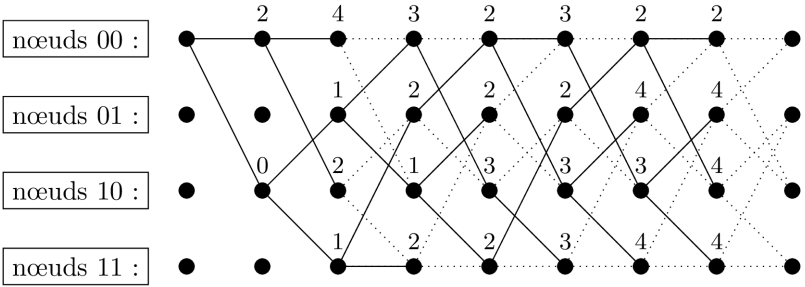
Déroulé complet de l'algorithme pour $\boldsymbol{y} = (110101), (110111)$.



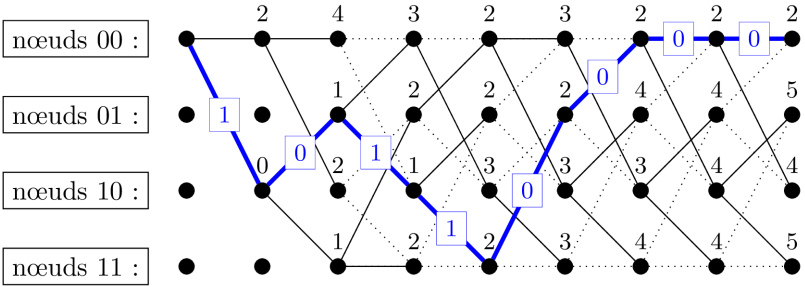
Déroulé complet de l'algorithme pour $\boldsymbol{y} = (110101), (110111)$.



Déroulé complet de l'algorithme pour $\boldsymbol{y} = (110101), (110111)$.



Déroulé complet de l'algorithme pour $\boldsymbol{y} = (110101), (110111)$.

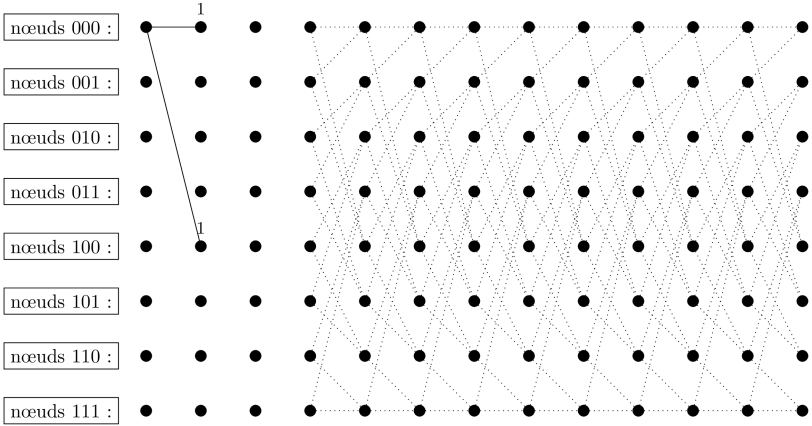


Plan

Pour $r = 4$ et $k = 2$

Exemple pour le couple de générateurs $(1 + x + x^3, 1 + x^2 + x^3)$ et le message bruité

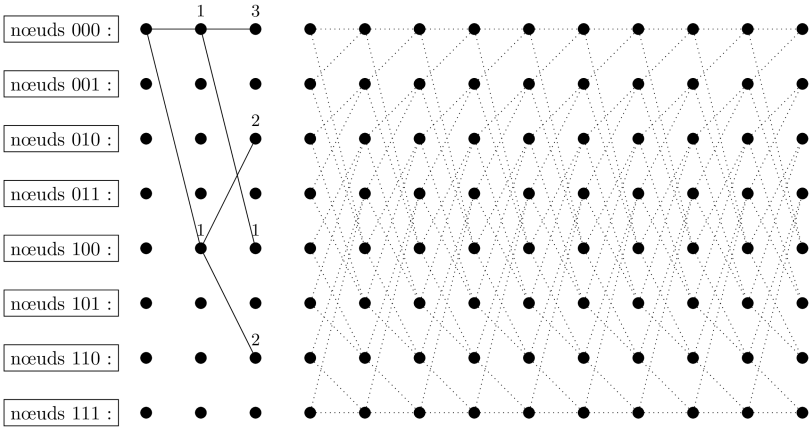
$$\boldsymbol{y}^{(1)} = 010010011 \qquad \boldsymbol{y}^{(2)} = 110000001$$



