Vkládání pomocí Viterbiho algoritmu

Andrew Kozlik

KA MFF UK

Vkládání pomocí Viterbiho algoritmu

- Cíl:
  - Využít teorii konvolučních kódů.

- Motivace:
  - Viterbiho dekodér je soft-decision dekodér.
  - Každému prvku nosiče přiřadíme váhu, která udává jeho citlivost na změnu.
  - Nebudeme nutně minimalizovat počet změn v nosiči, ale celkovou váhu změn.

Značení

- Nosič rozdělujeme na bloky n hodnot z konečného tělesa $\mathbb{F}_q$.
- Rozdíl oproti maticovému vkládání:
  - Nepracujeme s jednotlivými bloky samostatně.
  - Sestrojíme z nich posloupnost vektorů $\{\mathbf{x}_i\}_{i=0}^{\ell-1}$, kde $\mathbf{x}_i = (x_i^{(1)}, \ldots, x_i^{(n)})^{\mathrm{T}} \in \mathbb{F}_q^n$.
- Máme tedy nosič $\{\mathbf{x}_i\}_{i=0}^{\ell-1}$ a stegoobjekt $\{\mathbf{y}_i\}_{i=0}^{\ell-1}$.
- Zpráva je obecně posloupnost vektorů $\{\mathbf{z}_i\}_{i=0}^{\ell-1}$, kde $\mathbf{z}_i \in \mathbb{F}_q^m$.
  - My se omezíme na případ $m = 1$.
  - Zpráva je pak posloupnost jednosložkových vektorů.
  - Budeme ji psát jako posloupnost skalárů $\{z_i\}_{i=0}^{\ell-1}$.

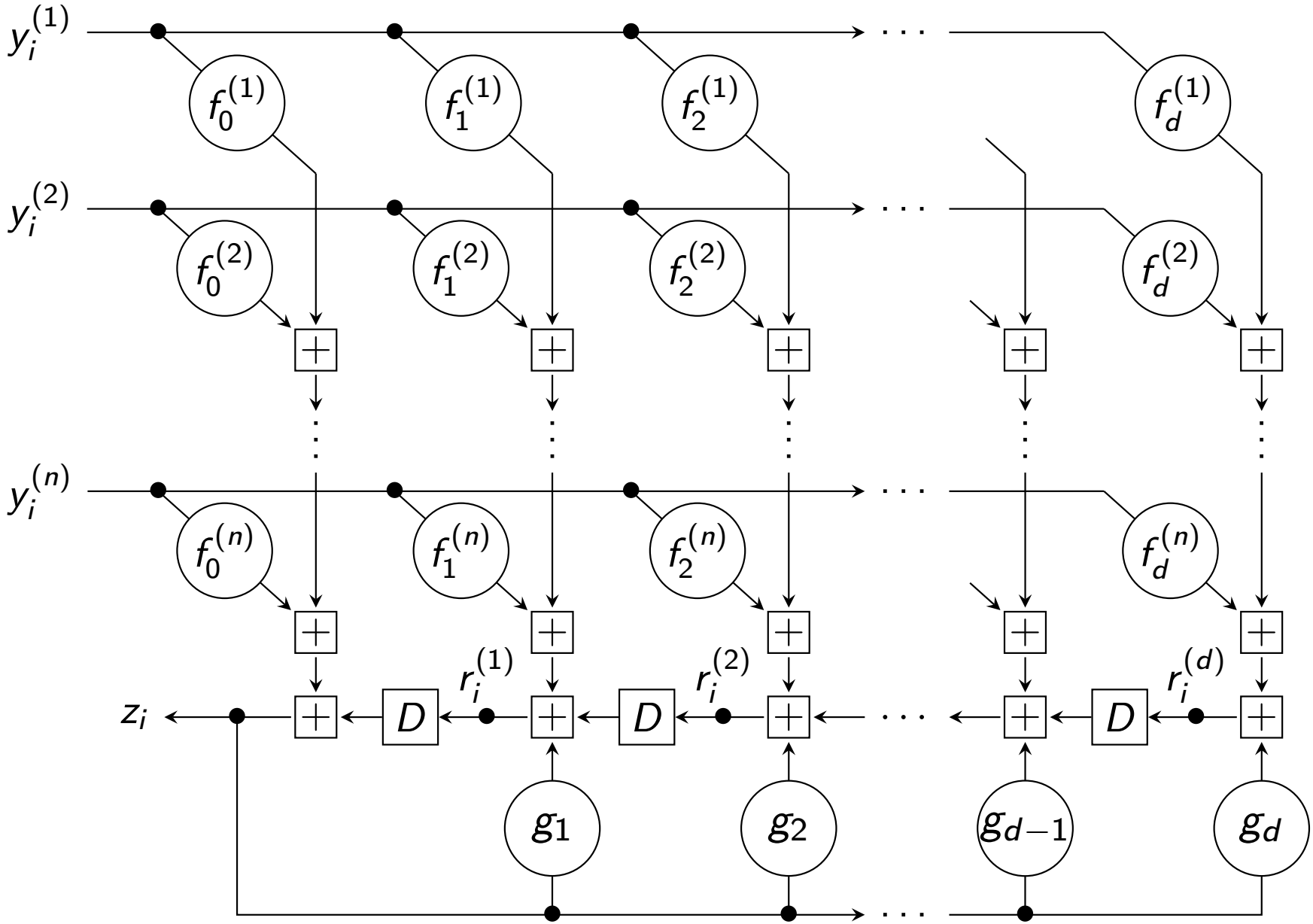
Proč se omezujeme na $m = 1$?

- Zobecnění pro $m > 1$ je jednoduché.
- Složitost Viterbiho algoritmu neúměrně narůstá s m.
- Relativní kapacita nosiče je $\alpha = \dfrac{m \log_2 q}{n}$.
- Čili jsme omezeni na hodnoty tvaru $\alpha = \dfrac{\log_2 q}{n}$.
- Nevadí. Ve steganografii obvykle cílíme na hodnoty blízké 0.

Konvoluční extrakce

- Extrakce zprávy se provádí konvolučním překladačem.
- Implementace pomocí konvolučního kodéru.
  - Kontrolorova normální forma: n paměťových registrů.
  - Pozorovatelova normální forma: $m = 1$ paměťový registr.
- $r_i^{(j)}$ = hodnota j-té buňky registru na konci i-tého kroku.
- d = délka registru.
- D = operátor zpoždění.
- Definujeme $r_0^{(j)} = 0$ pro $1 \leq j \leq d$.
- Hodnoty $f_j^{(k)} \in \mathbb{F}_q$ a $g_j \in \mathbb{F}_q$ jsou konstanty a $g_0 = 0$.

Konvoluční extrakce



Algoritmus (konvoluční extrakce)

vstup: stegoobjekt $\{\mathbf{y}_i\}_{i=0}^{\ell-1}$ z $\mathbb{F}_q^n$,
parametry $f_j^{(k)} \in \mathbb{F}_q$ a $g_j \in \mathbb{F}_q$

výstup: zpráva $\{z_i\}_{i=0}^{\ell-1}$

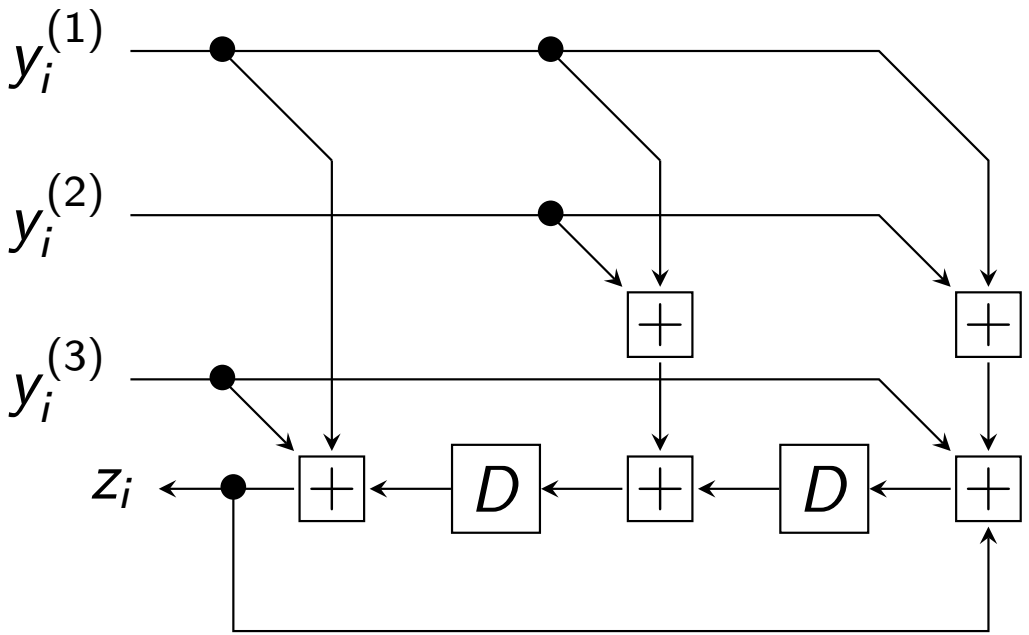
1 $(s_0, \ldots, s_d) := (0, \ldots, 0)$
2 **for** $i = 0, \ldots, \ell - 1$ **do**
3 **for** $j = 1, \ldots, n$ **do**
4 $(s_0, \ldots, s_d) := (s_0, \ldots, s_d) + y_i^{(j)}(f_0^{(j)}, \ldots, f_d^{(j)})$
5 $z_i := s_0$
6 $(s_0, \ldots, s_d) := (s_0, \ldots, s_d) + s_0(g_0, \ldots, g_d)$
7 $(s_0, \ldots, s_d) := (s_1, \ldots, s_d, 0)$
8 **return** $\{z_i\}_{i=0}^{\ell-1}$

Vkládání

- Extrakční algoritmus popíšeme jako konečný překladač.
- Pro tento překladač sestrojíme trelážový graf.
  (Graf vývoje stavů v závislosti na vstupu.)
- Obecně: Cesty v grafu = všechny možné vstupy překladače.
- Náš trik: Graf sestavíme tak, aby obsahoval pouze ty cesty (vstupy), jejichž výstupem je zpráva, kterou chceme vložit.
- Tyto cesty jsou kandidátky na stegoobjekt.
- Viterbiho algoritmus potom najde cestu, která se nejméně liší od nosiče, resp. má nejmenší váhu (distorzi).

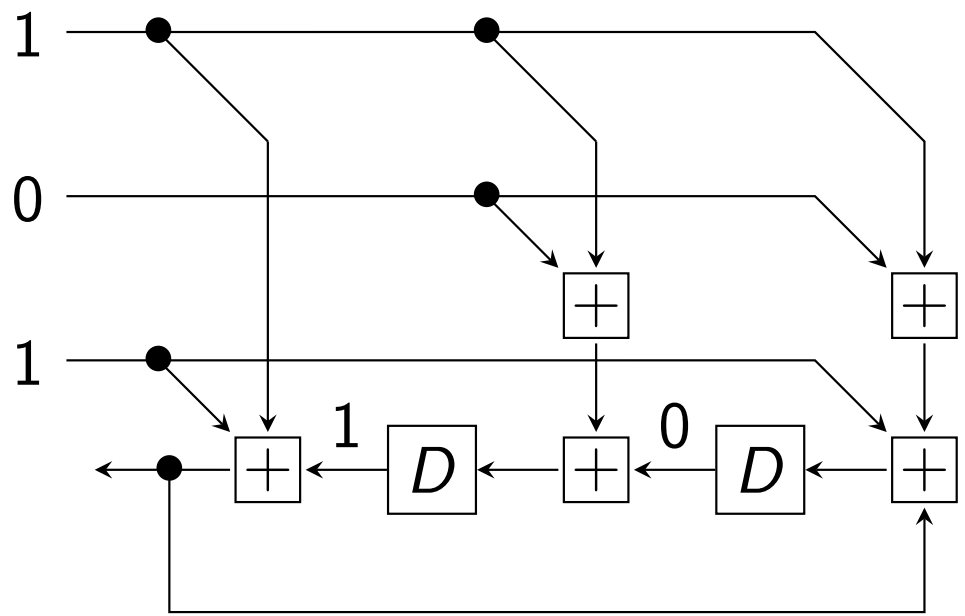
Příklad

- Celý proces předvedeme na následujícím příkladu.
- Mějme extrakční algoritmus nad $\mathbb{F}_2$ s parametry $\mathbf{f}(D) = \left(1 + D + D^2, D + D^2, 1 + D^2\right)$ a $g(D) = D^2$:



