CT-216 PROJECT

CONVOLUTION CODING

GROUP-20

PROF. YASH VASAVADA

TEAM MEMBERS

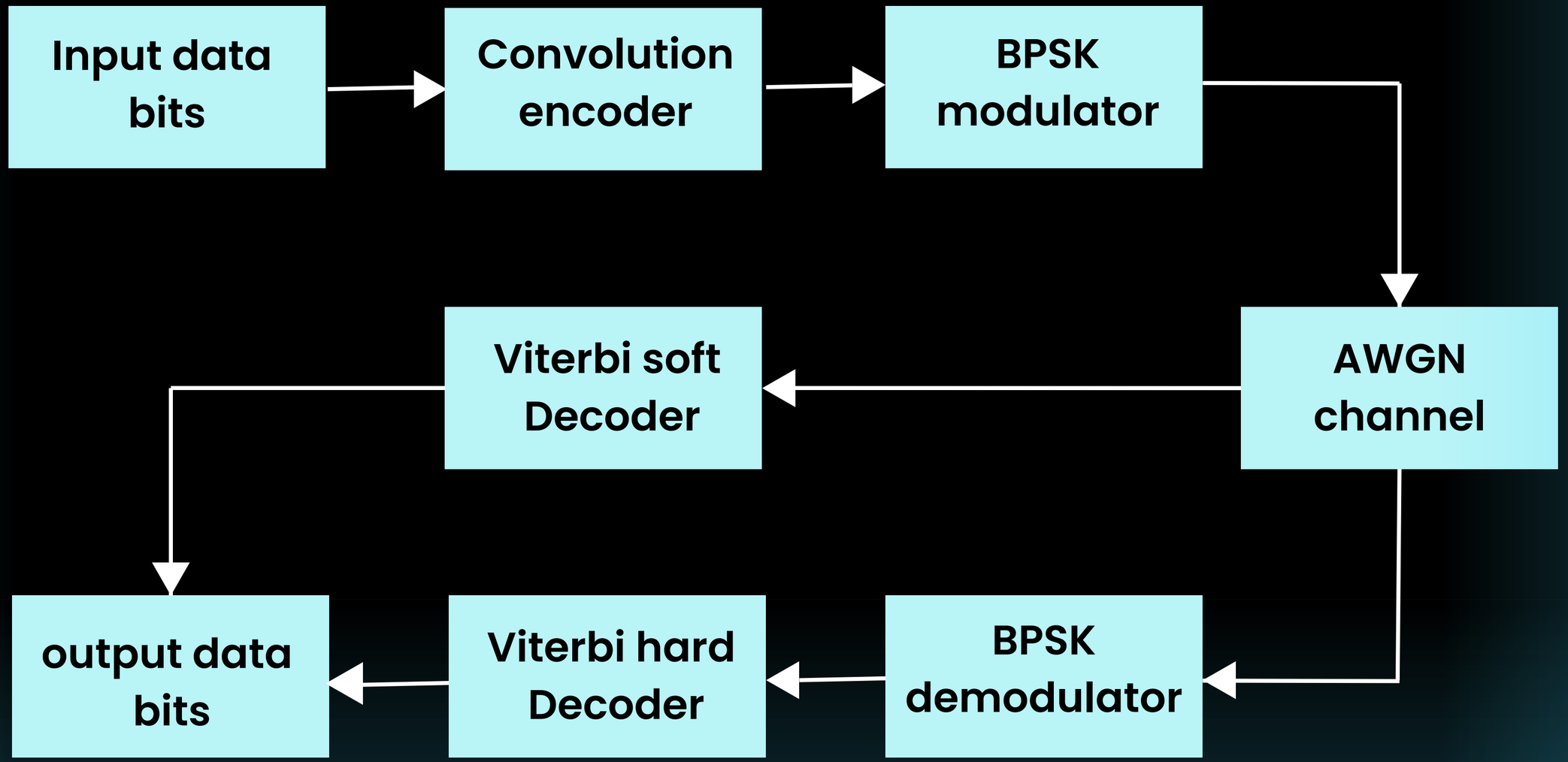
Harita rathod→202301211
Ishti patel →202301212
Prayag kalriya→202301213
Gautam modi → 202301214
Jay patoliya → 202301215
Archan maru→202301217
Meet gandhi → 202301219
Nandeesh trivedi→202301220
Ajaykumar rathod→202301221
Manan chhabhaya →202301222

HONOR CODE

We declare that:

- The work that we are presenting is our own work.We have not copied the work (the code, the results etc.) that someone else has done.
- Concepts, understanding and insights we will be describing are our own.
- We make this pledge truthfully. We know that violation of this solemn pledge can carry grave consequences

FLOW CHART


Input data bits
Convolution encoder
BPSK modulator
AWGN channel
Viterbi soft Decoder
output data bits
Viterbi hard Decoder
BPSK demodulator

CONVOLUTIONAL CODE

- A convolutional code is a type of error-correcting code used to detect and correct errors in data transmission or storage. It works by combining the current and previous input bits to generate encoded output bits, using a shift register and generator polynomials.
- It introduces memory into the encoding process by using past input bits, which improves error correction capability.
- Convolutional codes are typically decoded using the Viterbi algorithm, which is efficient for real-time decoding.

CONVOLUTION ENCODER

- A convolutional encoder is a device or algorithm used to implement convolutional codes, which add redundancy to data for error detection and correction.
- Memory-based: Uses past inputs to influence the output, giving it better error correction than memoryless codes.
- Rate: Defined as k/n where k is the number of input bits and n is the number of output bits per cycle.
- Constraint length: input bit + memory bits , defines how many bits affect output.

CONVOLUTION ENCODER

- The total length of the encoded output (code word) in convolutional coding is calculated using the formula:

Code Word Length = n(L + m)

Where :

- n = Number of output bits generated per input bit
- L = Length of the input data sequence (number of input bits)
- m = Number of memory elements (i.e., how many previous bits influence the output)

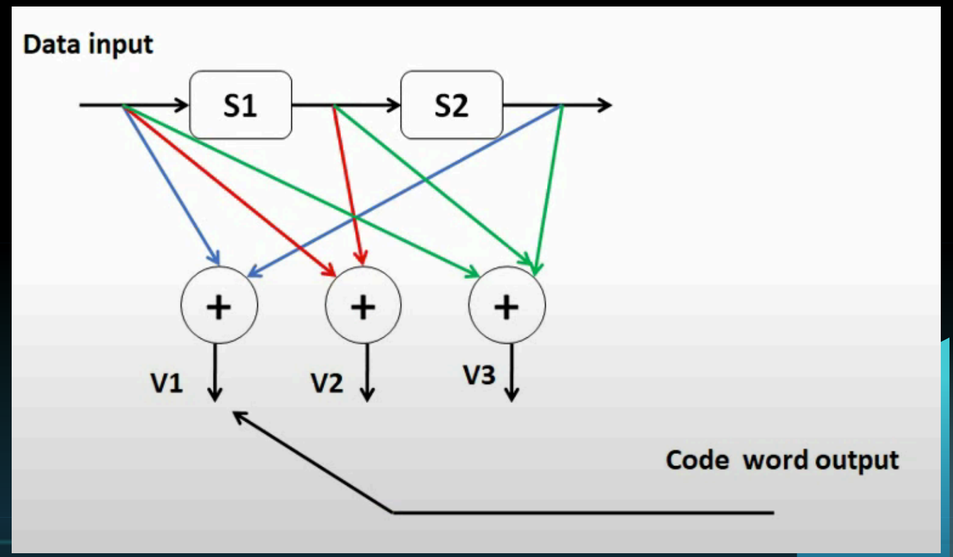
CONVOLUTION ENCODER

- Number of Zeros Appended
  - Append (Kc - 1) zeros to the input before encoding.
  - This ensures the encoder's memory is flushed.
  - Also written as m zeros, where:
  - m = number of memory elements = constraint length - 1

CONVOLUTION ENCODER

- Generator Polynomials define how input and memory bits are combined to produce encoded output bits.
- Example of Generator polynomials:

$g_1 = \{1, 0, 1\} \Rightarrow \text{output} = \text{input} \oplus S_2$
$g_2 = \{1, 1, 0\} \Rightarrow \text{output} = \text{input} \oplus S_1$
$g_3 = \{1, 1, 1\} \Rightarrow \text{output} = \text{input} \oplus S_1 \oplus S_2$



- Number of states = $2^{(Kc-1)}$

CONVOLUTION ENCODER

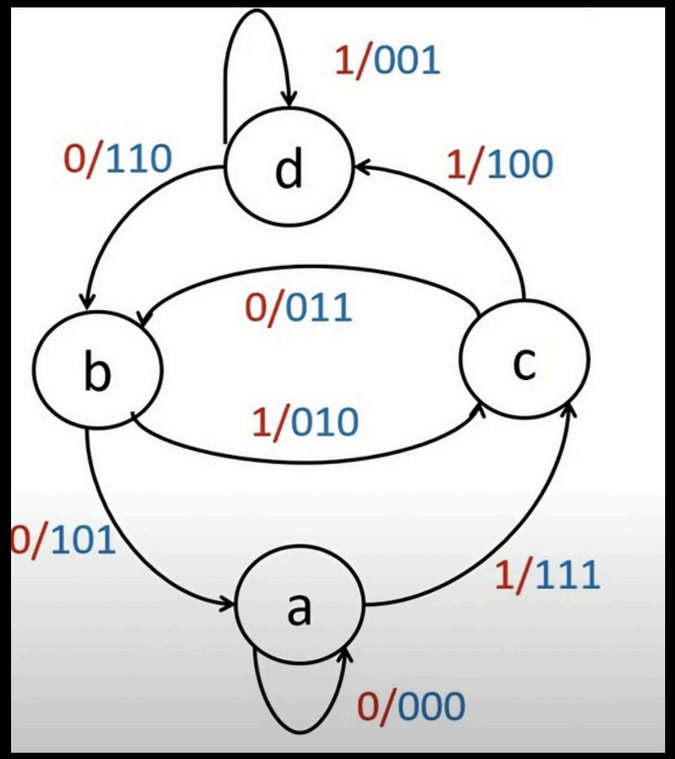
- Example of convolution code
- For an input data stream of [**1 0 0 1 1 0 0**]
  (with (Kc - 1) zeros added)
  Kc = 3 , n = 3
  $g_1 = \{1, 0, 1\}$
  $g_2 = \{1, 1, 0\}$
  $g_3 = \{1, 1, 1\}$

Data	S1	S2	v1	v2	v3
1	0	0	1	1	1
0	1	0	0	1	1
0	0	1	1	0	1
1	0	0	1	1	1
1	1	0	1	0	0
0	1	1	1	1	0
0	0	1	1	0	1

- Codeword
  [**111 011 101 111 100 110 101**]

STATE DIAGRAM

- Input data stream of [**1 0 0 1 1 0 0**]