Pour $r = 4$ et $k = 2$

Exemple pour le couple de générateurs $(1 + x + x^3, 1 + x^2 + x^3)$ et le message bruité

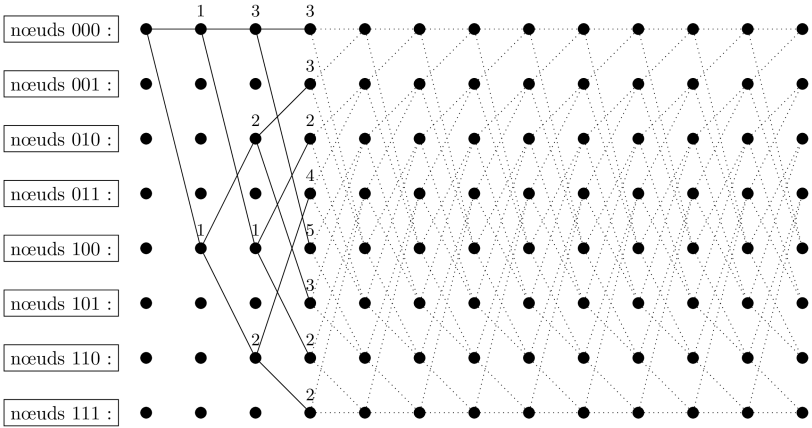
$$\boldsymbol{y}^{(1)} = 010010011 \qquad \boldsymbol{y}^{(2)} = 110000001$$



Pour $r = 4$ et $k = 2$

Exemple pour le couple de générateurs $(1 + x + x^3, 1 + x^2 + x^3)$ et le message bruité

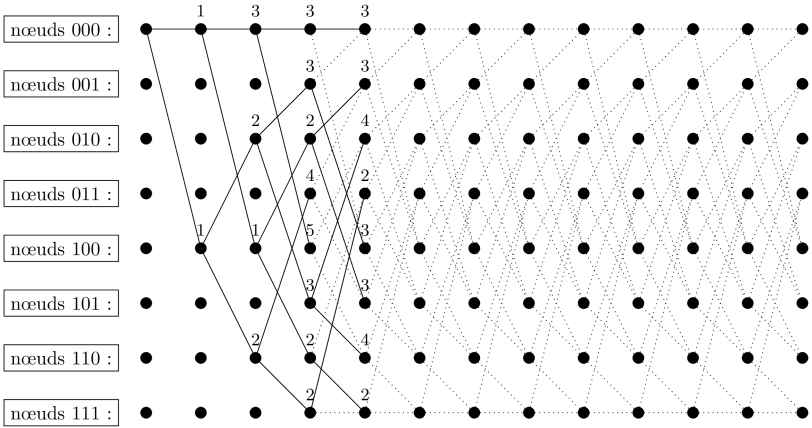
$$\boldsymbol{y}^{(1)} = 010010011 \qquad \boldsymbol{y}^{(2)} = 110000001$$



Pour $r = 4$ et $k = 2$

Exemple pour le couple de générateurs $(1 + x + x^3, 1 + x^2 + x^3)$ et le message bruité

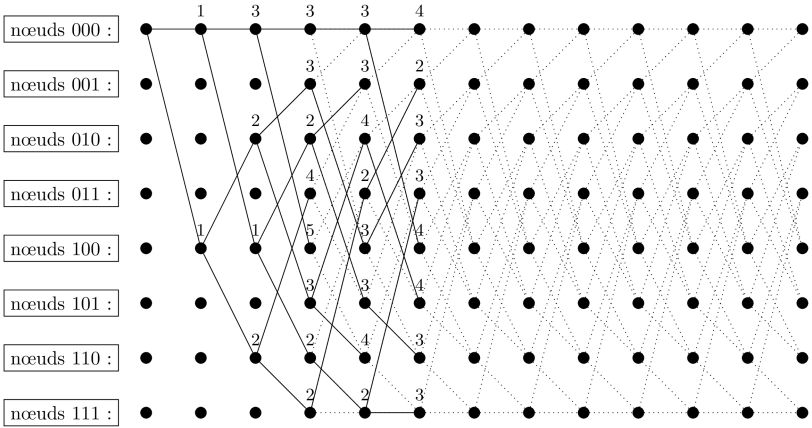
$$\boldsymbol{y}^{(1)} = 010010011 \qquad \boldsymbol{y}^{(2)} = 110000001$$



Pour $r = 4$ et $k = 2$

Exemple pour le couple de générateurs $(1 + x + x^3, 1 + x^2 + x^3)$ et le message bruité