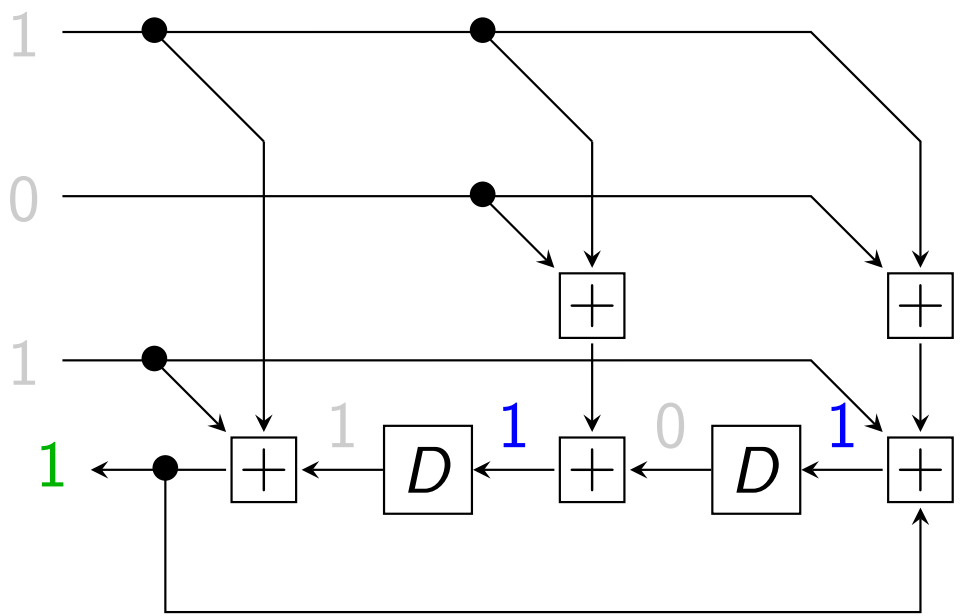
Příklad

- Ukázka přechodu automatu:
  - Počáteční stav $r_{i-1}^{(1)} r_{i-1}^{(2)} = 10$. Vstup $\mathbf{y}_i = (1, 0, 1)^{\mathrm{T}}$.
  - Výsledný stav $r_i^{(1)} r_i^{(2)} = 11$. Výstup $z_i = 1$.
  - Zápis: $\boxed{10} \xrightarrow{101/1} \boxed{11}$

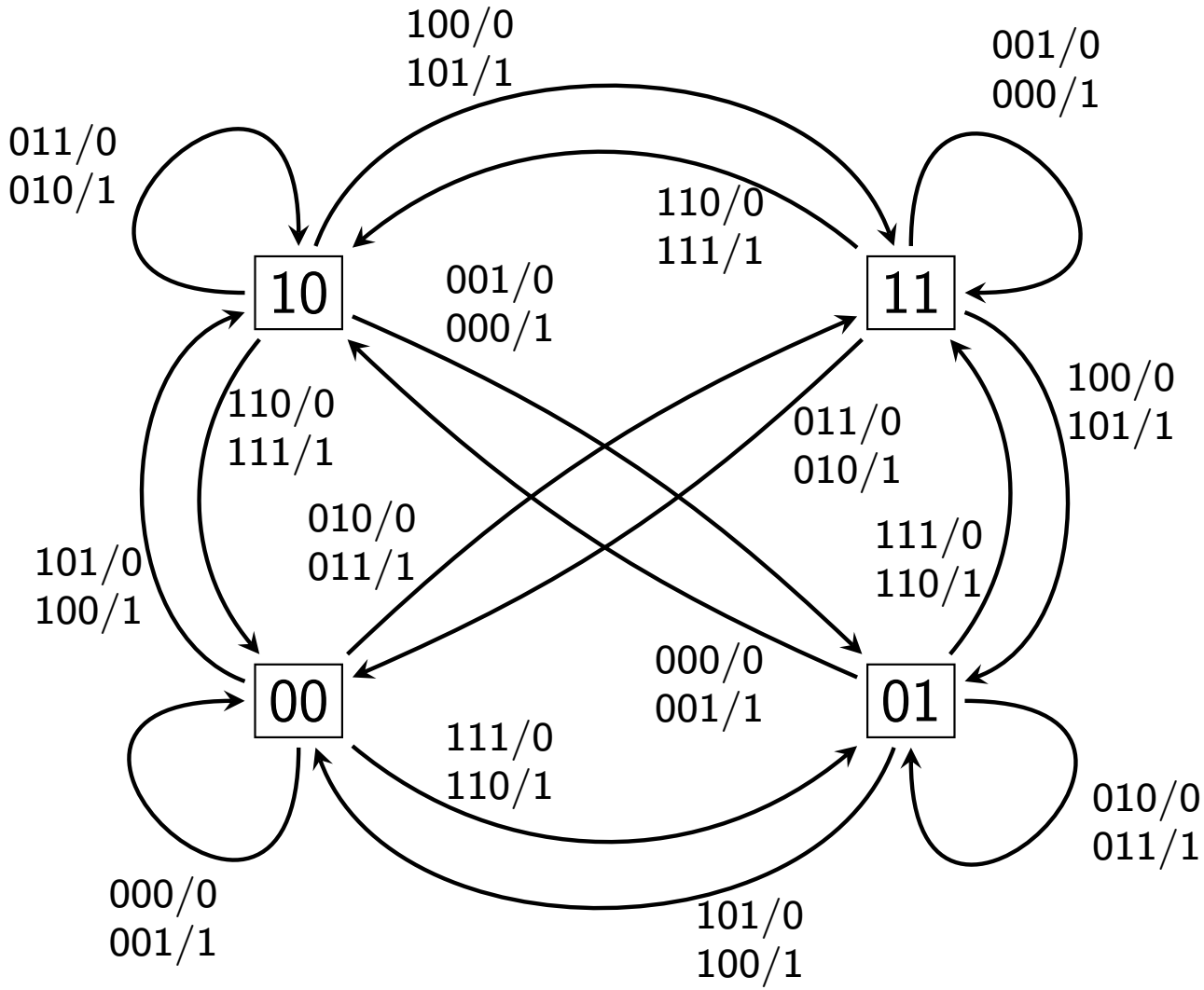


Příklad

- Ukázka přechodu automatu:
  - Počáteční stav $r_{i-1}^{(1)} r_{i-1}^{(2)} = 10$. Vstup $\mathbf{y}_i = (1, 0, 1)^{\mathrm{T}}$.
  - Výsledný stav $r_i^{(1)} r_i^{(2)} = 11$. Výstup $z_i = 1$.
  - Zápis: $\boxed{10} \xrightarrow{101/1} \boxed{11}$

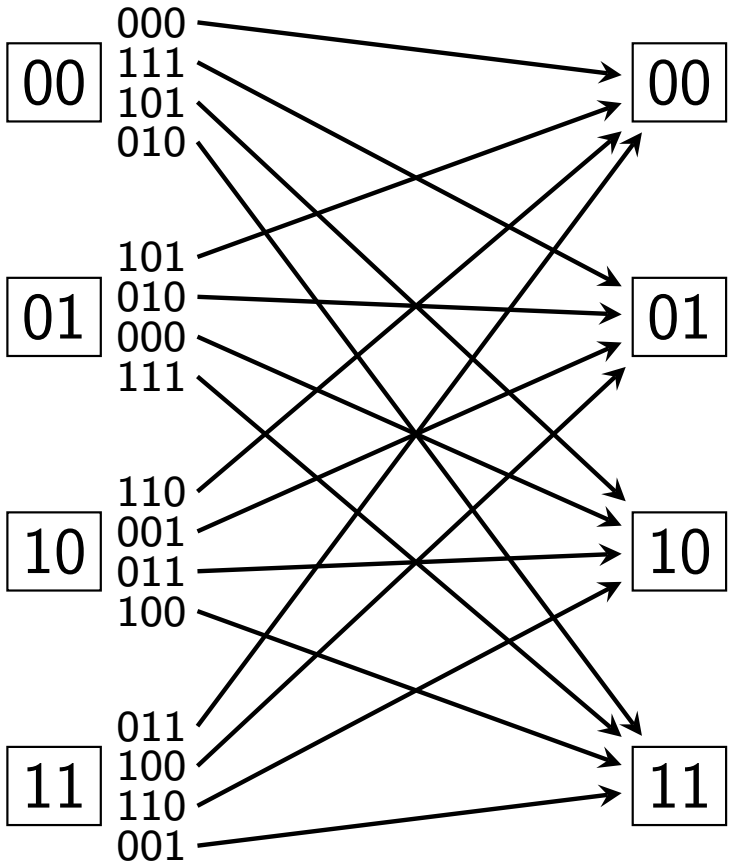


Příslušný konečný překladač

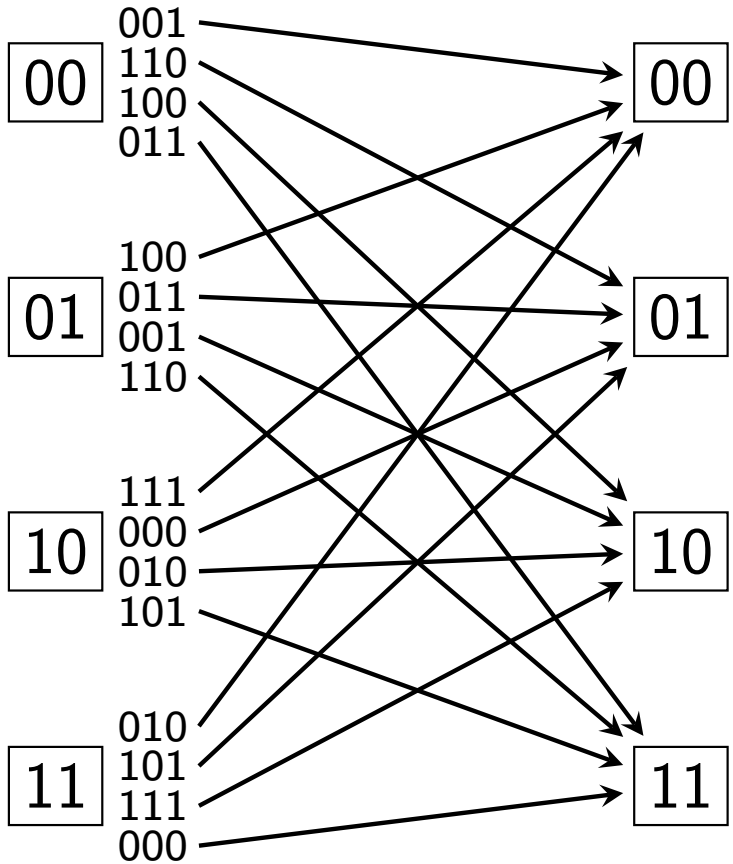


Trelážové moduly

Přechody rozdělíme do dvou trelážových modulů podle jejich výstupních hodnot.



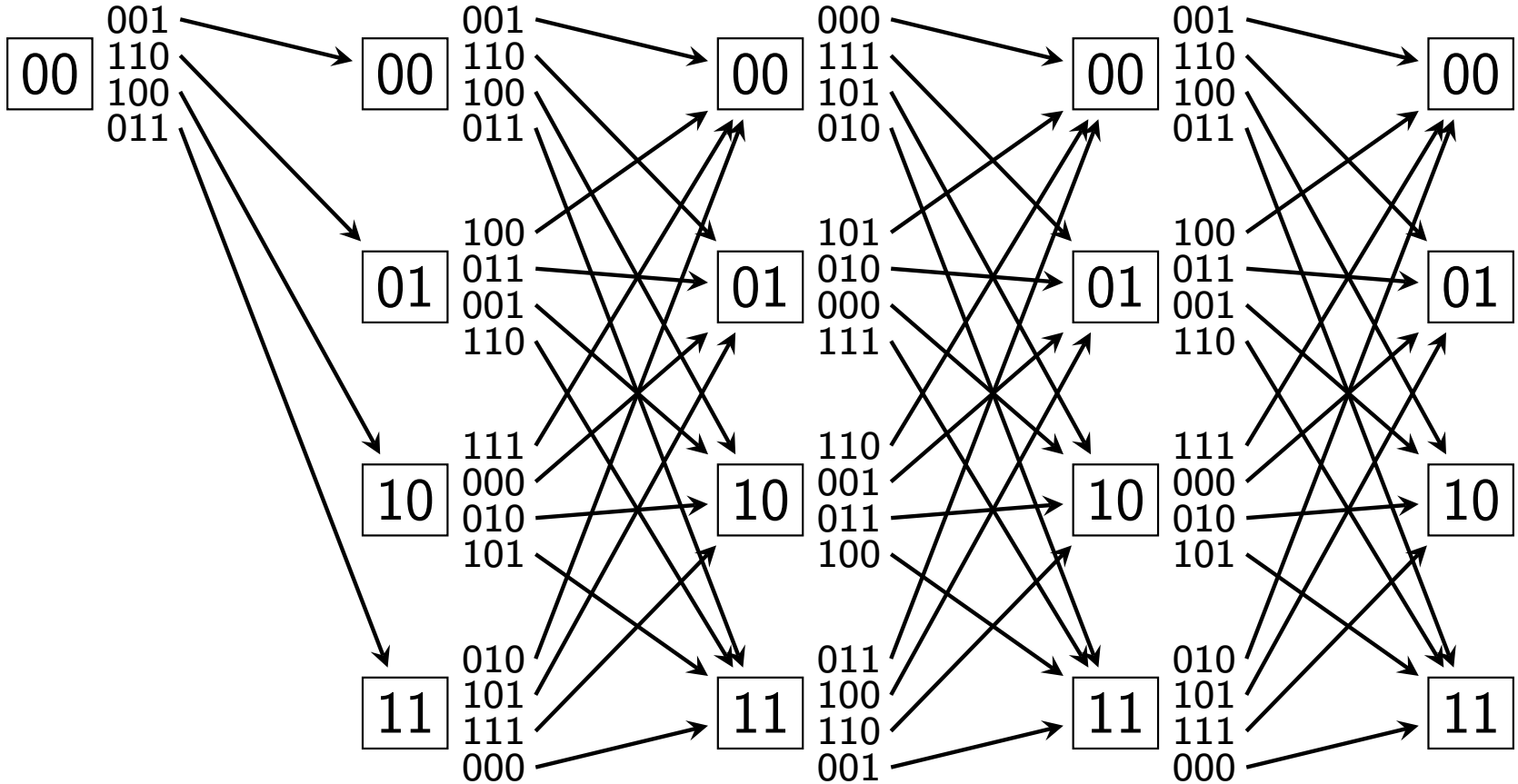
Výstup 0



Výstup 1

Treláž pro vložení zprávy 1101

V závislosti na zprávě pospojujeme moduly do treláže.