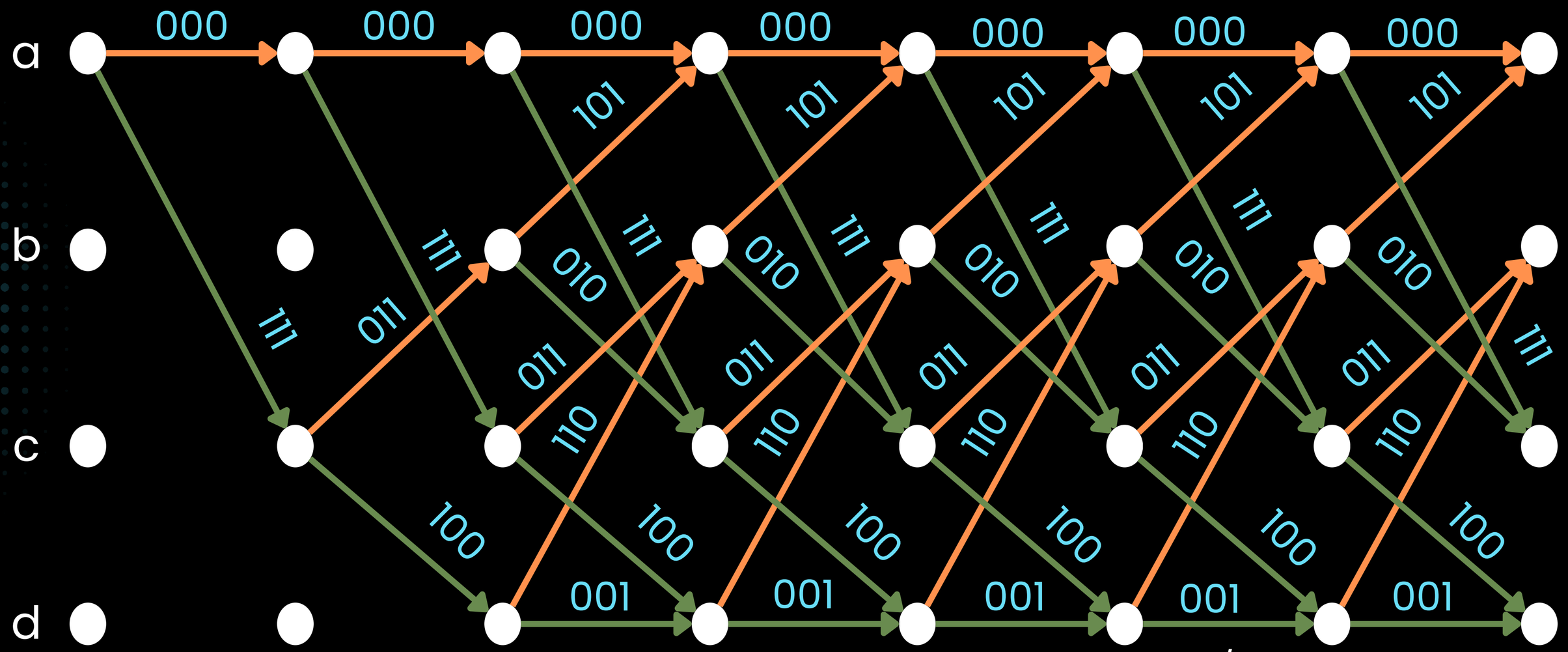


State	S1	S2
a	0	0
b	0	1
c	1	0
d	1	1

Input	Present state		Next state		Output		
data	S1	S2	S1	S2	V1	V2	V3
0	0	0	0	0	0	0	0
1	0	0	1	0	1	1	1
0	0	1	0	0	1	0	1
1	0	1	1	0	0	1	0
0	1	0	0	1	0	1	1
1	1	0	1	1	1	0	0
0	1	1	0	1	1	1	0
1	1	1	1	1	0	0	1

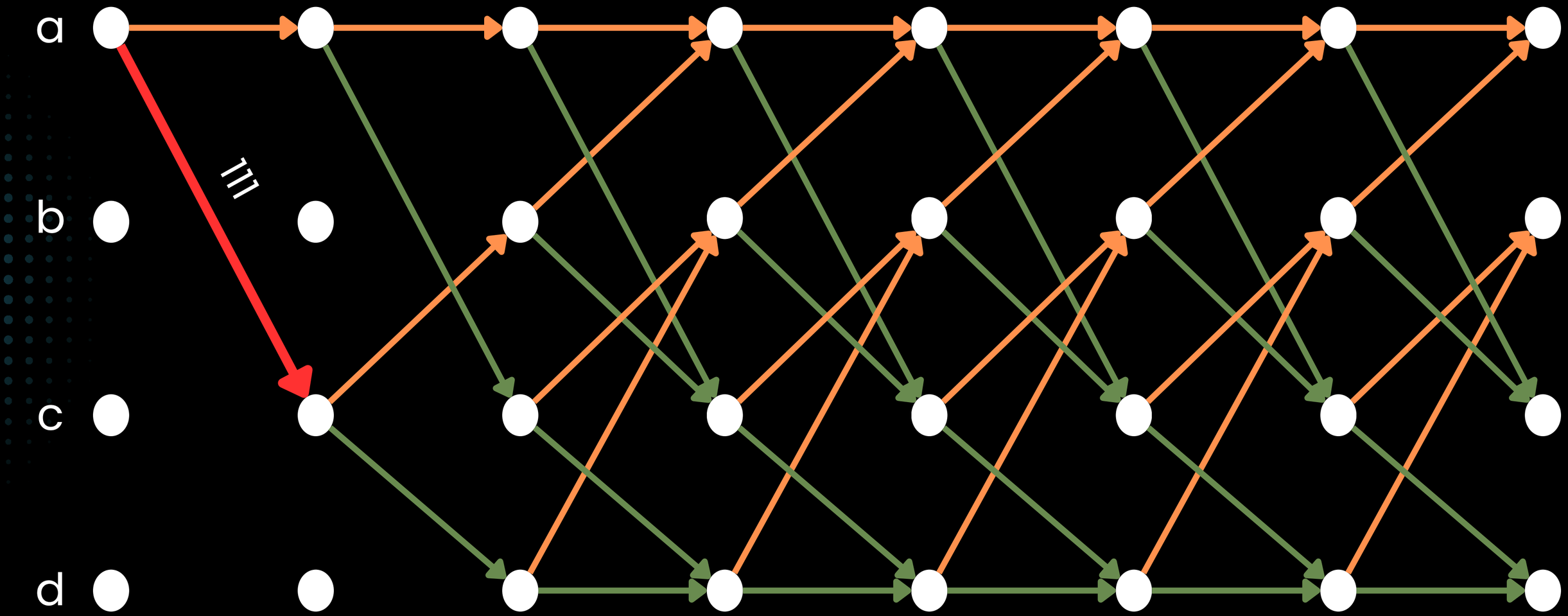
- Codeword
  [111 011 101 111 100 110 101]

TRELLIS DIAGRAM



Number of stages = L + m + 1

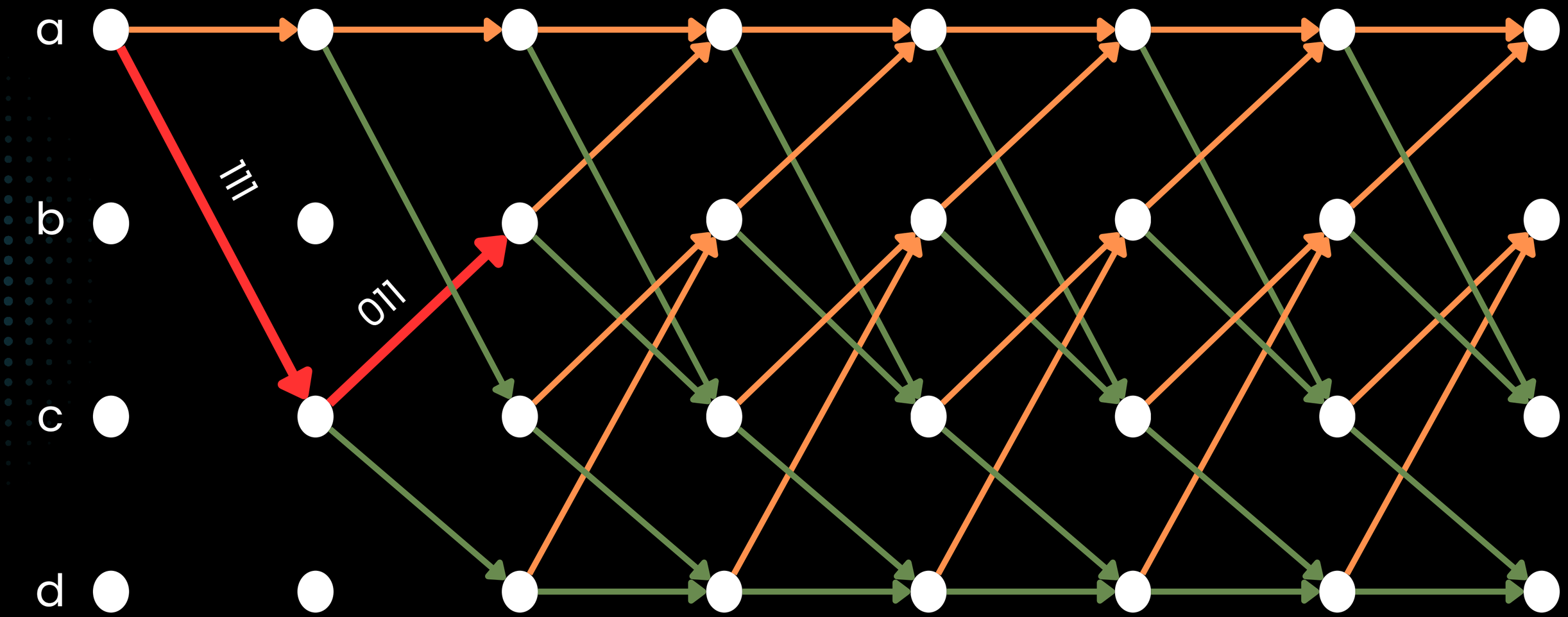
ENCODING



input data stream of [**1 0 0 1 1 0 0**]

codeword [**111** 011 101 111 100 110 101]

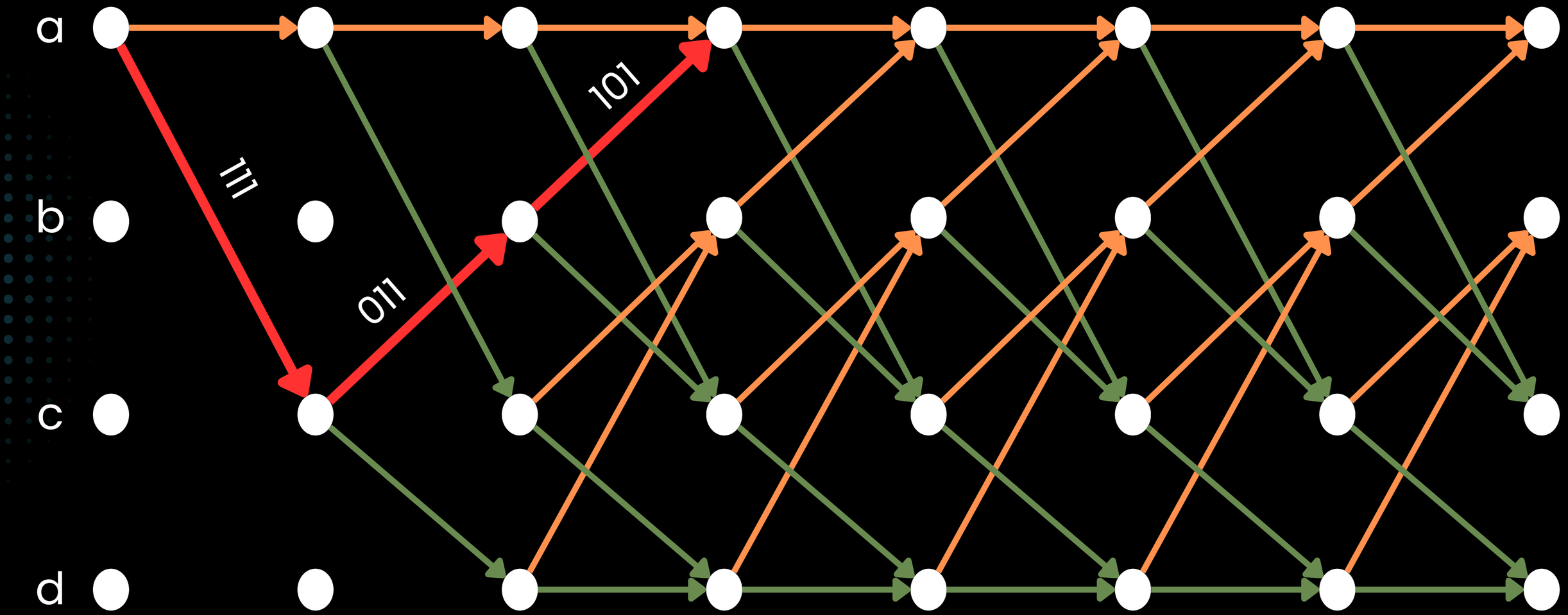
ENCODING



input data stream of [**1 0 0 1 1 0 0**]

codeword [**111 011** 101 111 100 110 101]