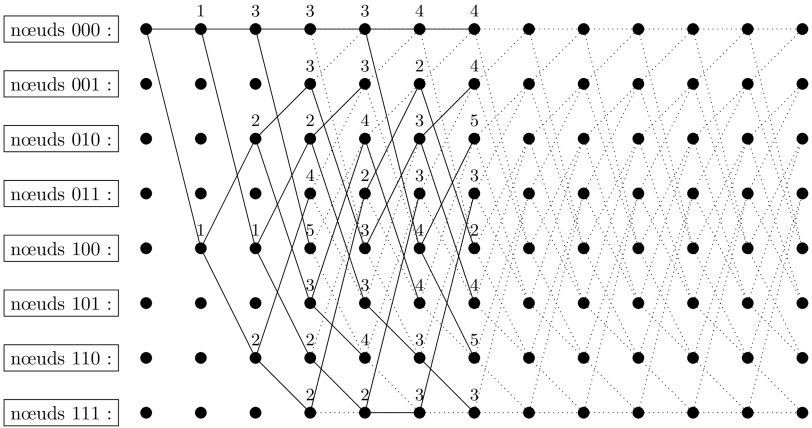
$$\boldsymbol{y}^{(1)} = 010010011 \qquad \boldsymbol{y}^{(2)} = 110000001$$



Pour $r = 4$ et $k = 2$

Exemple pour le couple de générateurs $(1 + x + x^3, 1 + x^2 + x^3)$ et le message bruité

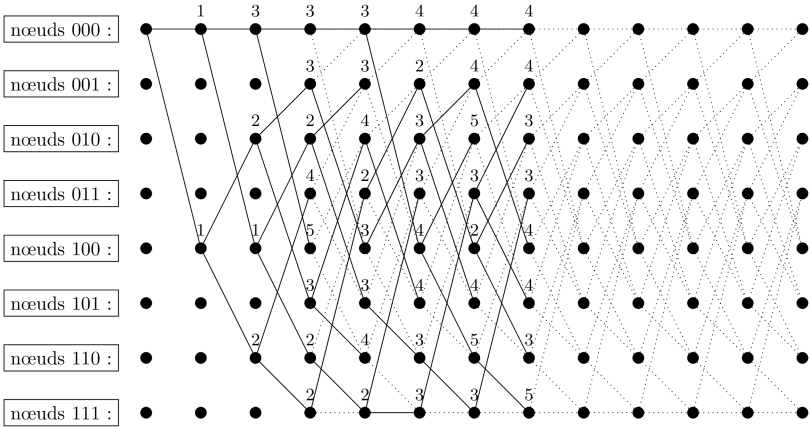
$$\boldsymbol{y}^{(1)} = 010010011 \qquad \boldsymbol{y}^{(2)} = 110000001$$



Pour $r = 4$ et $k = 2$

Exemple pour le couple de générateurs $(1 + x + x^3, 1 + x^2 + x^3)$ et le message bruité

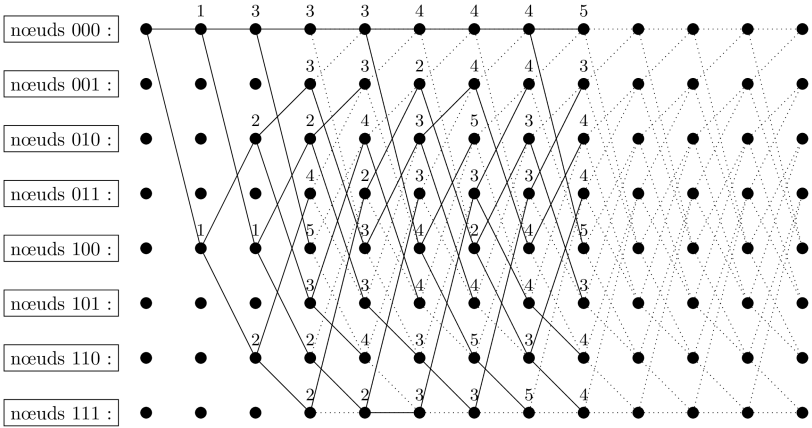
$$\boldsymbol{y}^{(1)} = 010010011 \qquad \boldsymbol{y}^{(2)} = 110000001$$



Pour $r = 4$ et $k = 2$

Exemple pour le couple de générateurs $(1 + x + x^3, 1 + x^2 + x^3)$ et le message bruité