Viterbiho algoritmus najde cestu treláží, která se nejvíce „podobá“ nosiči.

Rozdíly oproti treláži pro dekódování kódů

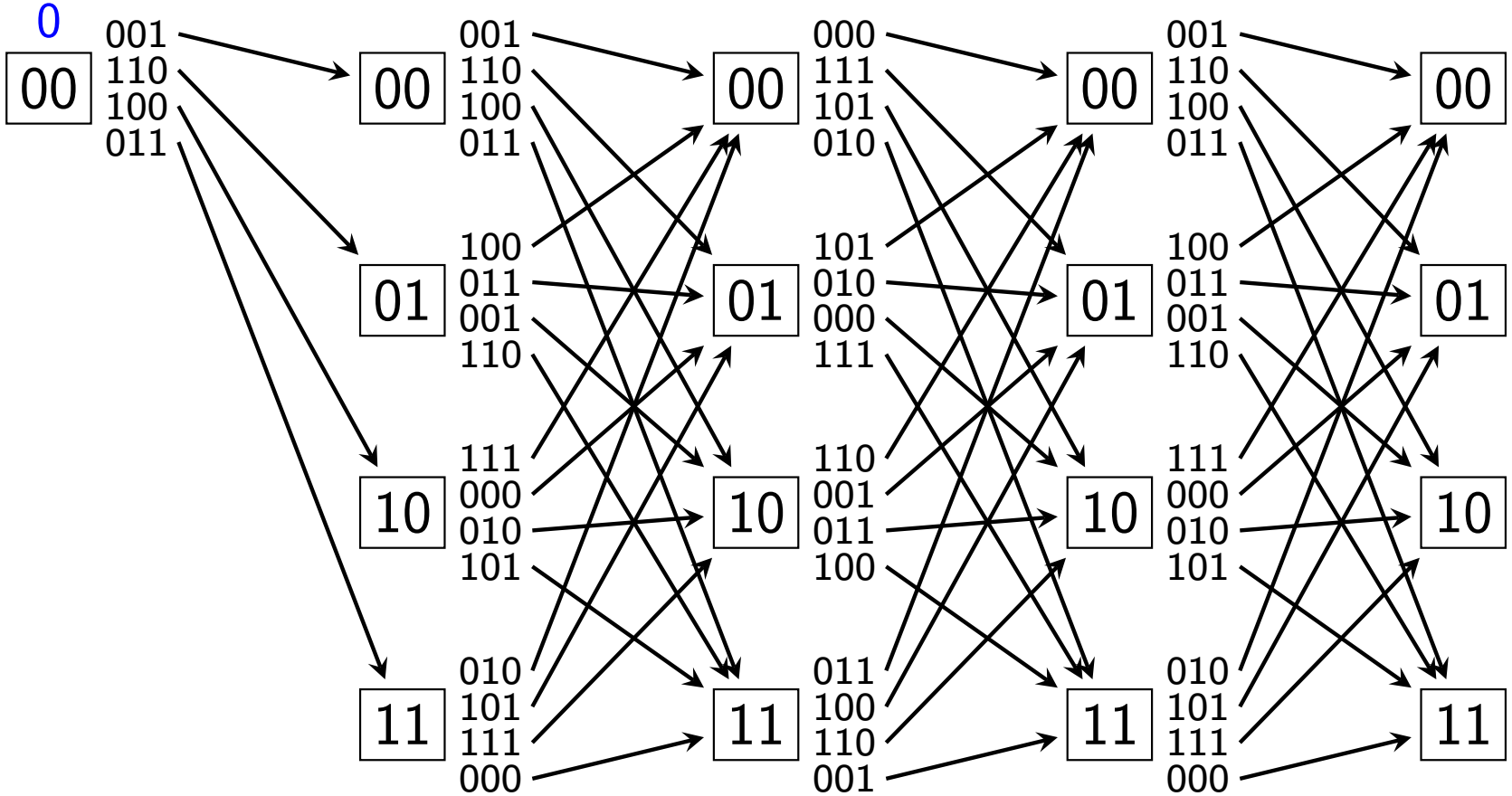
- Treláž se skládá z různých modulů v závislosti na zprávě.
- Nevyžadujeme, aby překladač skončil v nulovém stavu.
- Hrany označujeme vstupními hodnotami překladače místo výstupních hodnot.
  - Vkládač: Upravuje vstup, aby dosáhl požadovaný výstup.
  - Dekodér: Upravuje poškozený výstup, aby dosáhl platný výstup.
- Vstup překladače je delší než výstup.
  - Konvoluční extraktor: $n \geq m$.
  - Konvoluční kodér: $n \leq m$.

Viterbiho algoritmus

- V první fázi procházíme treláž zleva doprava, tj. pro $i = 0, \ldots, \ell - 1$.
  - Pro každé i máme až q^d možných stavů překladače.
  - Pro každý z těchto stavů spočítáme váhu nejlehčí cesty, která do něho vede, a zaznamenáme poslední hranu cesty.
- Jakmile dojdeme na konec treláže, vybereme stav, do kterého vede nejlehčí cesta.
- Zpětným průchodem tuto cestu zrekonstruujeme.
- Průběh algoritmu si předvedeme na příkladu nosiče $\{\mathbf{x}_i\}_{i=0}^{3} = \{(0,0,1)^{\mathrm{T}},\ (1,1,0)^{\mathrm{T}},\ (1,1,0)^{\mathrm{T}},\ (1,1,1)^{\mathrm{T}}\}$
- Používáme Hammingovu metriku.