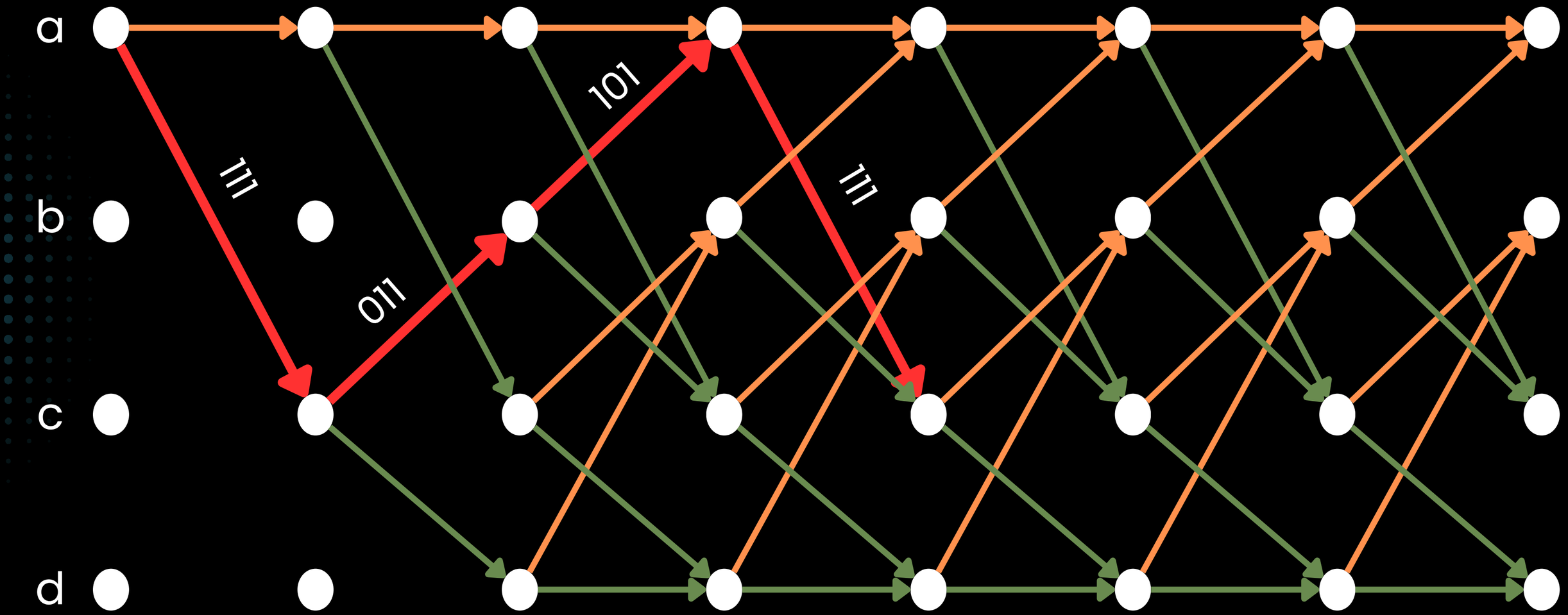
ENCODING



input data stream of [**1 0 0 1 1 0 0**]

codeword [**111 011 101** 111 100 110 101]

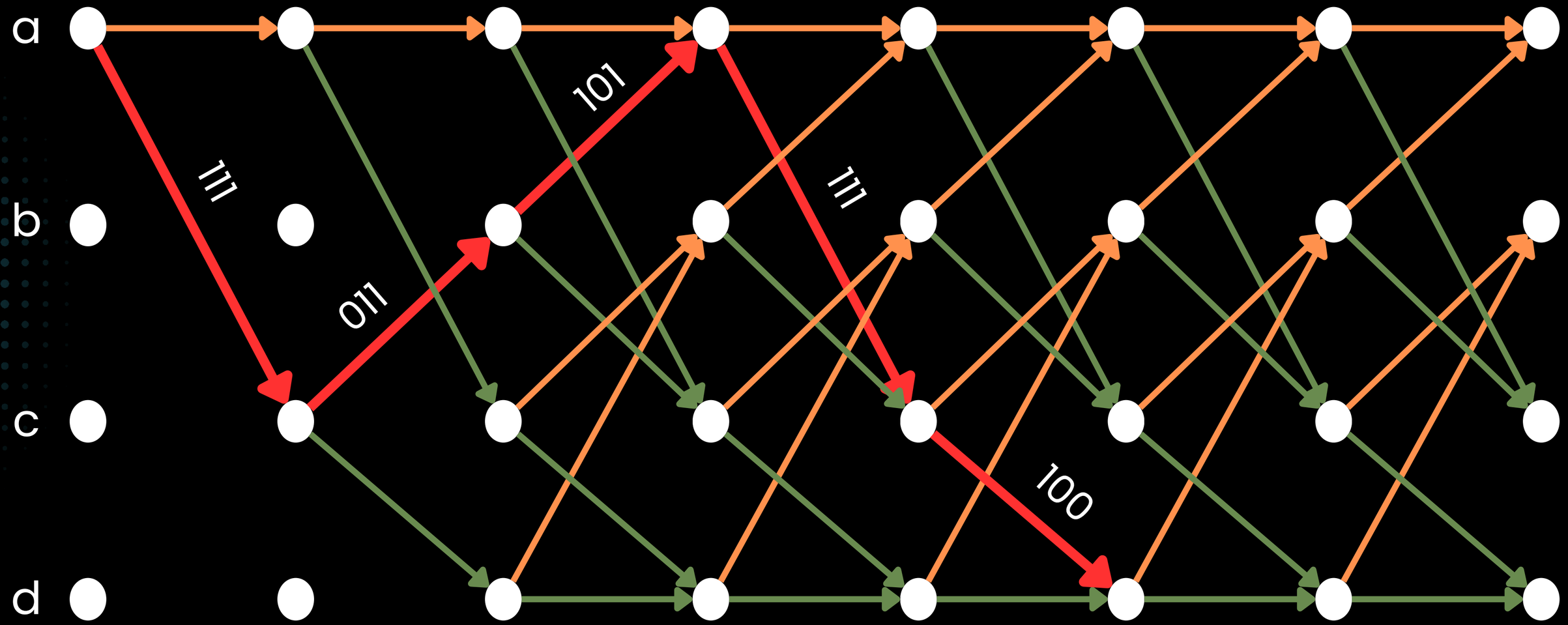
ENCODING



input data stream of [**1 0 0 1 1 0 0**]

codeword [**111 011 101 111** 1 0 0 1 1 0 1 0 1]

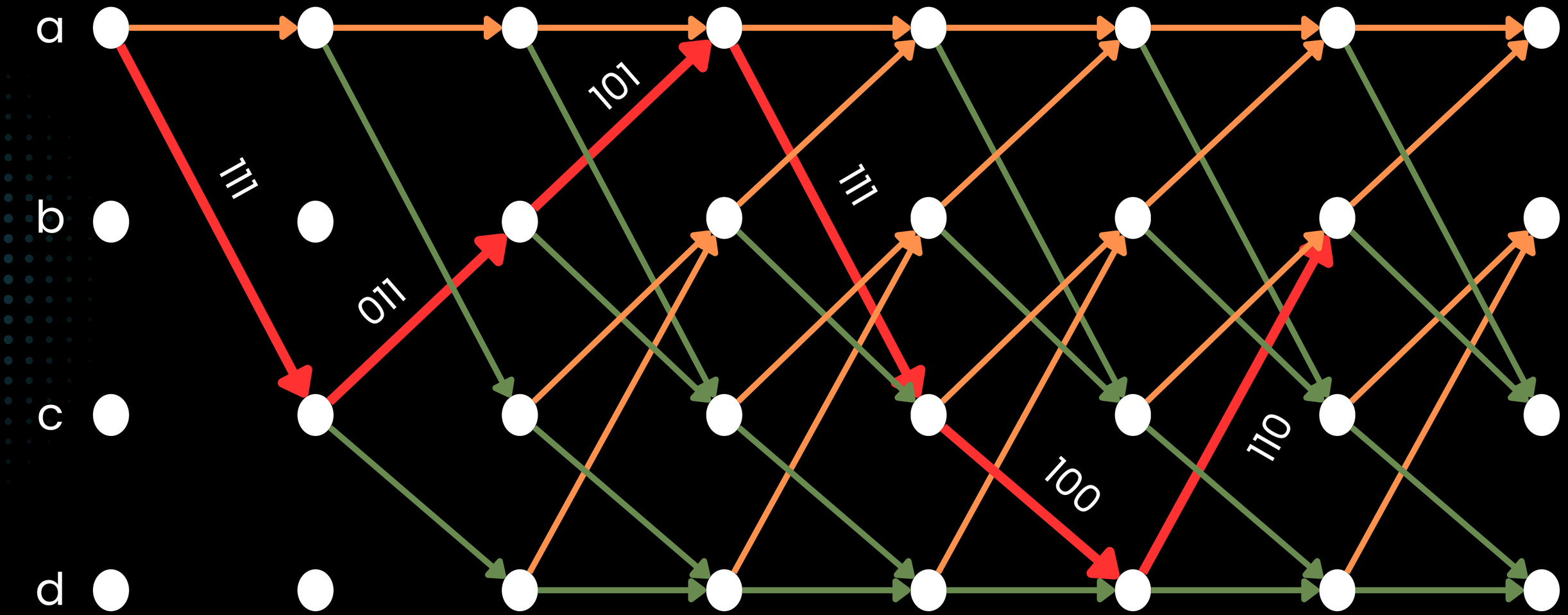
ENCODING



input data stream of [**1 0 0 1 1 0 0**]

codeword [**111 011 101 111 100** 110 101]

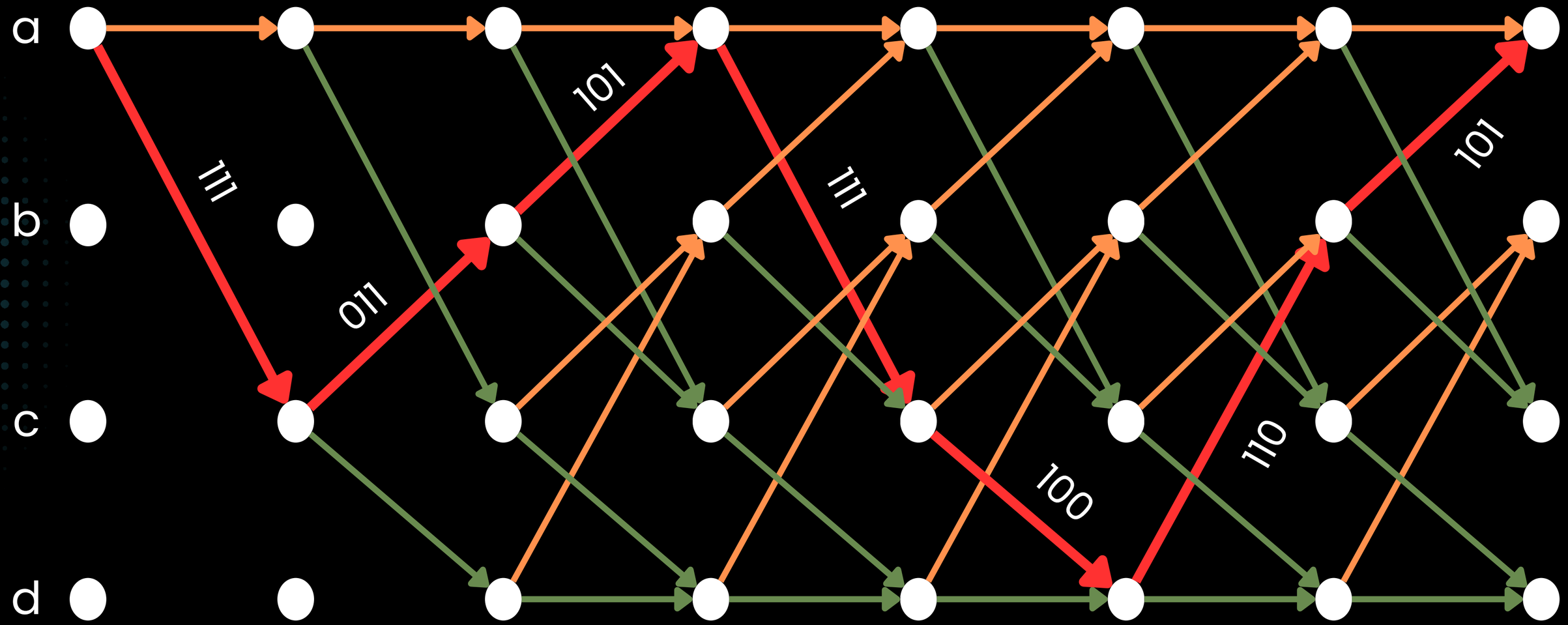
ENCODING



input data stream of [**1 0 0 1 1 0 0**]

codeword [**111 011 101 111 100 110** 101]