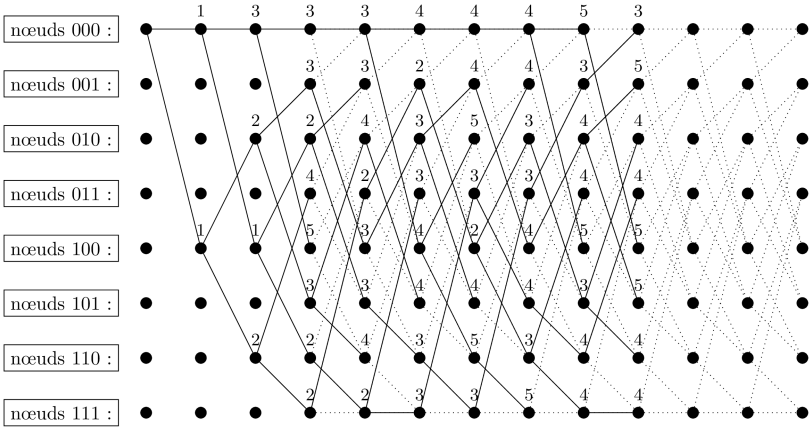
$$\boldsymbol{y}^{(1)} = 010010011 \qquad \boldsymbol{y}^{(2)} = 110000001$$



Pour $r = 4$ et $k = 2$

Exemple pour le couple de générateurs $(1 + x + x^3, 1 + x^2 + x^3)$ et le message bruité

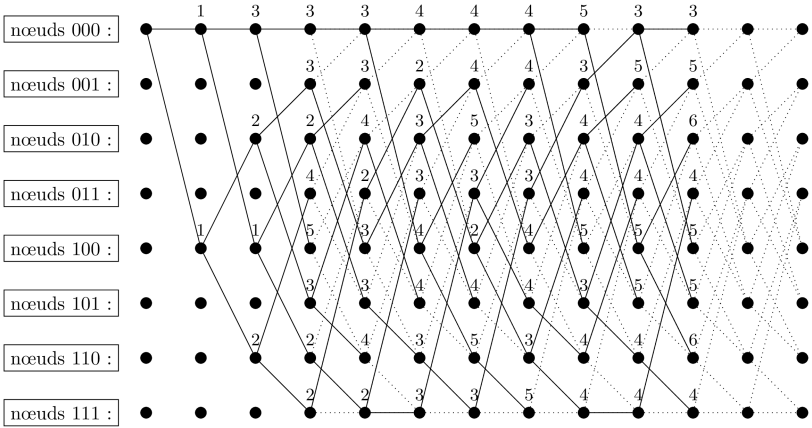
$$\boldsymbol{y}^{(1)} = 010010011 \qquad \boldsymbol{y}^{(2)} = 110000001$$



Pour $r = 4$ et $k = 2$

Exemple pour le couple de générateurs $(1 + x + x^3, 1 + x^2 + x^3)$ et le message bruité

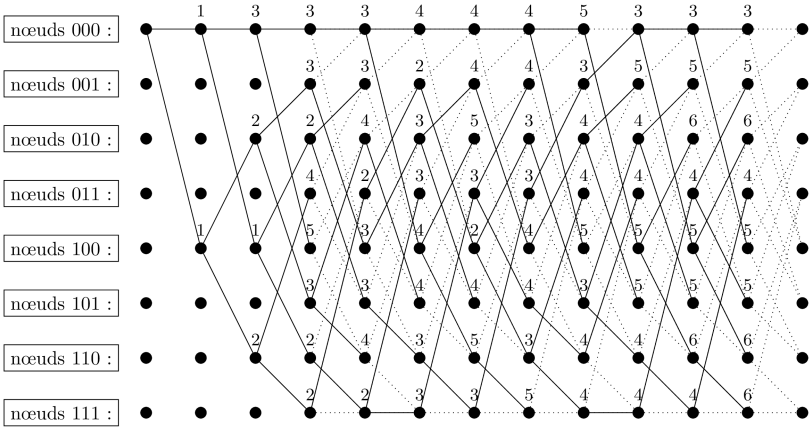
$$\boldsymbol{y}^{(1)} = 010010011 \qquad \boldsymbol{y}^{(2)} = 110000001$$



Pour $r = 4$ et $k = 2$

Exemple pour le couple de générateurs $(1 + x + x^3, 1 + x^2 + x^3)$ et le message bruité

$$\boldsymbol{y}^{(1)} = 010010011 \qquad \boldsymbol{y}^{(2)} = 110000001$$



Pour $r = 4$ et $k = 2$

Exemple pour le couple de générateurs $(1 + x + x^3, 1 + x^2 + x^3)$ et le message bruité

$$\boldsymbol{y}^{(1)} = 010010011 \qquad \boldsymbol{y}^{(2)} = 110000001$$