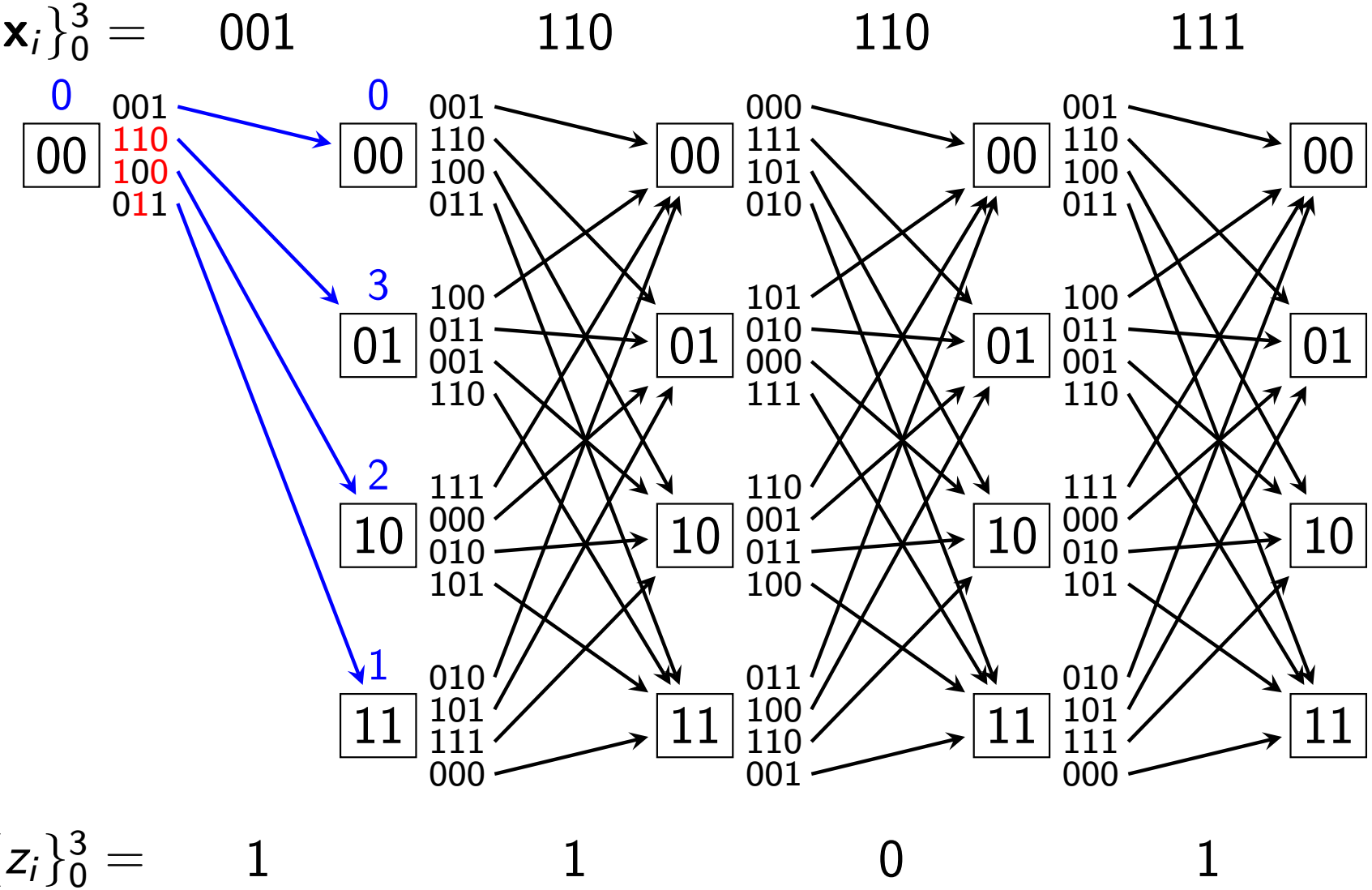
Viterbiho algoritmus

$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111



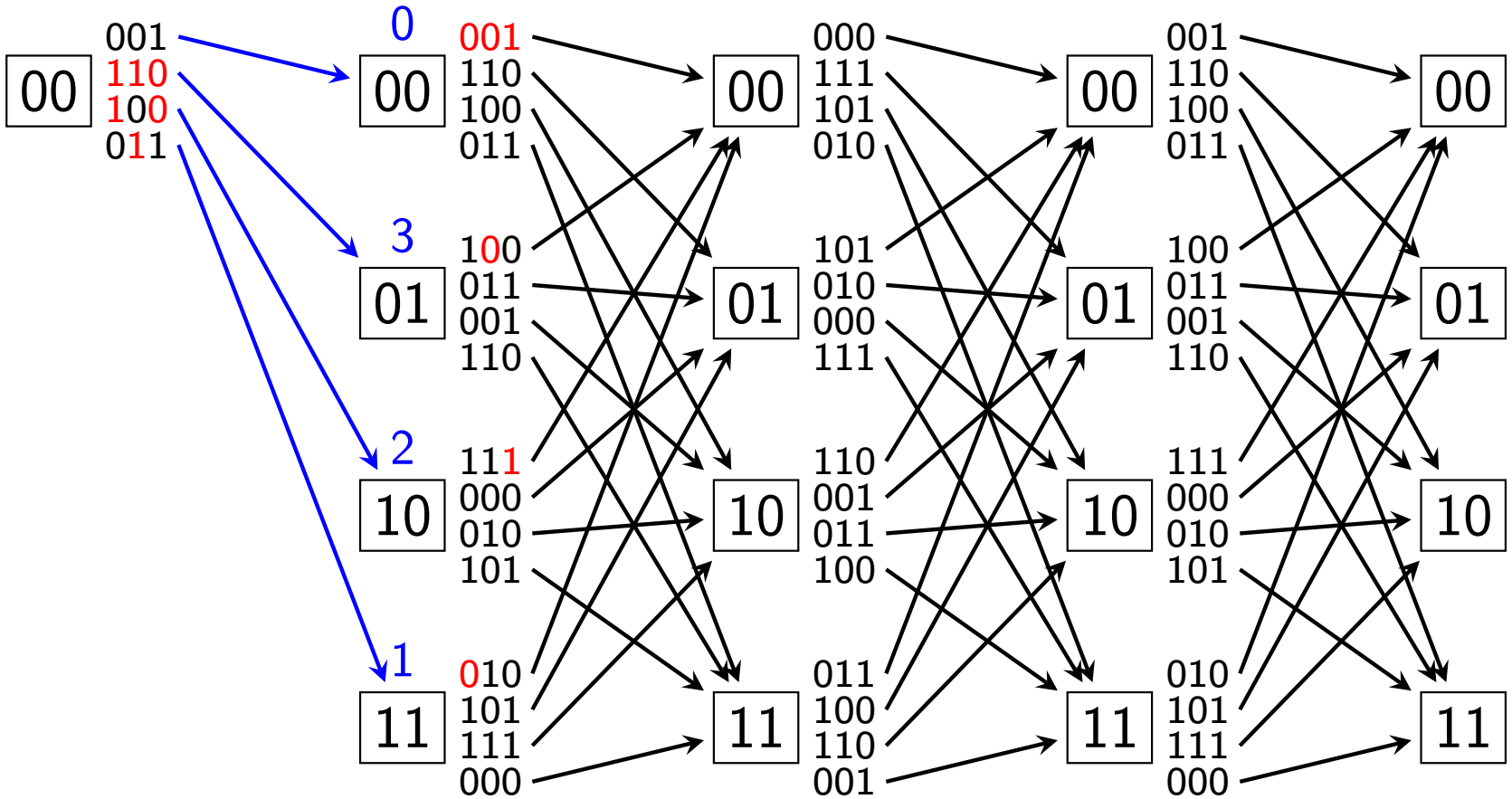
$\{z_i\}_0^3 =$ 1 1 0 1

Viterbiho algoritmus



Viterbiho algoritmus

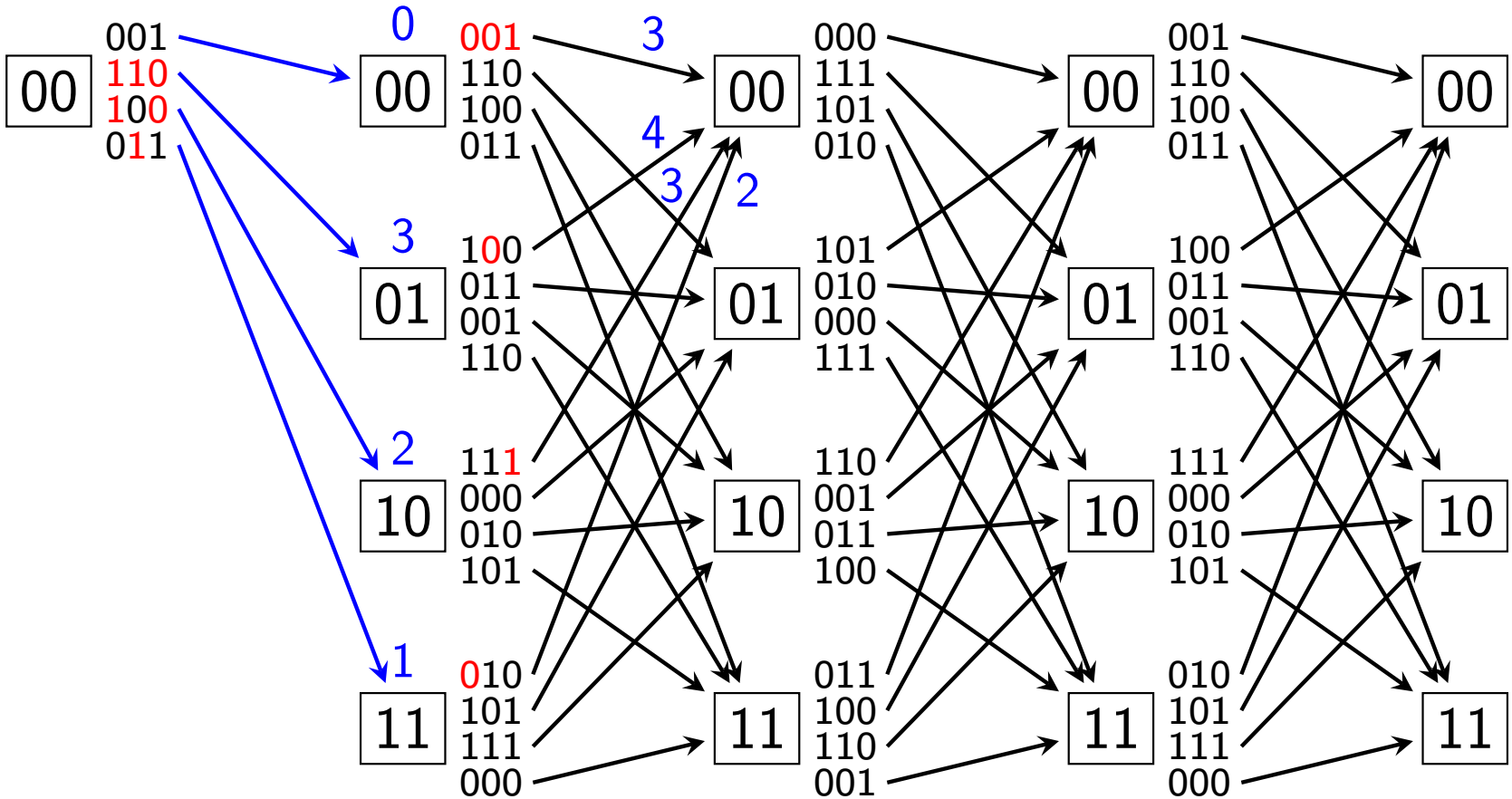
$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111



$\{z_i\}_0^3 =$ 1 1 0 1

Viterbiho algoritmus

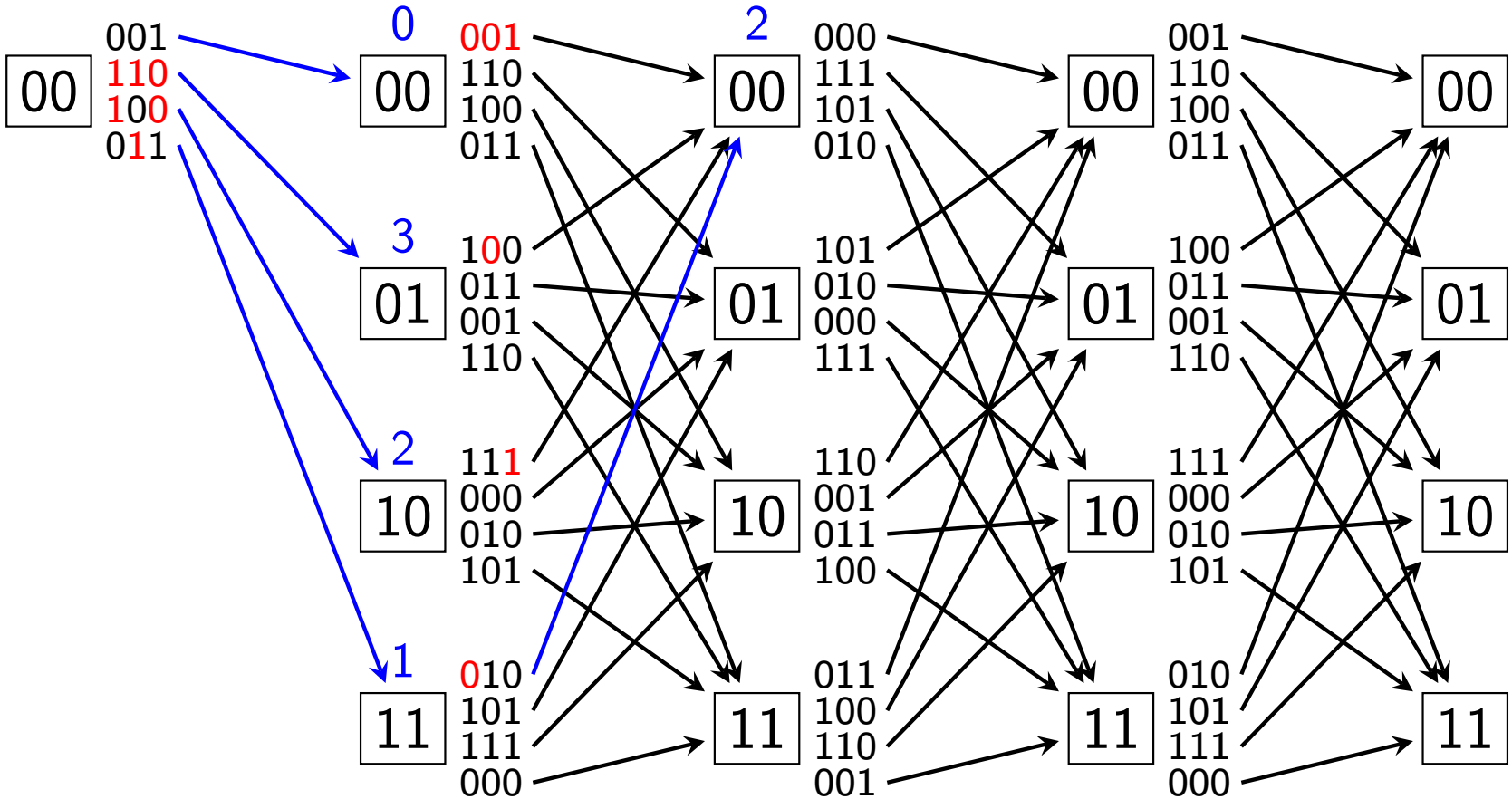
$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111



$\{z_i\}_0^3 =$ 1 1 0 1

Viterbiho algoritmus

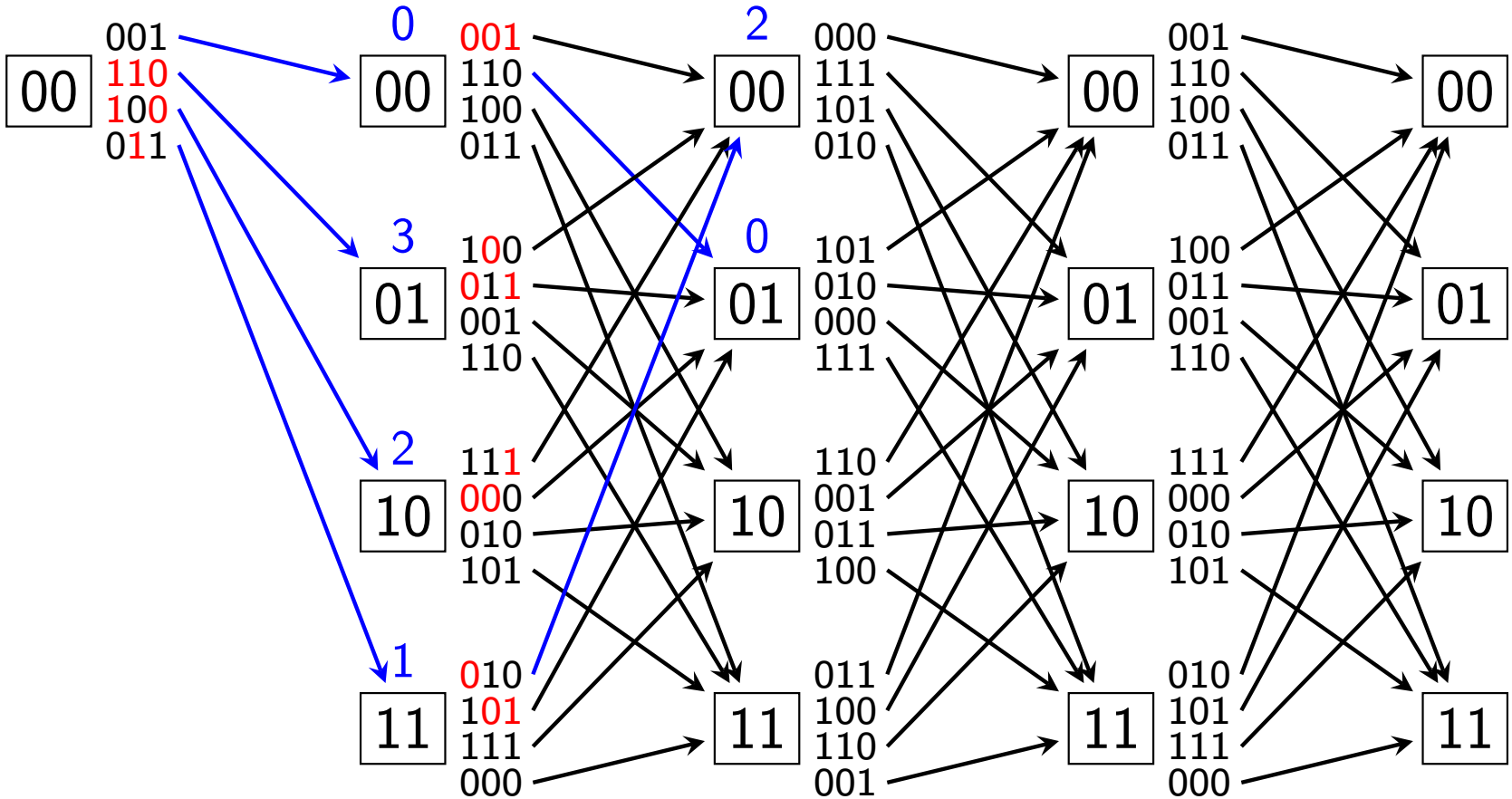
$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111