ENCODING

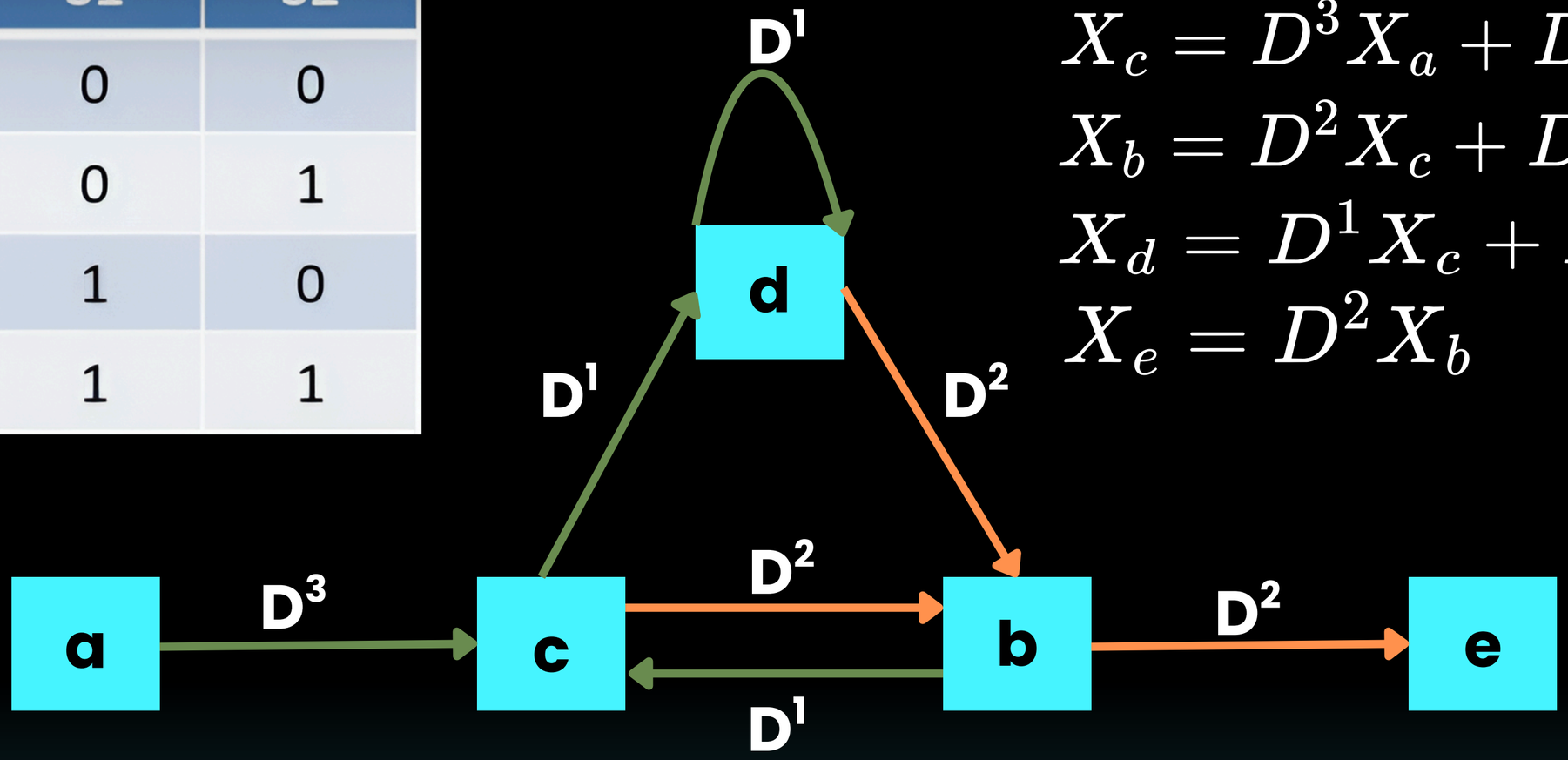


input data stream of [**1 0 0 1 1 0 0**]

codeword [**111 011 101 111 100 110 101**]

TRANSFER FUNCTION

State	S1	S2
a	0	0
b	0	1
c	1	0
d	1	1



$$X_c = D^3 X_a + D^1 X_b$$
$$X_b = D^2 X_c + D^2 X_d$$
$$X_d = D^1 X_c + D^1 X_d$$
$$X_e = D^2 X_b$$

TRANSFER FUNCTION

$$X_c = D^3 X_a + D^1 X_b$$
$$X_b = D^2 X_c + D^2 X_d$$
$$X_d = D^1 X_c + D^1 X_d$$
$$X_e = D^2 X_b$$

Transfer function $T(D) = \frac{X_e}{X_a} = \frac{D^7}{1 - D^1 - D^3}$

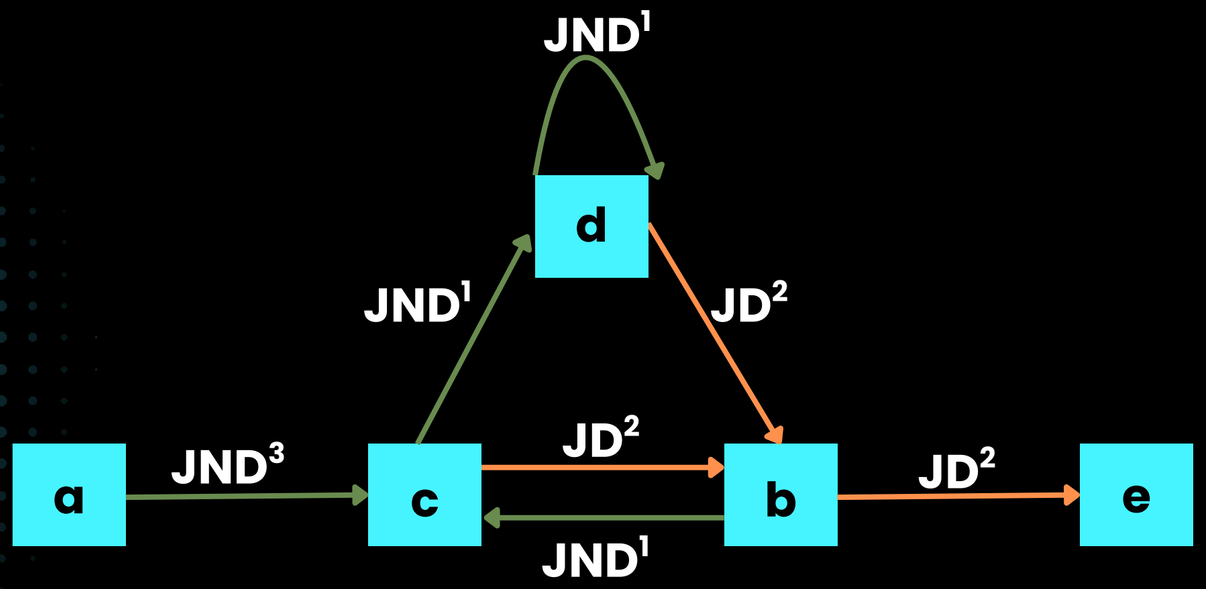
TRANSFER FUNCTION

Transfer function $T(D) = \frac{X_e}{X_a} = \frac{D^7}{1 - D^1 - D^3}$

$$T(D) = D^7 + D^8 + D^9 + 2D^{10} + \ldots$$

$$T(D) = \sum_{n=0}^{\infty} a_n D^n$$

TRANSFER FUNCTION



$$X_c = JND^3 X_a + JND^1 X_b$$
$$X_b = JD^2 X_c + JD^2 X_d$$
$$X_d = JND^1 X_c + JND^1 X_d$$
$$X_e = JD^2 X_b$$

$$T(D, N, J) = \sum_{d=dfree}^{\infty} a_d D^d N^{f(d)} J^{g(d)}$$

dfree = Minimum Hamming weight of non-zero paths from and to the zero state.
d = no. of ones in output codeword
f(d) = no. of ones in input (k-bits)
g(d) = no. of branches spanned by the path
a_d = Number of paths with Hamming weight d

BPSK MODULATION – BIT-TO-SIGNAL MAPPING

1.Source Bit Vector

- Output of the channel encoder is a bit vector:

$$x \in \{0, 1\}$$

- Each element is a binary value generated from the convolutional encoder.

2. Modulation Rule: Bit to Voltage Mapping

- These bits are passed to the BPSK modulator.
- Modulation is done using:

$$s = 1 - 2x$$

  - If x = 0, then s = +1 volt
  - If x = 1, then s = -1 volt
- This converts the binary sequence into a real-valued sequence suitable for transmission over a physical channel.

AWGN CHANNEL

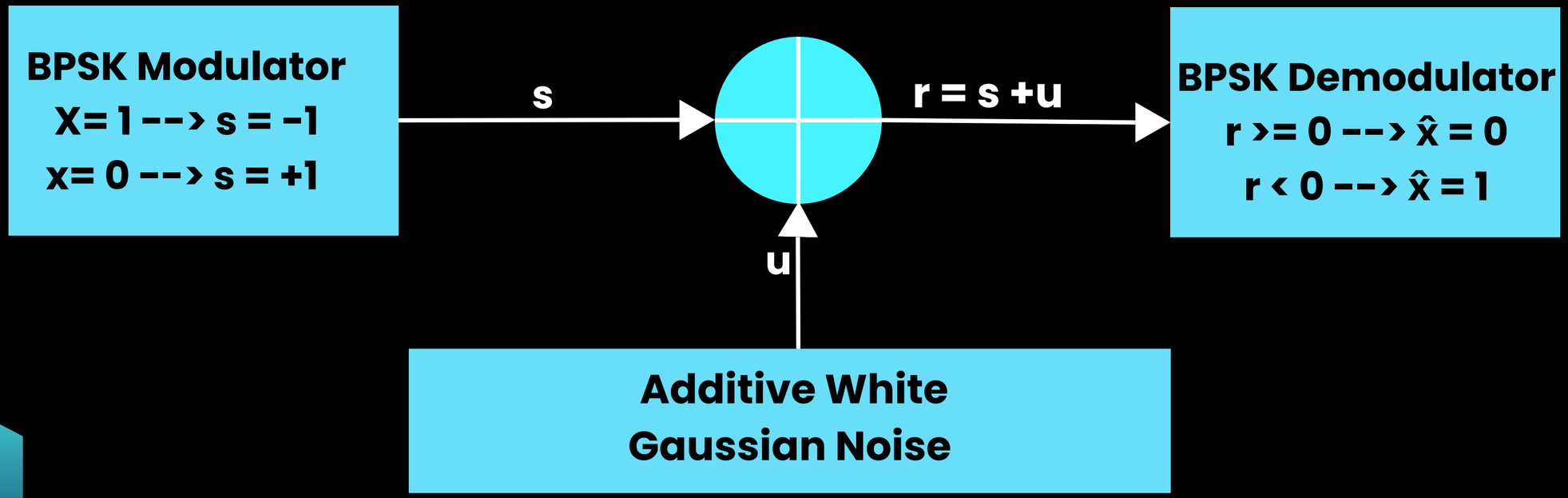
- Once BPSK modulation is done and the signal is ready for transmission, it passes through the AWGN (Additive White Gaussian Noise) channel.
- Signal-to-Noise Ratio (SNR)

  Each transmitted symbol experiences noise with an SNR of:

$$\gamma = \frac{E_s}{N_0}$$

- E_s: Signal energy per symbol (for BPSK, ES=1 Joule)
- N_o: Noise power spectral density
- now , we get the noise variance: $\sigma_n^2 = \frac{1}{\gamma}$
- This tells us how strong the noise is relative to the signal.