$\{z_i\}_0^3 =$ 1 1 0 1

Viterbiho algoritmus

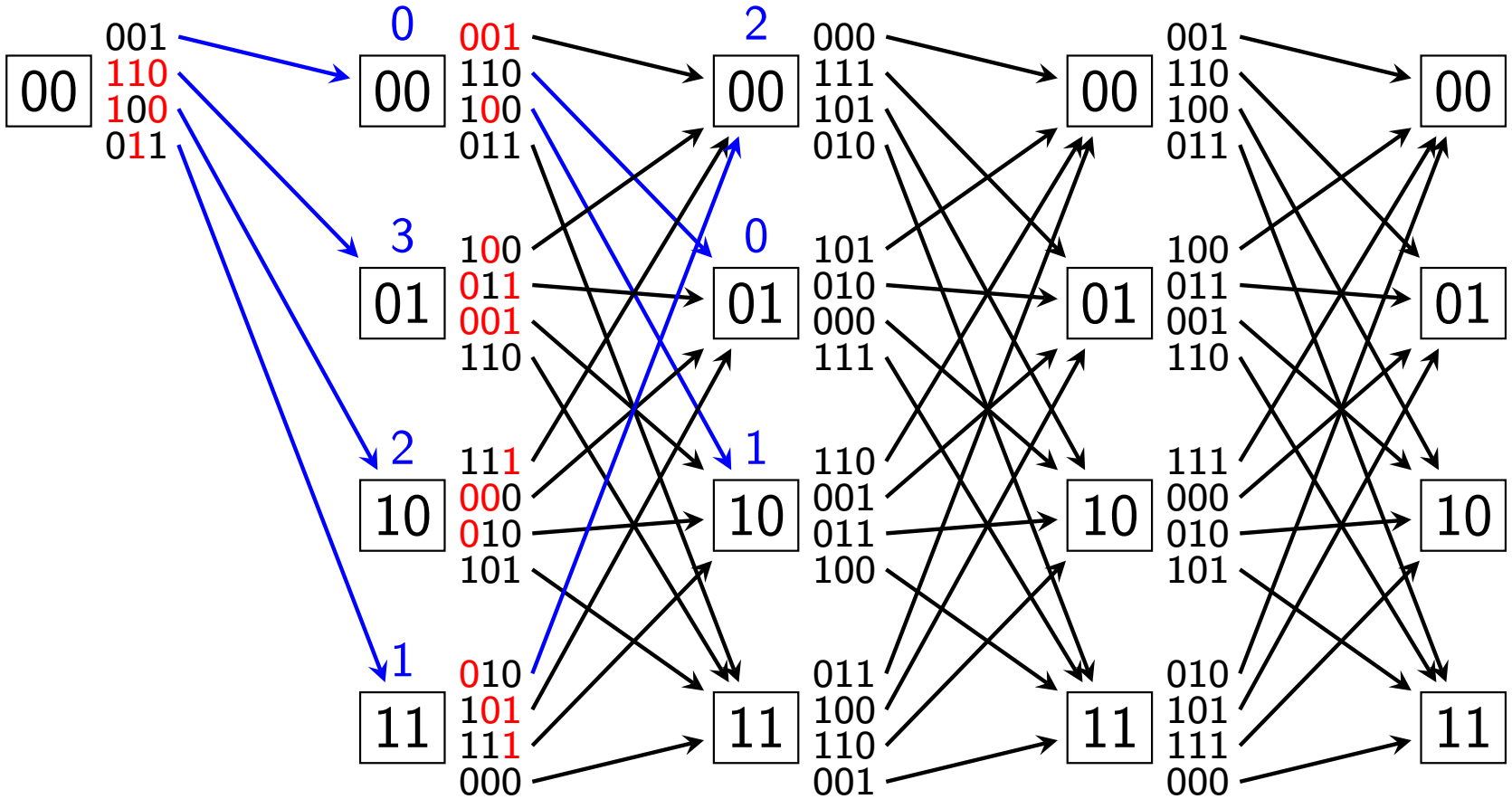
$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111



$\{z_i\}_0^3 =$ 1 1 0 1

Viterbiho algoritmus

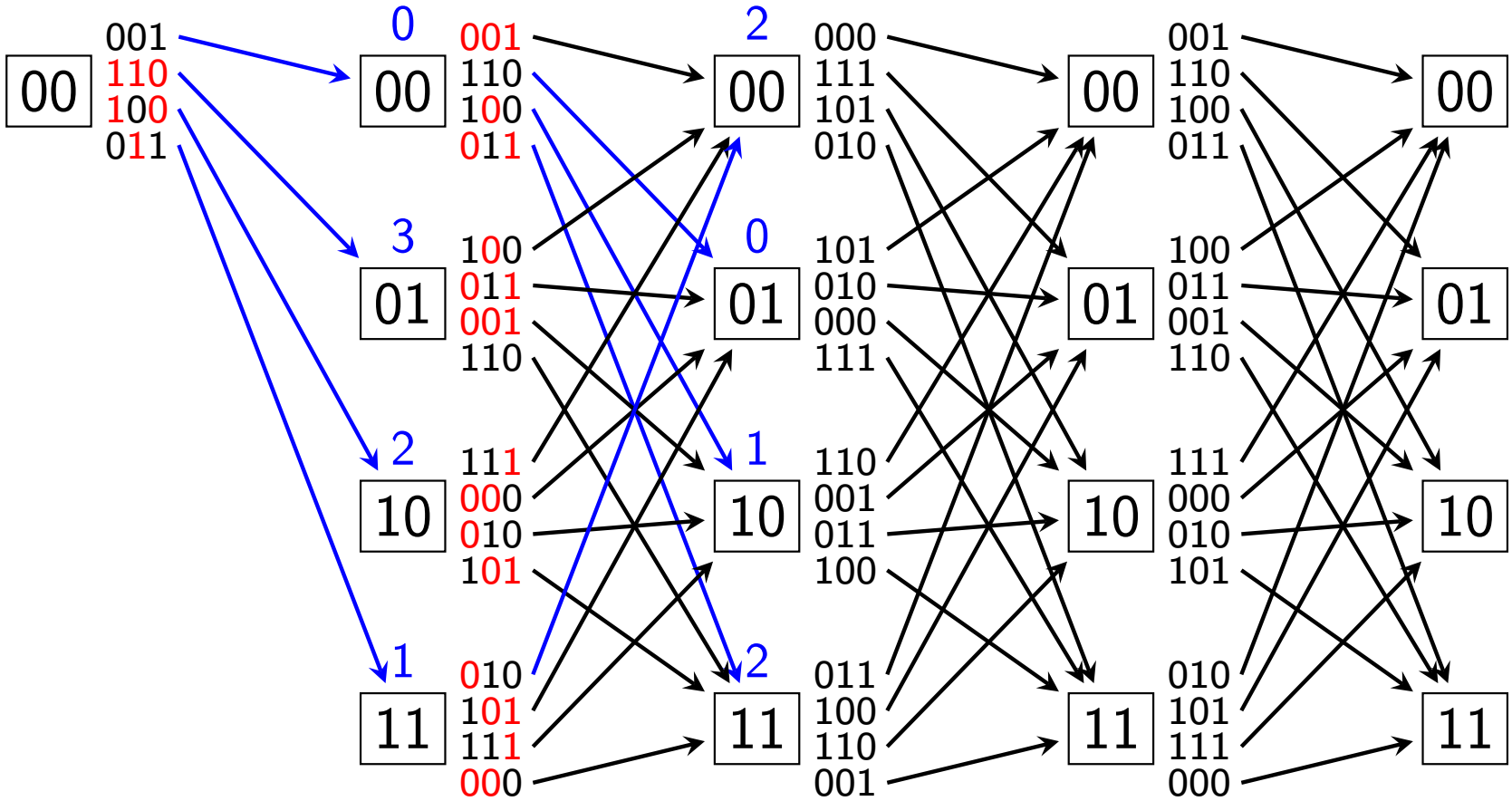
$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111



$\{z_i\}_0^3 =$ 1 1 0 1

Viterbiho algoritmus

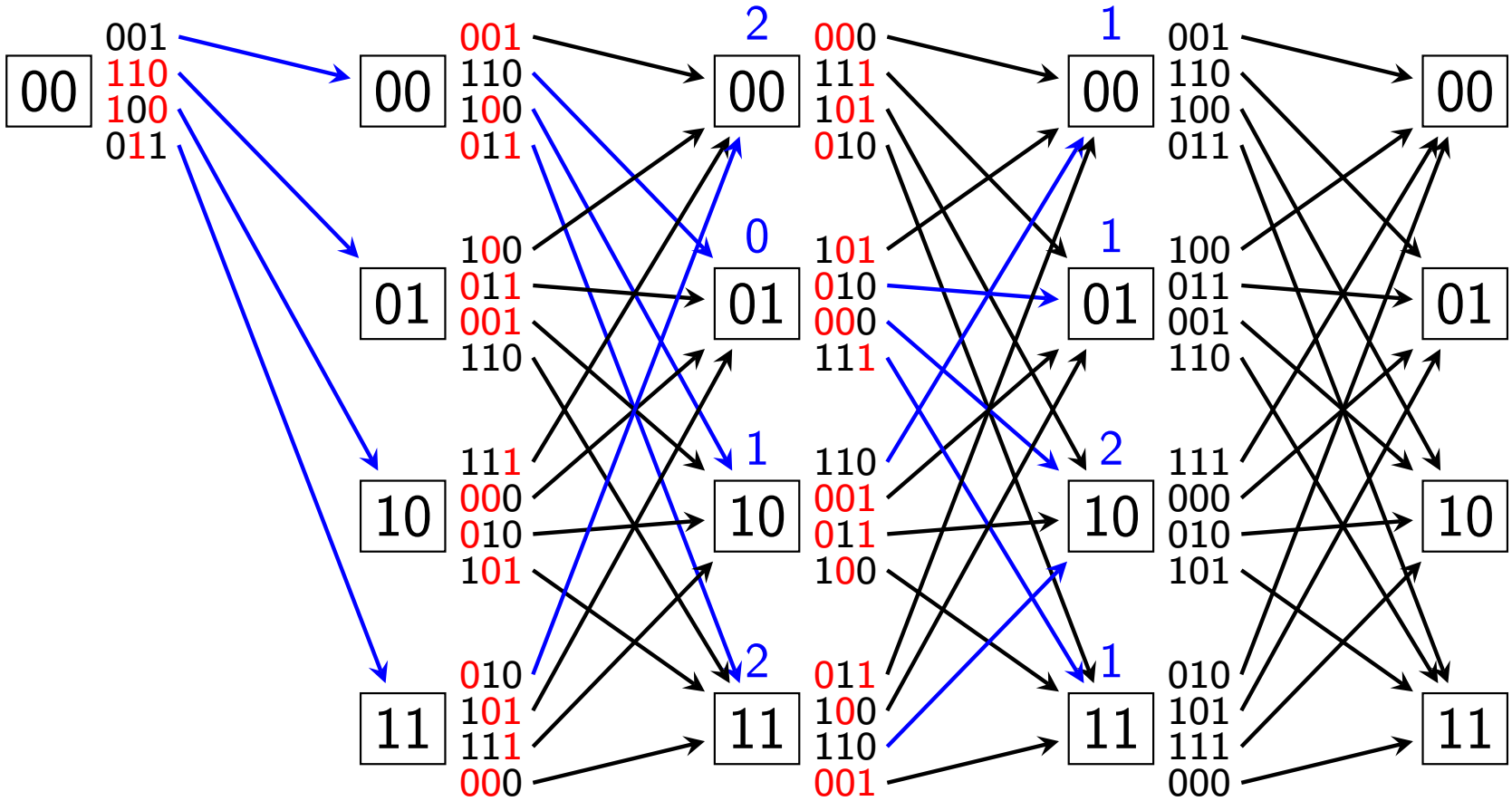
$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111