AWGN Channel



$$u = \sigma_n \times \mu_s$$

$$\mu_s \sim N(0,1)$$

VITERBI ALGORITHM

- It is a maximum likelihood decoding algorithm used for decoding convolutional codes.
- Works by finding the most probable path through a trellis diagram of states.

- Operates in three main steps:
- Trellis Construction – Build all possible state transitions.
- Path Metric Calculation – Calculate distance metric at each stage.
- Traceback – Follow the path with the lowest metric to recover the original data.

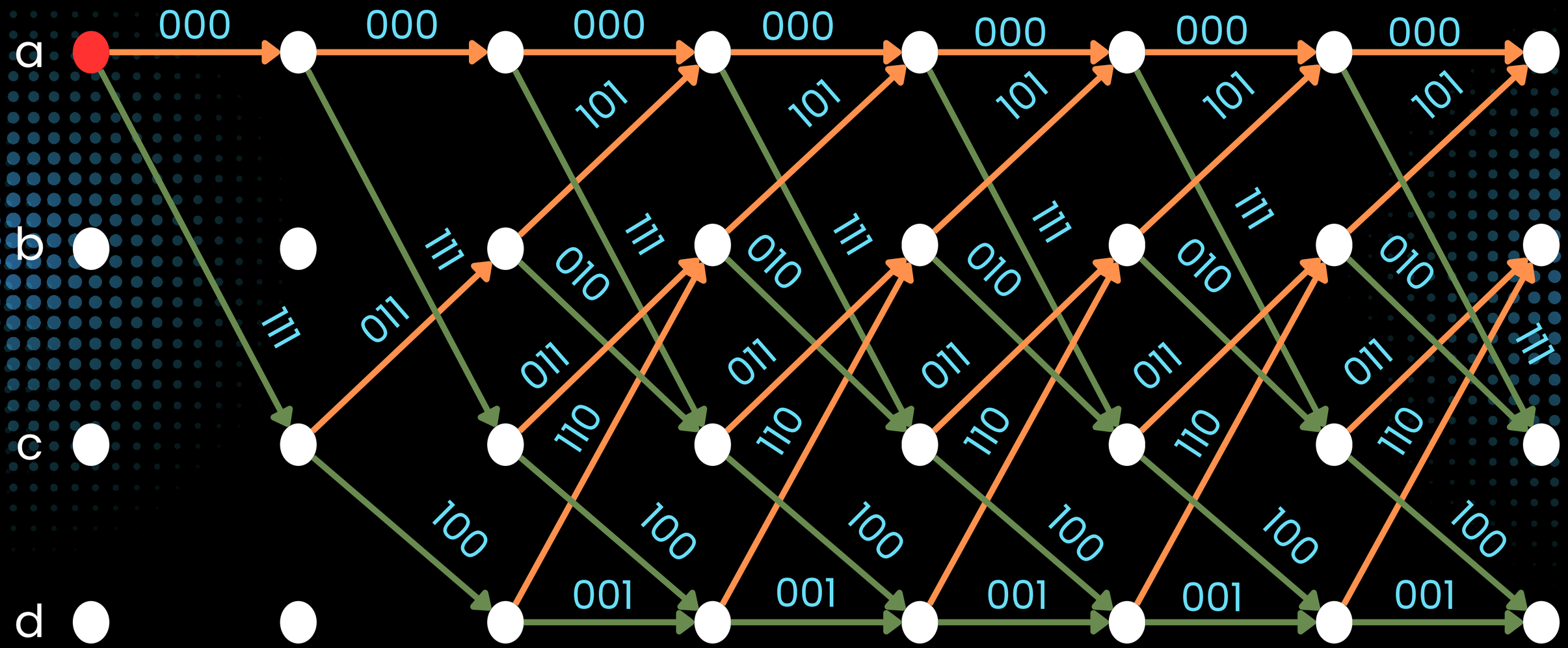
HARD DECISION DECODING

- In hard decoding, the receiver first demodulates the signal and converts it to bits (0 or 1) using a threshold decision .
- These hard-decided bits are then passed into the Viterbi decoder.
- **The Viterbi algorithm :**
  - Constructs a trellis of all possible state transitions.
  - Computes the Hamming distance between the received bits and expected bits for each transition.
  - Keeps track of the path with the minimum total Hamming distance (least bit errors).
  - At the end, uses traceback to recover the most likely original message.

HAMMING DISTANCE IN DECODING

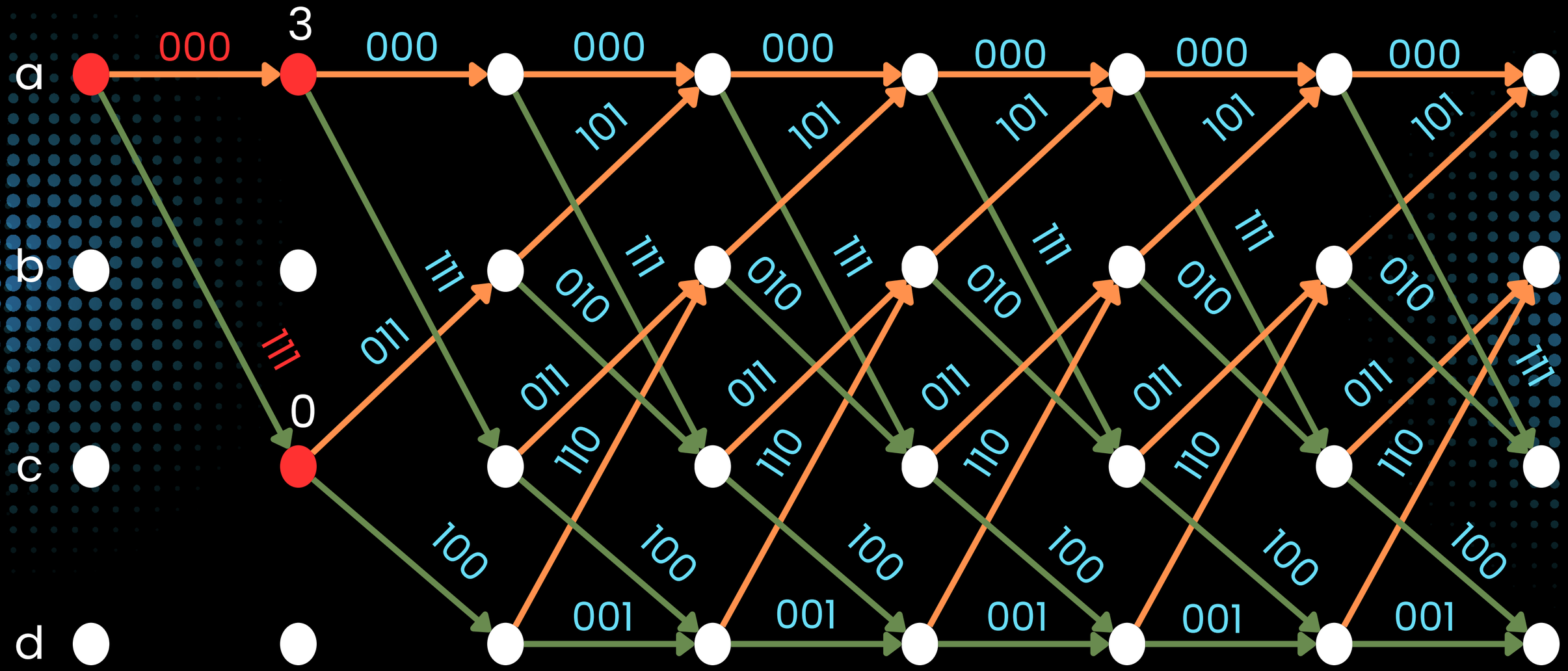
- **Hamming Distance in Hard Decoding :**
  - Input: Hard-decision bits (0 or 1) after BPSK demodulation
  - Compared With: Expected output bits from each possible encoder path
- **How It's Calculated:**
  - Count the number of bit positions where the received and expected bits differ

DECODING USING TRELLIS DIAGRAM



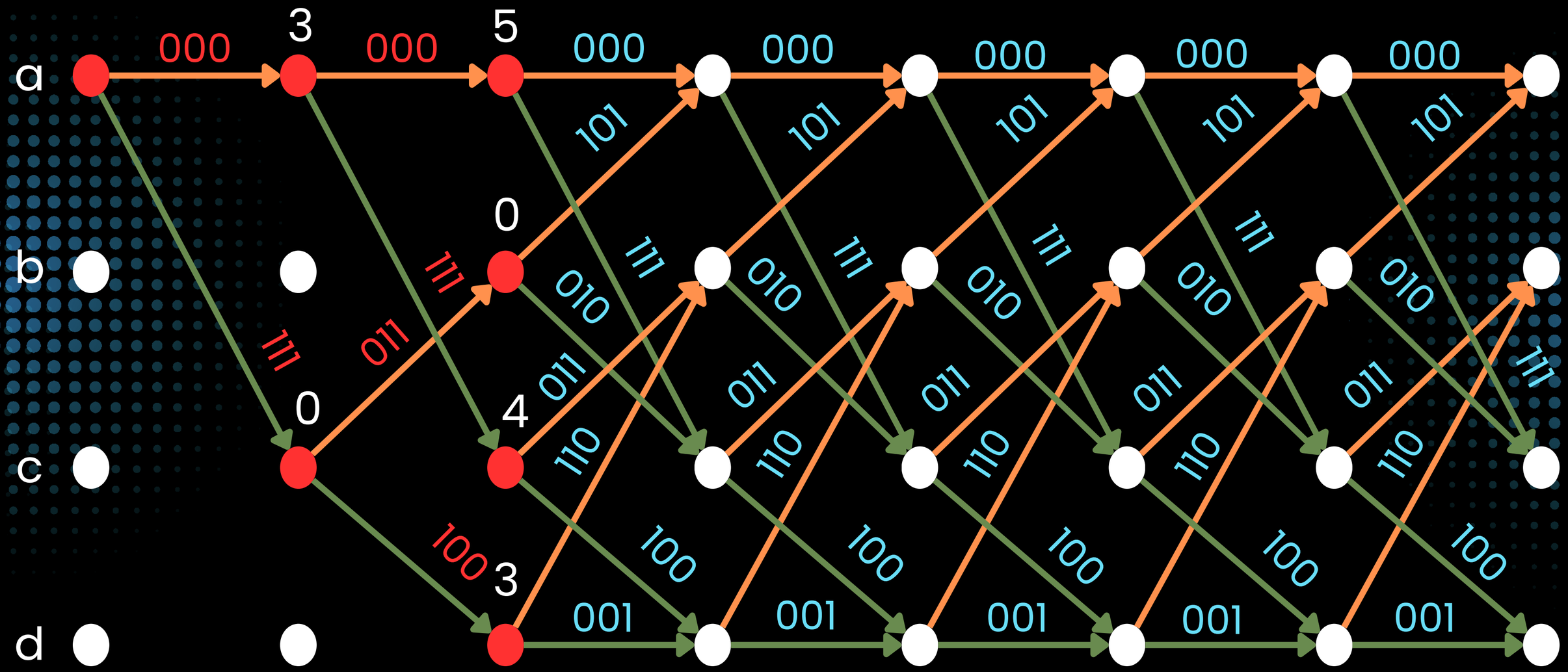
Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



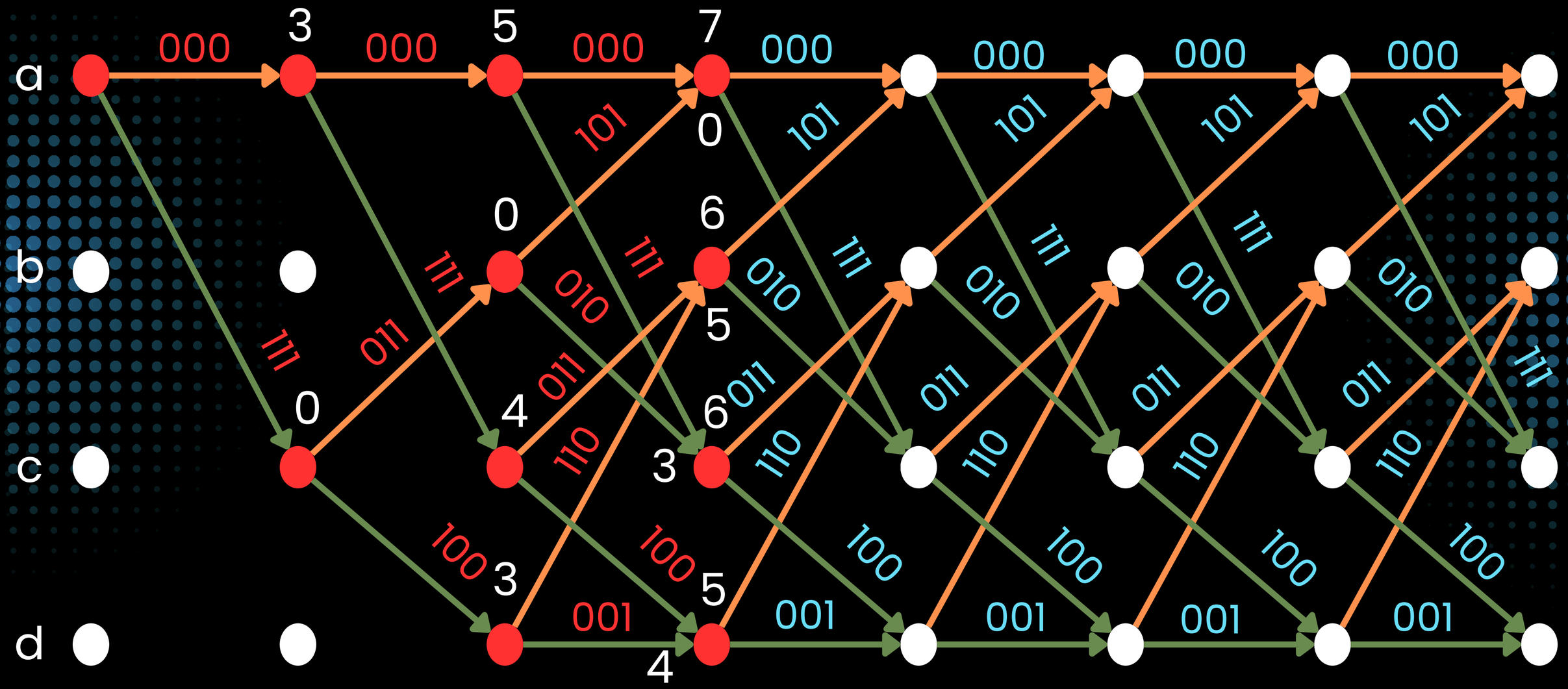
Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



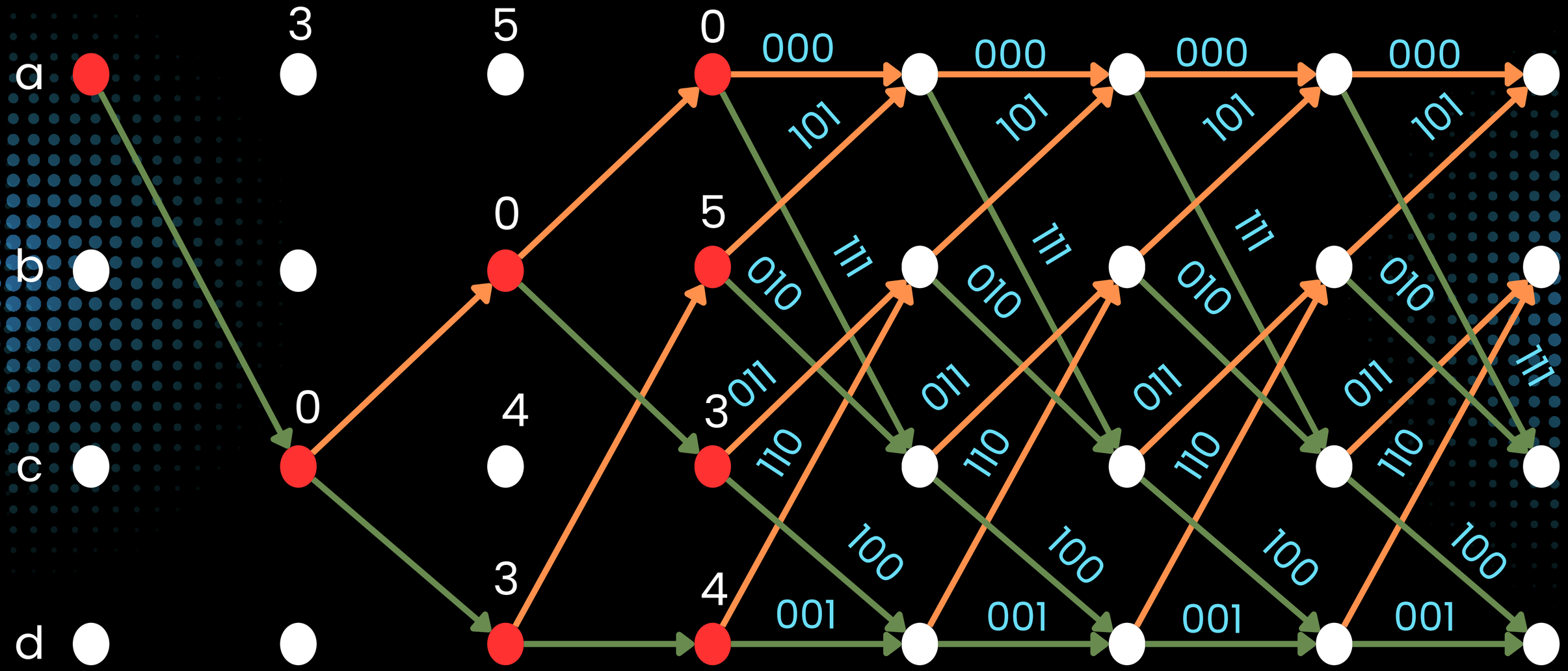
Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



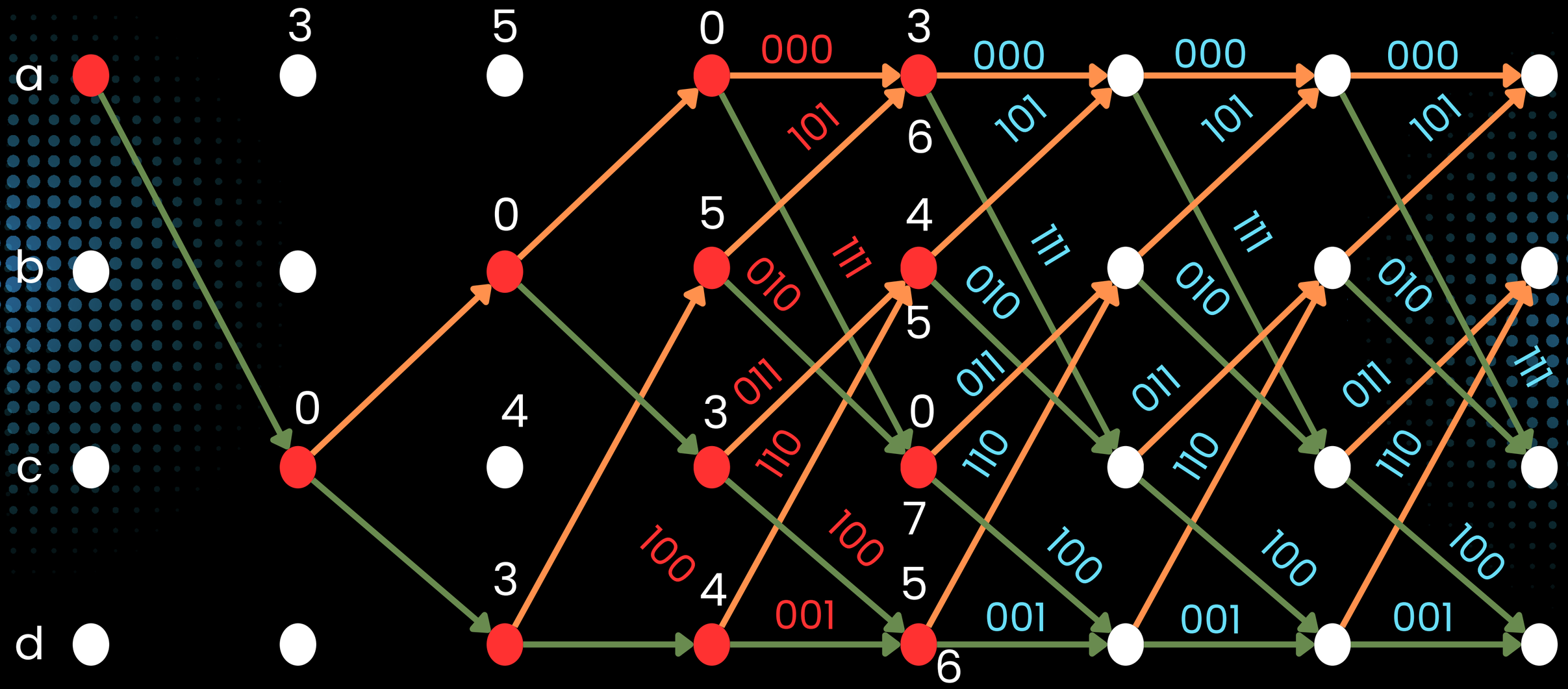
Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



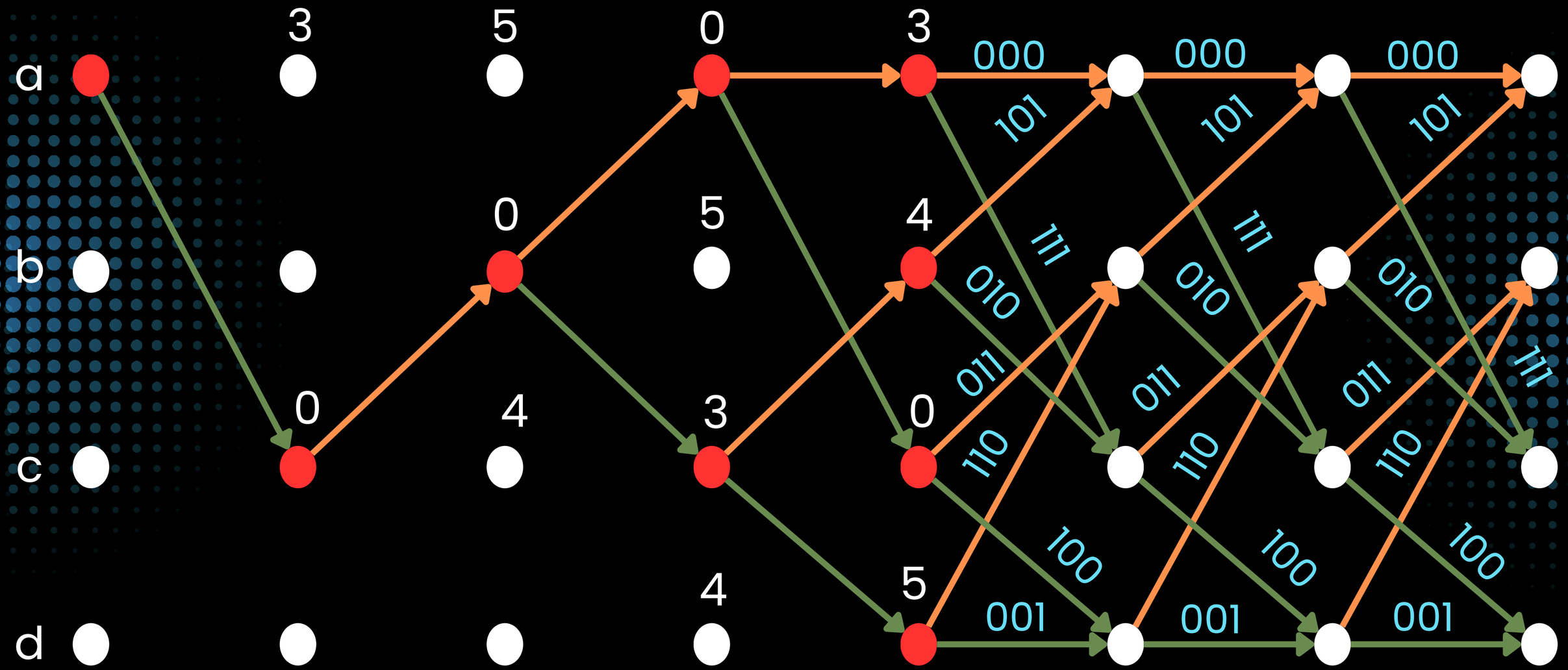
Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