$\{z_i\}_0^3 =$ 1 1 0 1

Viterbiho algoritmus

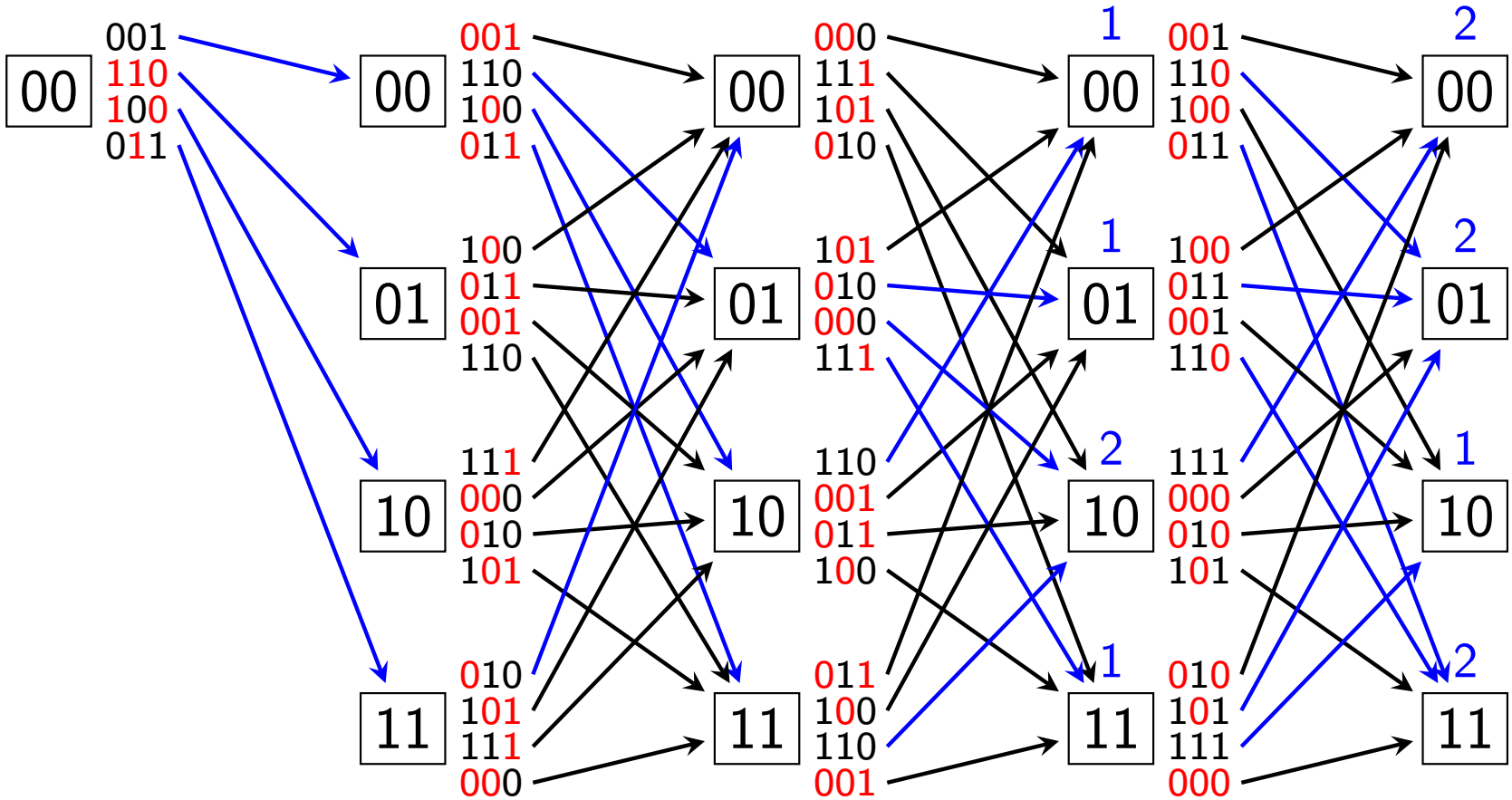
$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111



$\{z_i\}_0^3 =$ 1 1 0 1

Viterbiho algoritmus

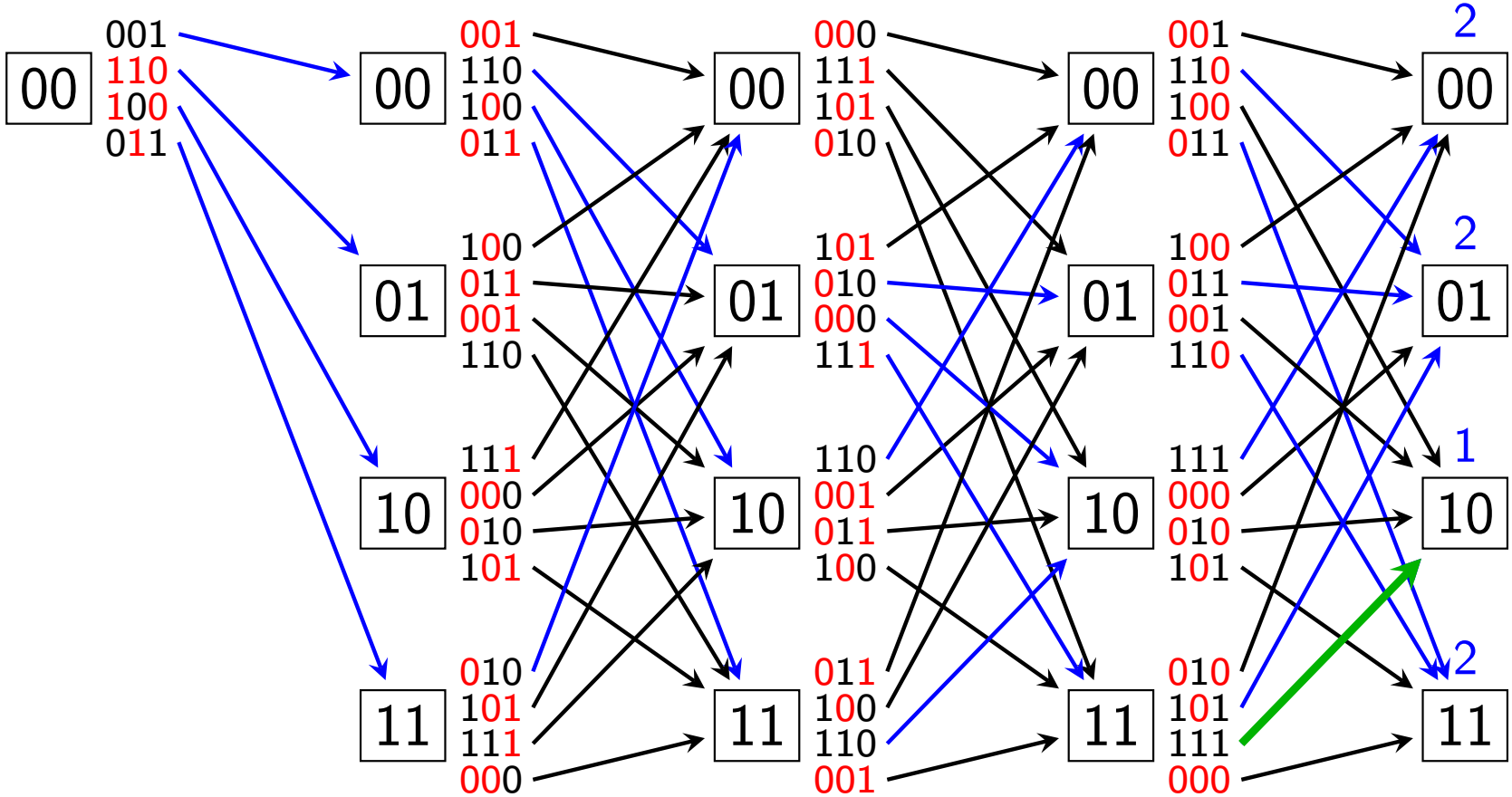
$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111