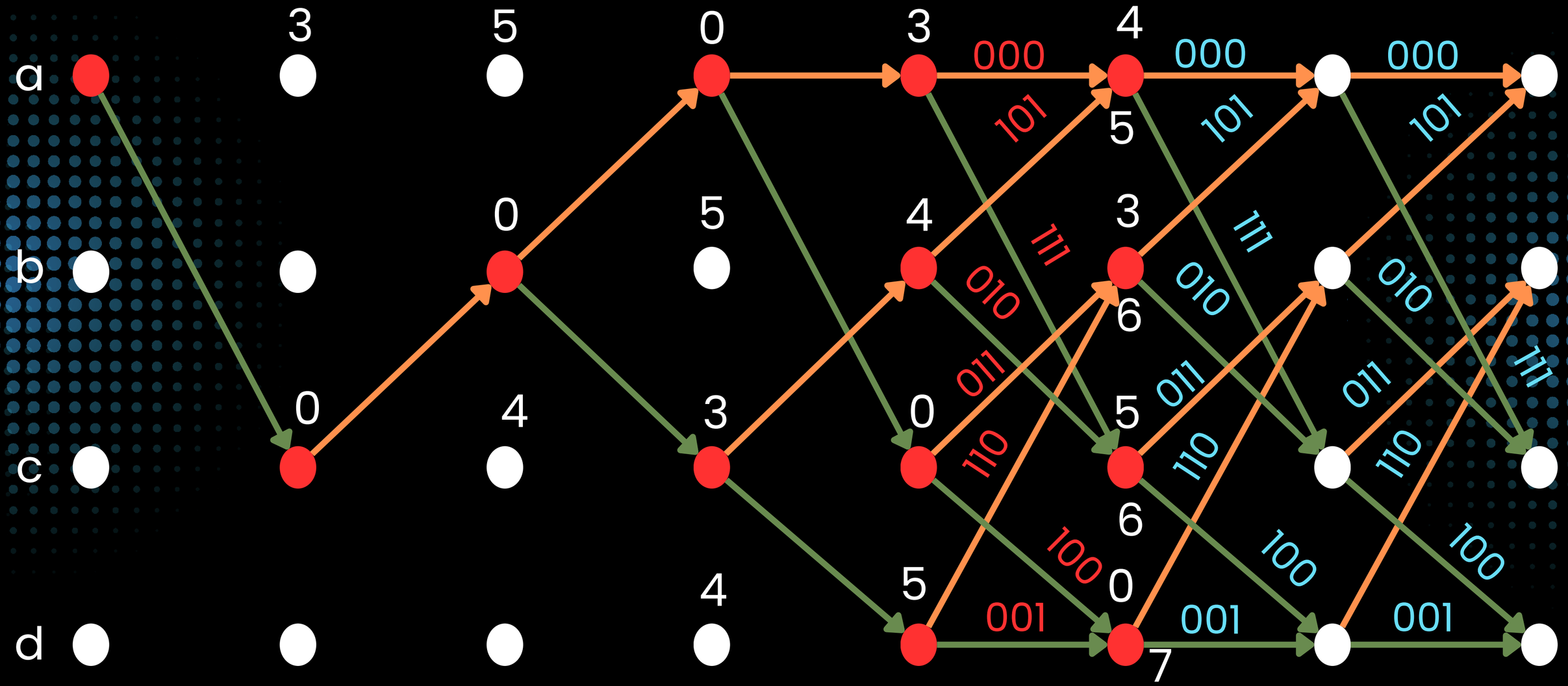
Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



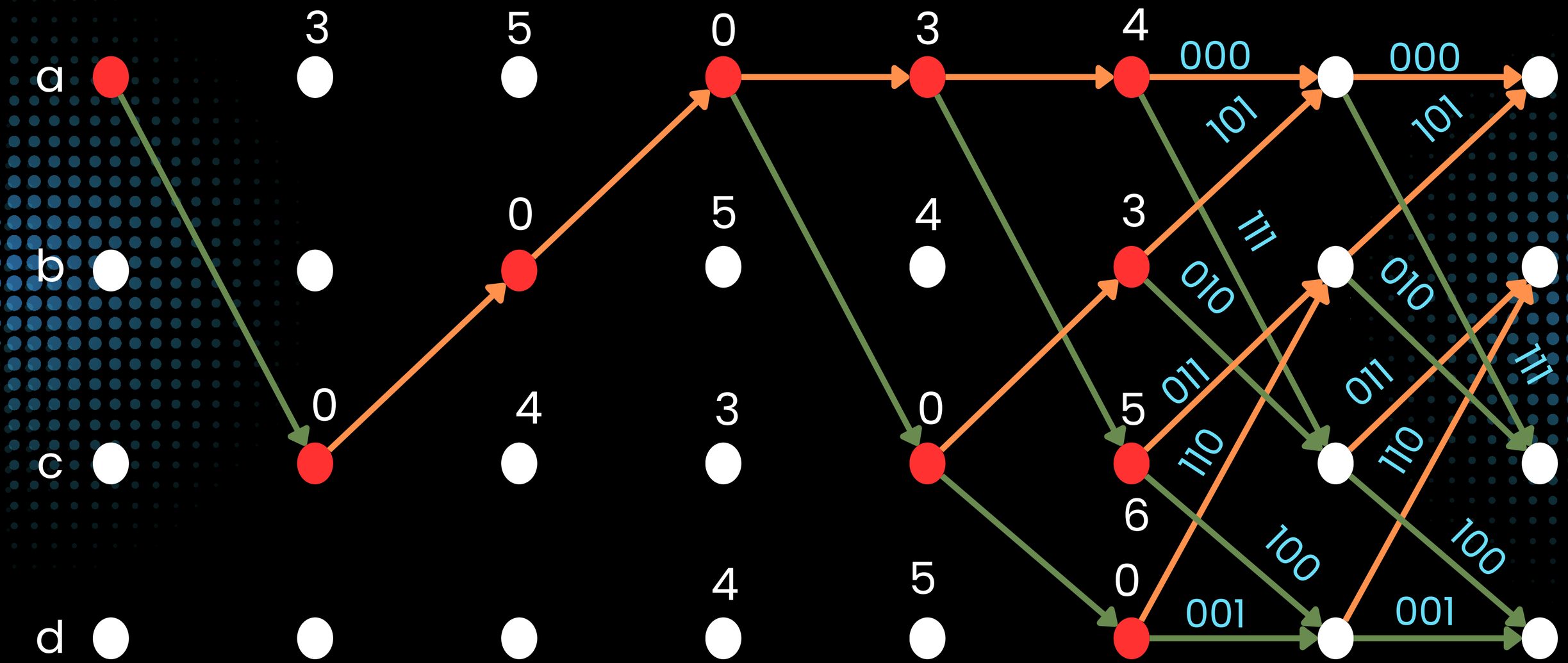
Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



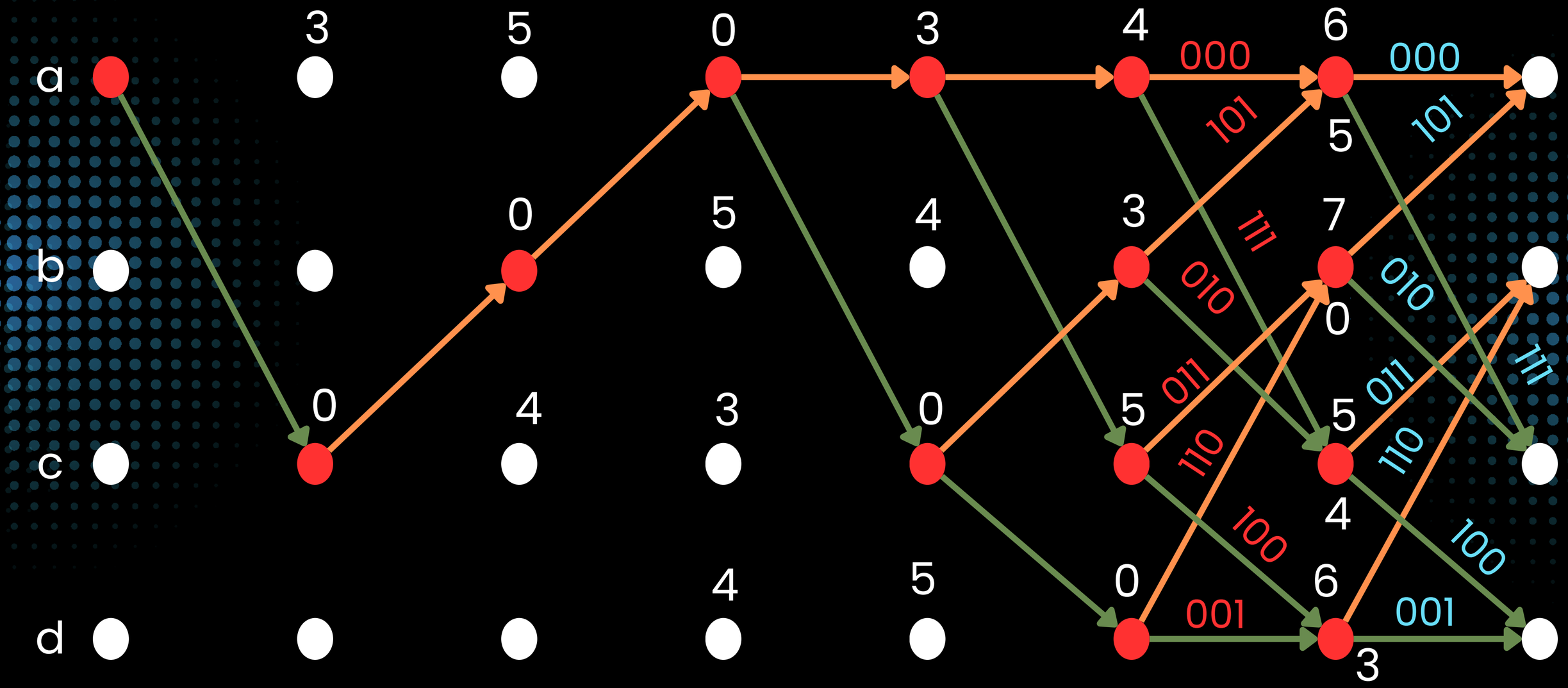
Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