$\{z_i\}_0^3 =$ 1 1 0 1

Viterbiho algoritmus

$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111

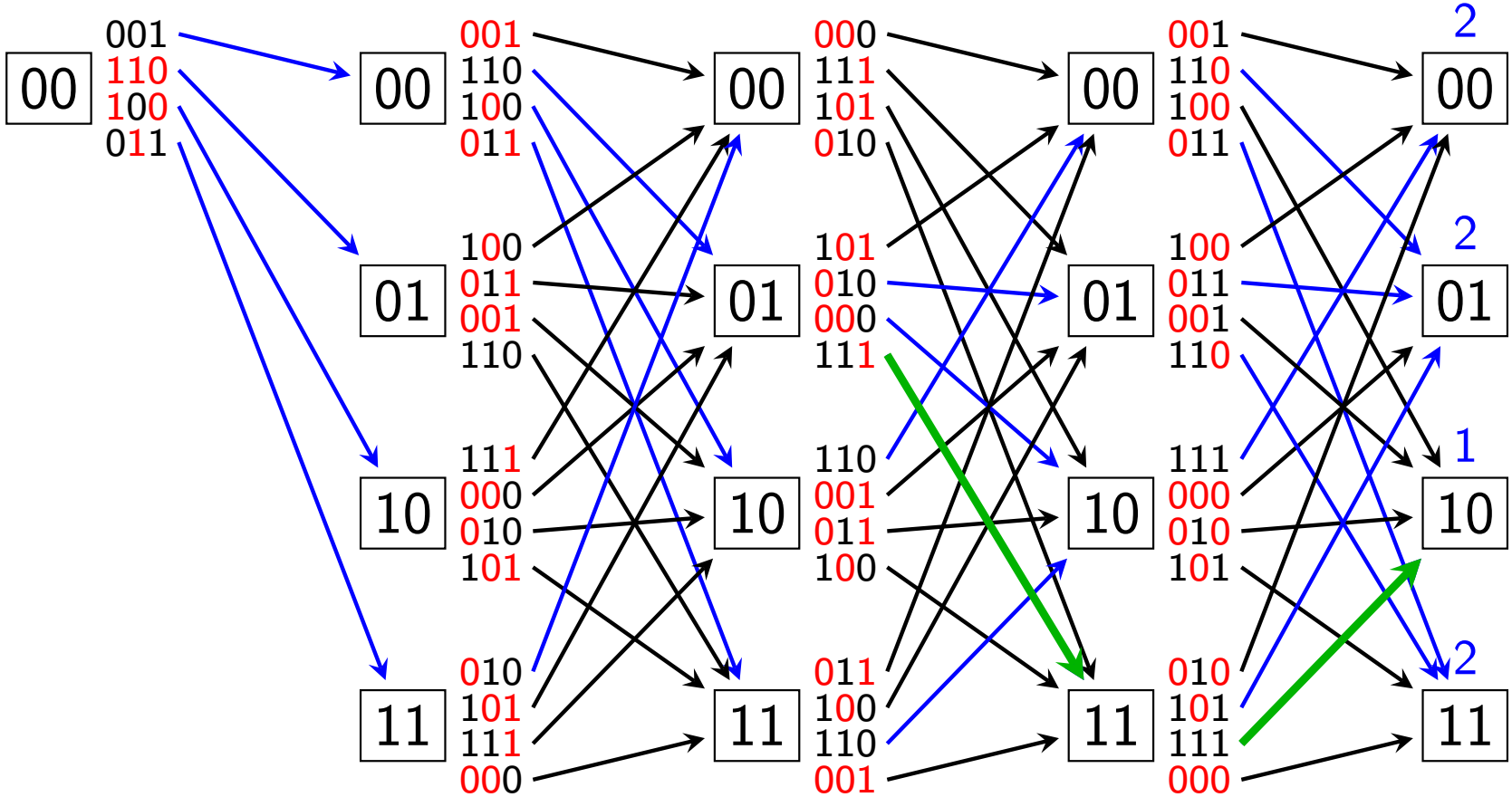


$\{z_i\}_0^3 =$ 1 1 0 1

$\{\mathbf{y}_i\}_0^3 =$ 111

Viterbiho algoritmus

$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111

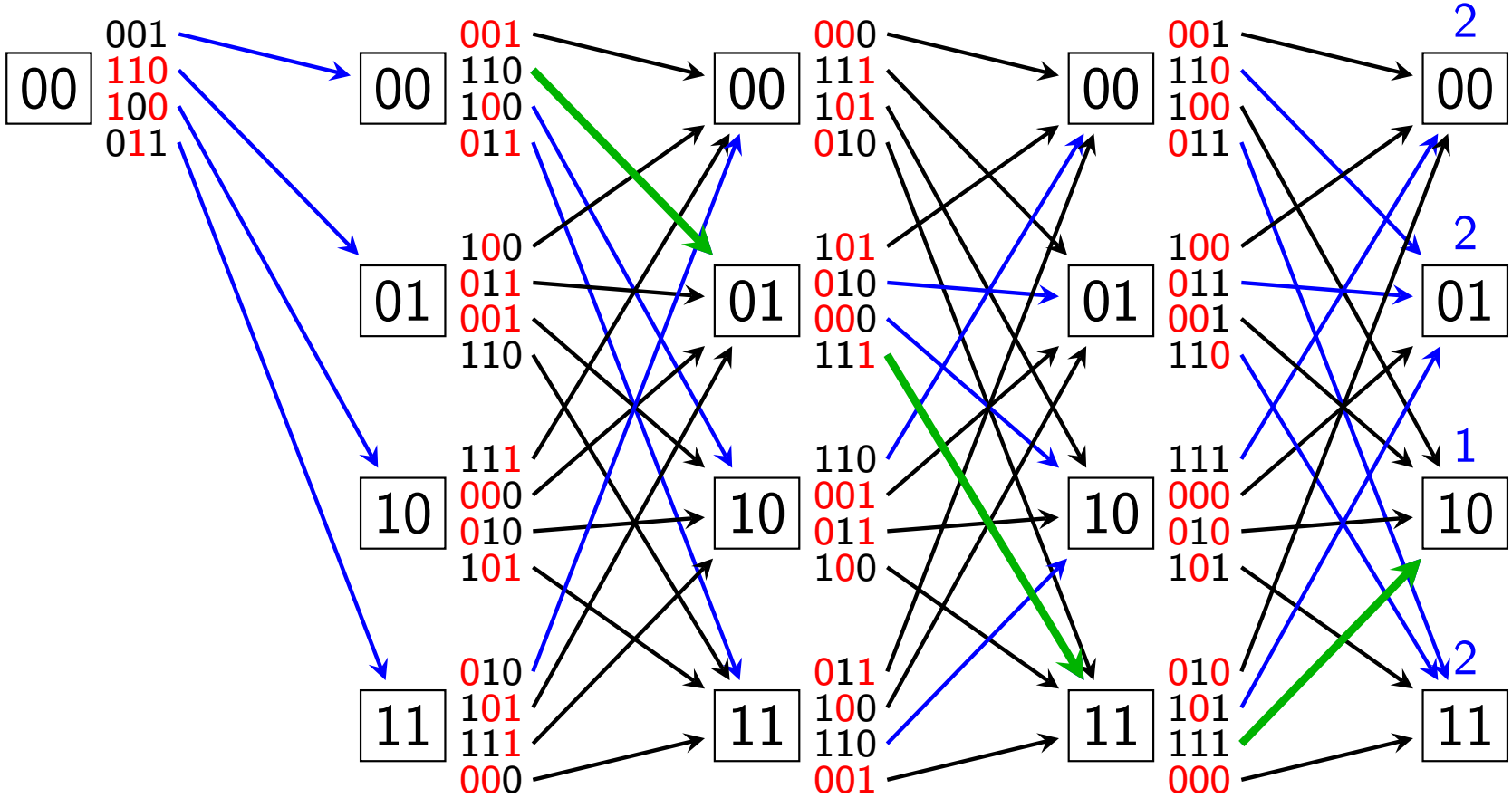


$\{z_i\}_0^3 =$ 1 1 0 1

$\{\mathbf{y}_i\}_0^3 =$ 111 111

Viterbiho algoritmus

$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111

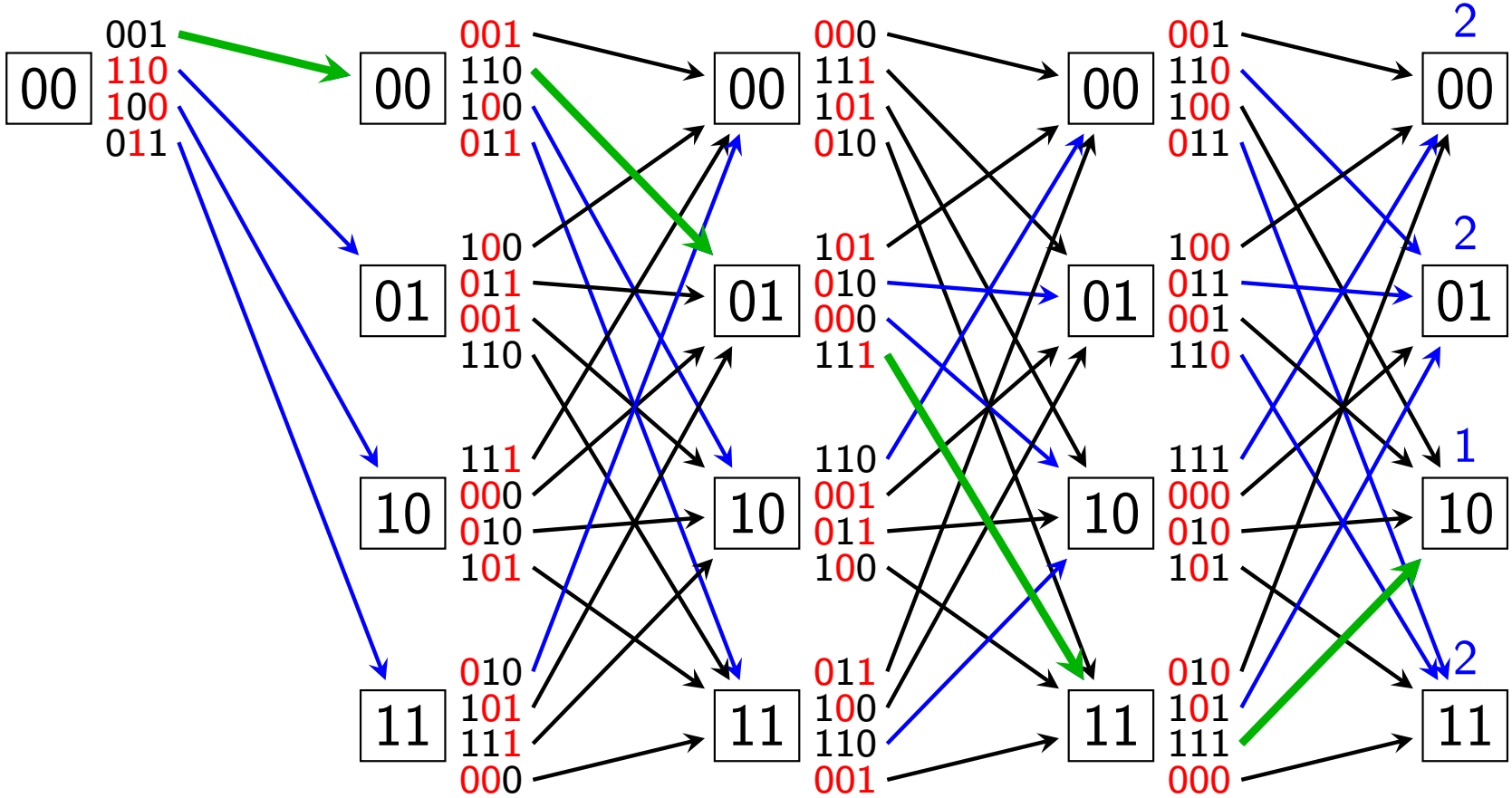


$\{z_i\}_0^3 =$ 1 1 0 1

$\{\mathbf{y}_i\}_0^3 =$ 110 111 111

Viterbiho algoritmus

$\{\mathbf{x}_i\}_0^3 =$ 001 110 110 111



$\{z_i\}_0^3 =$ 1 1 0 1

$\{\mathbf{y}_i\}_0^3 =$ 001 110 111 111

Časová složitost algoritmu

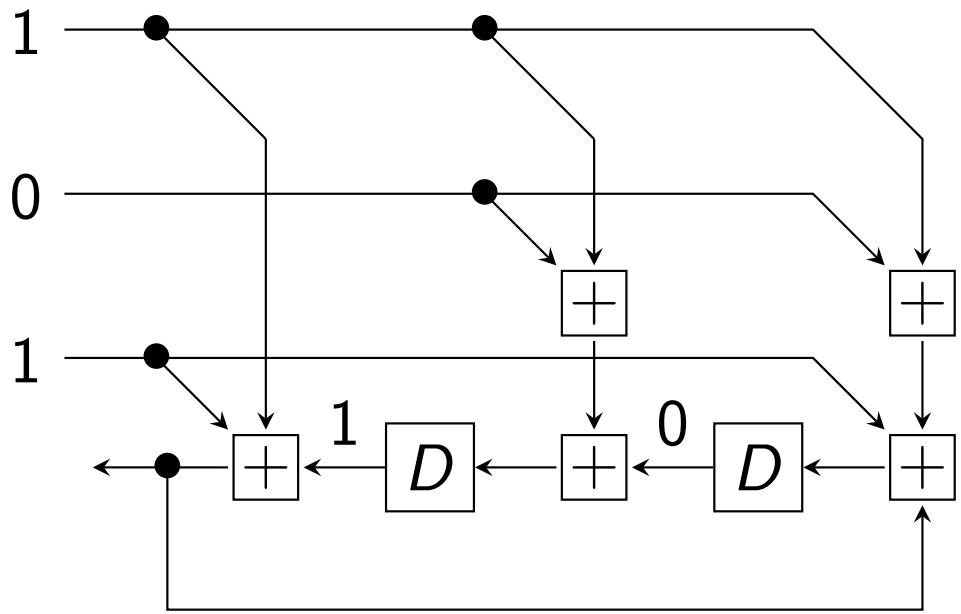
- Procházíme ℓ modulů zleva doprava.
- V každém modulu projdeme (až) q^d stavů a spočítáme váhu nejlehčí cesty, která do každého stavu vede.
  - Pro každý stav tedy zvážíme všechny vstupující hrany.
  - Do každého stavu vede v průměru $\frac{1}{q} \cdot q^n$ hran.
  - Zvážení jedné hrany vyžaduje $O(n)$ porovnání.
- Zpětný průchod má složitost $O(n)$ operací. (Zanedbatelné.)
- Časová složitost algoritmu je $O(q^{d+n-1}\ell n)$ porovnání.
- Složitost ještě vylepšíme na $O(q^{d+2}\ell n)$ vektorových operací tak, že rozvineme treláž.

Rozvinutí modulů

- Každý přechod automatu rozvineme na $n+1$ podkroků.
- Čili každý modul rozvineme na $n+1$ podmodulů.
- Připomeňme algoritmus konvoluční extrakce:

1 $(s_0, \ldots, s_d) := (0, \ldots, 0)$
2 **for** $i = 0, \ldots, \ell - 1$ **do**
3 **for** $j = 1, \ldots, n$ **do**
4 $(s_0, \ldots, s_d) := (s_0, \ldots, s_d) + y_i^{(j)}(f_0^{(j)}, \ldots, f_d^{(j)})$
5 $z_i := s_0$
6 $(s_0, \ldots, s_d) := (s_0, \ldots, s_d) + s_0(g_0, \ldots, g_d)$
7 $(s_0, \ldots, s_d) := (s_1, \ldots, s_d, 0)$
8 **return** $\{z_i\}_{i=0}^{\ell-1}$

Příklad: Rozvinutí přechodu automatu

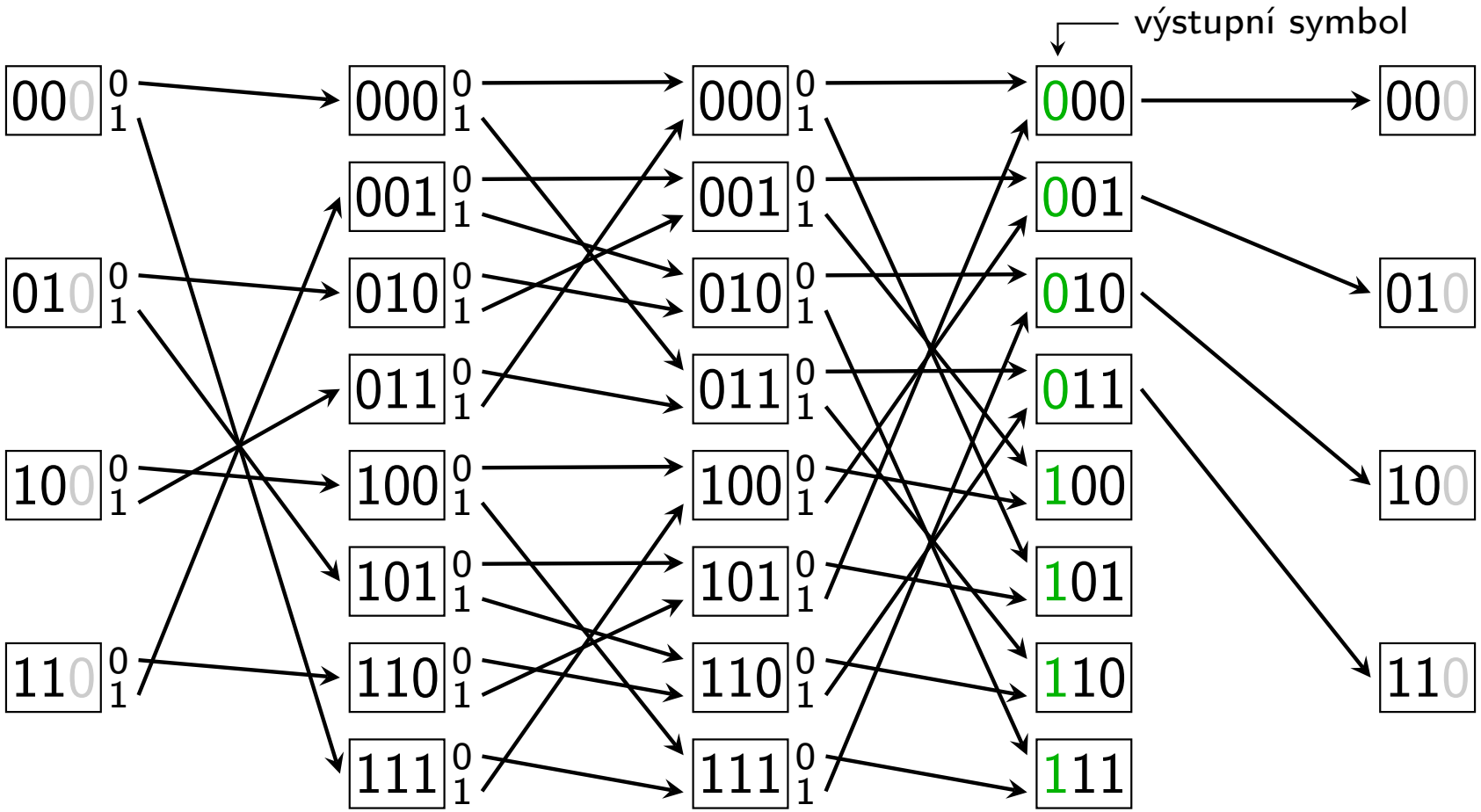


Na počátku máme $(s_0, s_1, s_2) = (\mathbf{1}, \mathbf{0}, 0)$. Podkroky pak jsou:

1. $(s_0, s_1, s_2) := (s_0, s_1, s_2) + \mathbf{1}(1, 1, 1) = (0, 1, 1)$
2. $(s_0, s_1, s_2) := (s_0, s_1, s_2) + \mathbf{0}(0, 1, 1) = (0, 1, 1)$
3. $(s_0, s_1, s_2) := (s_0, s_1, s_2) + \mathbf{1}(1, 0, 1) = (1, 1, 0)$
4. $(s_0, s_1, s_2) := (s_0, s_1, s_2) + 1(0, 0, 1) = (1, 1, 1)$

   $(s_0, s_1, s_2) := (s_1, s_2, 0) = (1, 1, 0)$

Rozvinutí modulu s výstupem 0



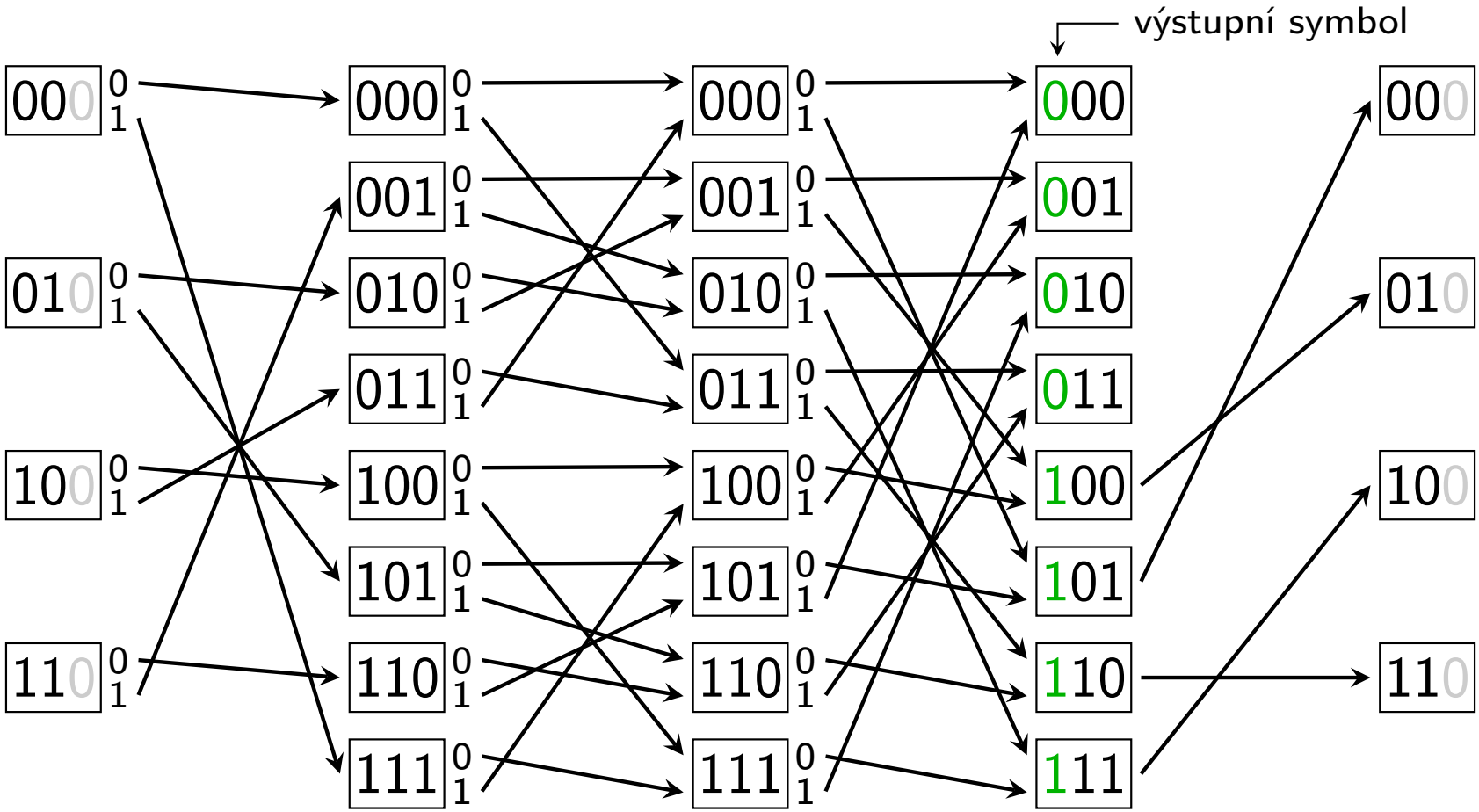
Přičtení $y_i^{(1)}(1,1,1)$ pro $y_i^{(1)} \in \mathbb{F}_2$.

Přičtení $y_i^{(2)}(0,1,1)$ pro $y_i^{(2)} \in \mathbb{F}_2$.

Přičtení $y_i^{(3)}(1,0,1)$ pro $y_i^{(3)} \in \mathbb{F}_2$.

Posunutí registru.

Rozvinutí modulu s výstupem 1



Přičtení $y_i^{(1)}(1,1,1)$ pro $y_i^{(1)} \in \mathbb{F}_2$.

Přičtení $y_i^{(2)}(0,1,1)$ pro $y_i^{(2)} \in \mathbb{F}_2$.

Přičtení $y_i^{(3)}(1,0,1)$ pro $y_i^{(3)} \in \mathbb{F}_2$.

Přičtení zpětné vazby $(0,0,1)$ a posunutí registru.

Rozvinutá treláž a distrozní funkce

- Trelážové moduly nahradíme jejich rozvinutými verzemi.
- Viterbiho algoritmus najde cestu rozvinutou treláží, která se nejvíce „podobá“ nosiči.
- Algoritmus umí zohlednit váhy jednotlivých změn v nosiči.
- Ke každému prvku nosiče $x_i^{(j)}$ přiřazujeme distorzní funkci $\rho_i^{(j)} : \mathbb{F}_q \to \mathbb{R}$.
- Hodnota $\rho_i^{(j)}(a)$ udává, jak moc by přispělo nastavení $y_i^{(j)} = a$ ve stegoobjektu k celkové distorzi vyvolané vkládáním.

Příklady distrozních funkcí

- Chceme-li Hammingovu metriku, definujeme

$$\rho_i^{(j)}(a) = \begin{cases} 0, & \text{jestliže } a = x_i^{(j)}, \\ 1, & \text{jestliže } a \neq x_i^{(j)}. \end{cases}$$

- Pro psaní na mokrý papír definujeme

$$\rho_i^{(j)}(a) = \begin{cases} \infty, & \text{jestliže prvek na } j\text{-té pozici v } i\text{-tém bloku} \\ & \text{je mokrý a zároveň } a \neq x_i^{(j)}, \\ 0 & \text{jinak.} \end{cases}$$

- Pro vkládání při kvantizaci definujeme

$$\rho_i^{(j)}(a) = (\text{distorze při vložení } a) - (\text{distorze při zaokrouhlení}).$$

Algoritmus (vkládání pomocí Viterbiho algoritmu)

vstup: distorzní funkce prvků nosiče $\{\boldsymbol{\rho}_i\}_{i=0}^{\ell-1}$, zpráva $\{z_i\}_{i=0}^{\ell-1}$ nad $\mathbb{F}_q$, parametry $f_j^{(k)} \in \mathbb{F}_q$ a $g_j \in \mathbb{F}_q$

výstup: stegoobjekt $\{\mathbf{y}_i\}_{i=0}^{\ell-1}$

— Inicializace.

1 **for** $\mathbf{s} \in \mathbb{F}_q^{d+1} \setminus \{\mathbf{0}\}$ **do**

2 $\quad w[\mathbf{s}] := \infty$

3 $w[\mathbf{0}] := 0$

— Dopředný průchod treláží.

4 **for** $i = 0, \ldots, \ell - 1$ **do**

5 $\quad$ **for** $j = 1, \ldots, n$ **do**

6 $\quad\quad$ **for** $\mathbf{s} \in \mathbb{F}_q^{d+1}$ **do**

7 $\quad\quad\quad path_i^{(j)}[\mathbf{s}] := \arg\min_{a \in \mathbb{F}_q} w[\mathbf{s} - a(f_0^{(j)}, \ldots, f_d^{(j)})] + \rho_i^{(j)}(a)$

8 $\quad\quad\quad w'[\mathbf{s}] := \min_{a \in \mathbb{F}_q} w[\mathbf{s} - a(f_0^{(j)}, \ldots, f_d^{(j)})] + \rho_i^{(j)}(a)$

9 $\quad\quad w := w'$

10 $\quad$ **for** $(s_0, \ldots, s_{d-1}) \in \mathbb{F}_q^d$ **do**

11 $\quad\quad w'[(s_0, \ldots, s_{d-1}, 0)] := w[(z_i, s_0, \ldots, s_{d-1}) - z_i(g^{(1)}, \ldots, g^{(n)})]$

12 $\quad\quad$ **for** $s_d \in \mathbb{F}_q \setminus \{0\}$ **do**

13 $\quad\quad\quad w'[(s_0, \ldots, s_d)] := \infty$

14 $\quad w := w'$

Algoritmus (pokračování)

— Volba stavu, ve kterém končí nejlehčí cesta.

15 $(s_0, \ldots, s_d) := \arg\min_{\mathbf{s}' \in \mathbb{F}_q^{d+1}} w[\mathbf{s}']$

— Rekonstrukce nejlehčí cesty zpětným průchodem treláže.

16 **for** $i = \ell - 1, \ldots, 0$ **do**

17 $\quad (s_0, s_1, \ldots, s_d) := (z_i, s_0, \ldots, s_{d-1}) - z_i(g_1, \ldots, g_n)$

18 $\quad$ **for** $j = n, \ldots, 1$ **do**

19 $\quad\quad y_i^{(j)} := path_i^{(j)}[(s_0, \ldots, s_d)]$

20 $\quad\quad (s_0, \ldots, s_d) = (s_0, \ldots, s_d) - y_i^{(j)}(f_0^{(j)}, \ldots, f_d^{(j)})$

21 **return** $\{y_i\}_{i=0}^{\ell-1}$

- $w[\mathbf{s}]$ = váha nejlehčí cesty, která vede do $\mathbf{s}$.
- $path_i^{(j)}[\mathbf{s}]$ = poslední hrana nejlehčí cesty, která vede do $\mathbf{s}$.

Složitost algoritmu

- Nejčastěji jsou volány instrukce na řádcích 7 a 8, celkem $(\ell n q^{d+1})$-krát.
- Instrukce na řádku 7 provádí q-krát:
  - vektorovou operaci (nejdražší),
  - nahlédnutí do paměti,
  - volání $\rho_i^{(j)}(a)$ (předpokládáme v konstantním čase),
  - porovnání.
- Časová složitost: $O(q^{d+2}\ell n)$ vektorových operací.
- Obvykle stačí specializace algoritmu pro $q = 2$, kterou lze implementovat pomocí operací XOR.
- V poli *path* ukládáme $q^{d+1}\ell n$ prvků tělesa $\mathbb{F}_q$.
- Paměťová složitost: $O(q^{d+1}\ell n \log q)$ bitů.

Výsledky několika experimentů

- Stegosystém popsaný v příkladu dosahuje efektivity $e = 3{,}87$, přičemž $\alpha = 1/3$.
- Toto velmi dobře odpovídá efektivitě Hammingových kódů v dané oblasti.
- Použijeme-li větší registr lze dosáhnout lepších výsledků.
- Např. $d = 10$, pak pro $\alpha = 1/3$ lze dosáhnout efektivity $4{,}94$.
- Přítomnost zpětné vazby $g(D)$ nevedla k měřitelnému zlepšení efektivity vkládání.