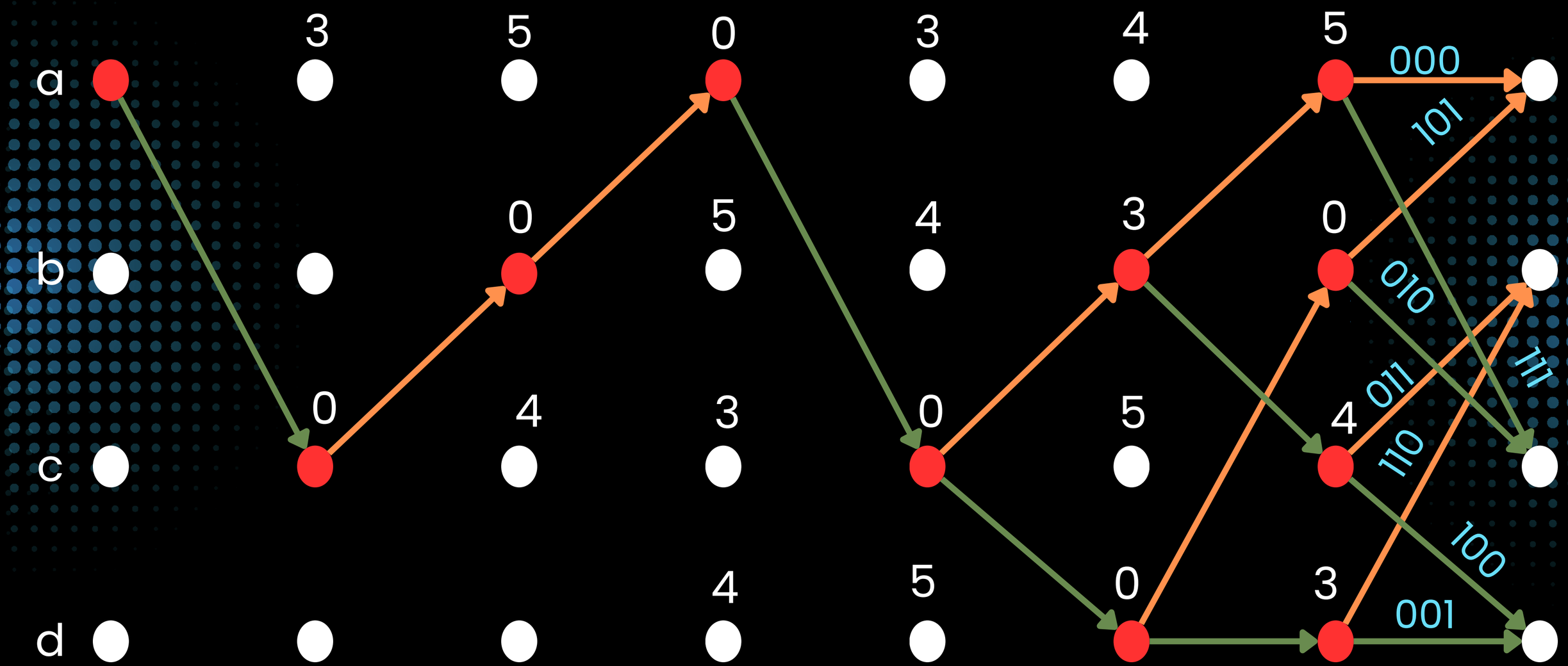
Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



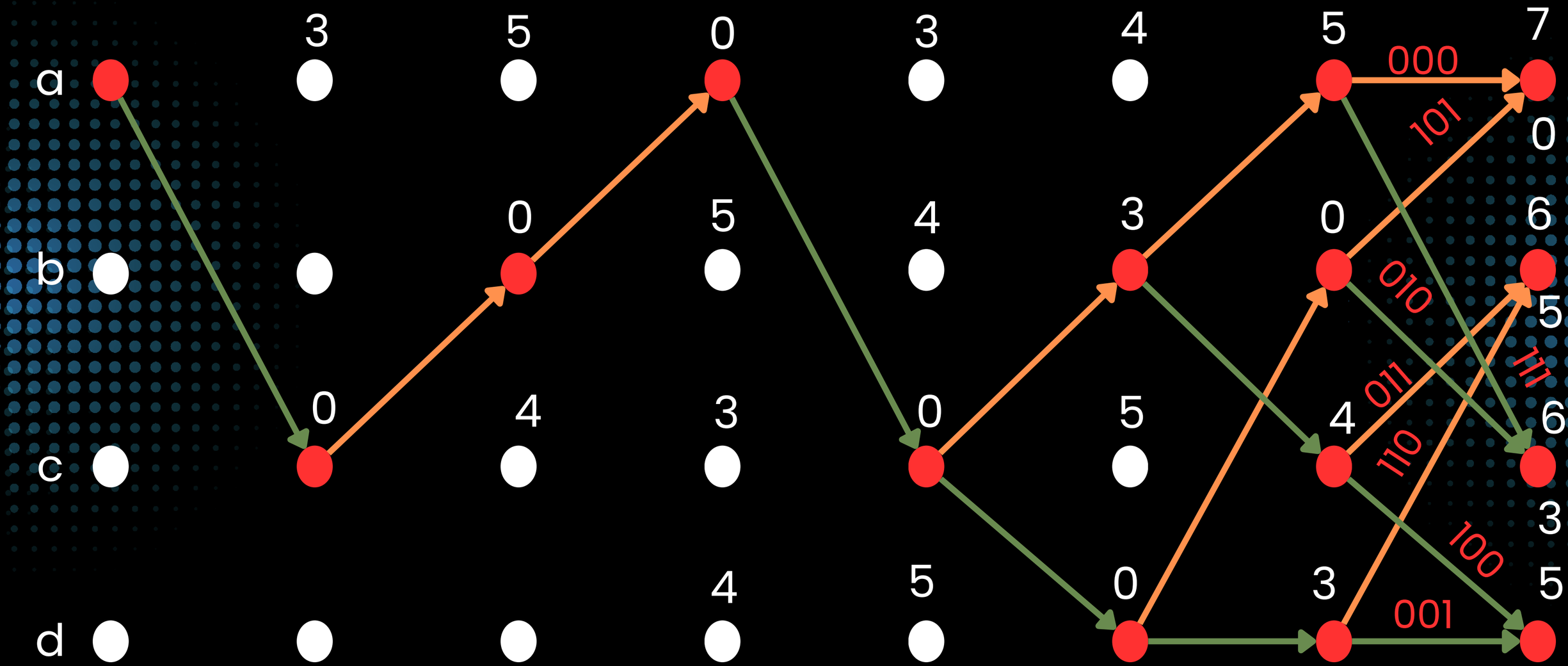
Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



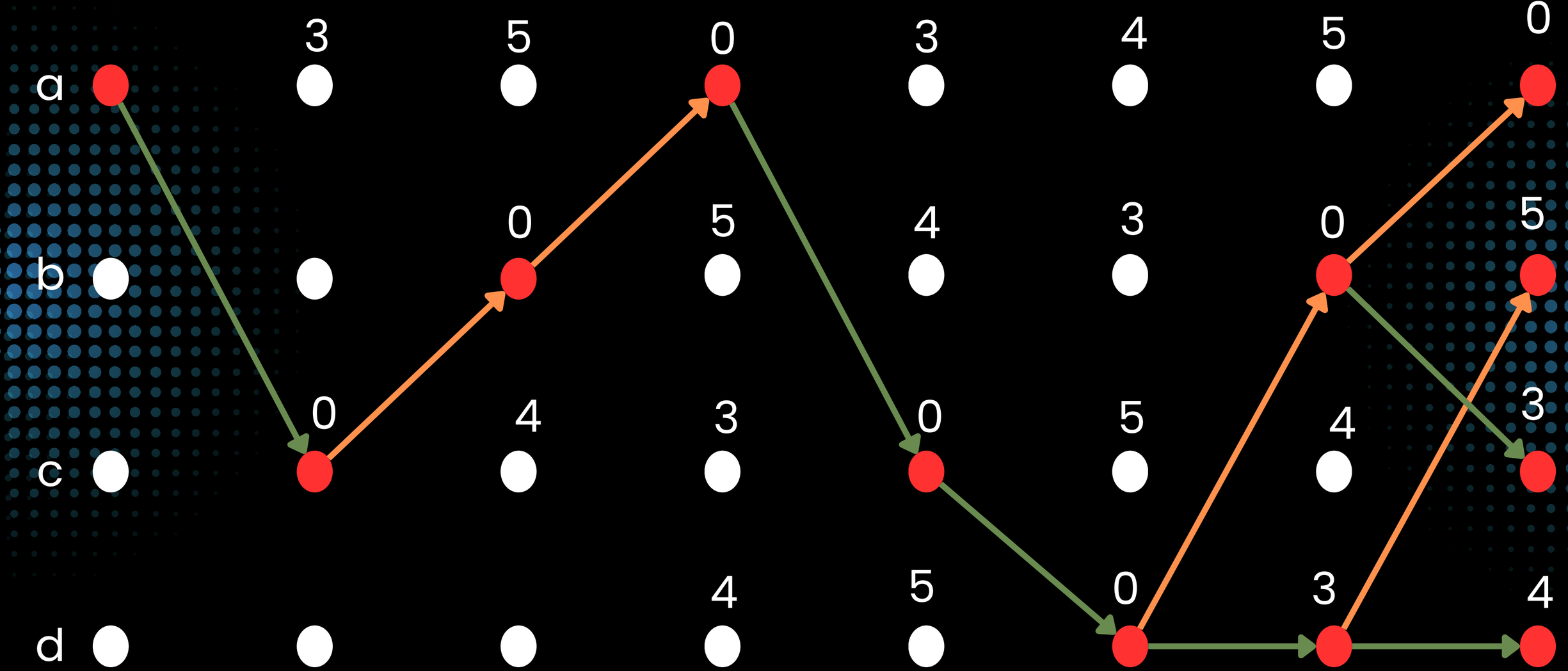
Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



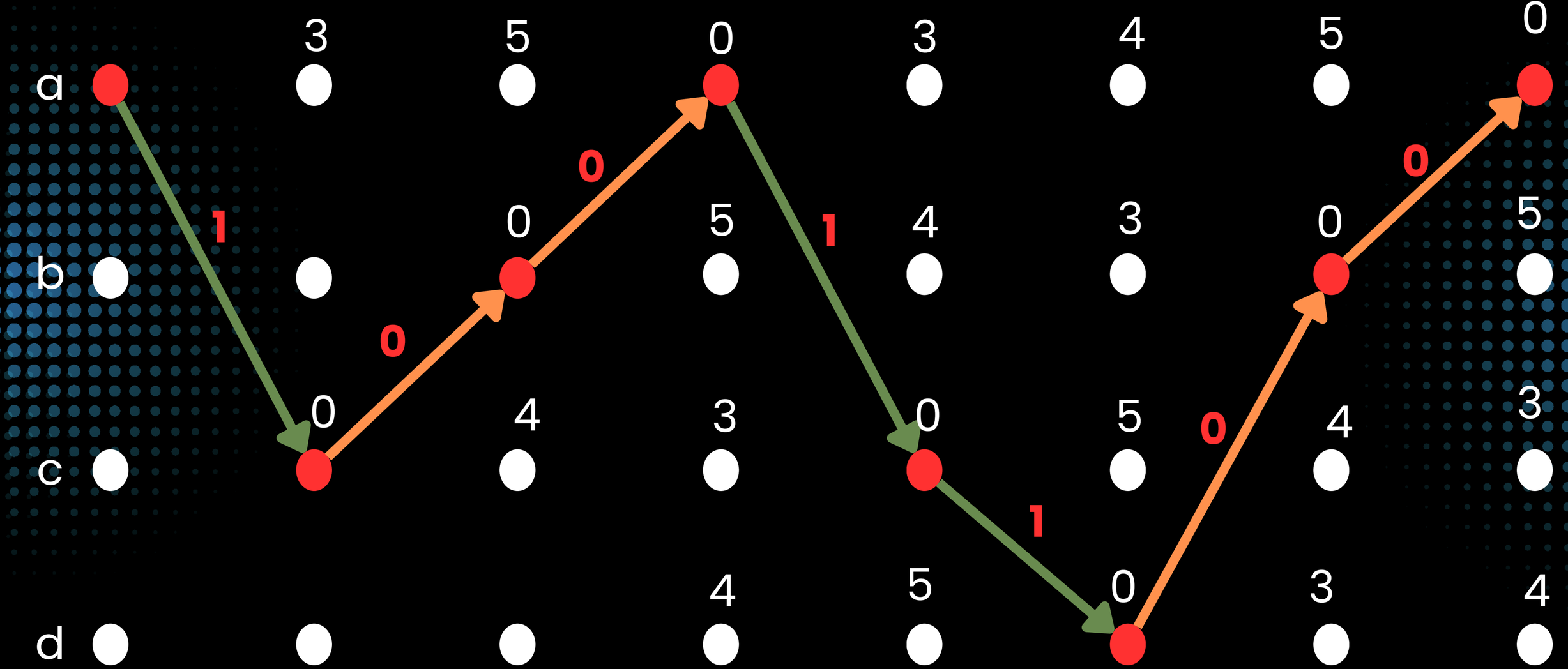
Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



Received Code word [111 011 101 111 100 110 101]

DECODING USING TRELLIS DIAGRAM



Decoded message [**1 0 0 1 1 0 0**]

SOFT DECISION DECODING

- In soft decoding, the real-valued signal received after AWGN(e.g., +0.9, -1.1) are directly fed to the Viterbi decoder.

- **The Viterbi algorithm:**
  - Builds a trellis of all possible state transitions.
  - Computes the Euclidean distance between received values and expected BPSK outputs.
  - Tracks the path with the minimum cumulative distance.
  - Uses traceback to find the most likely original message.

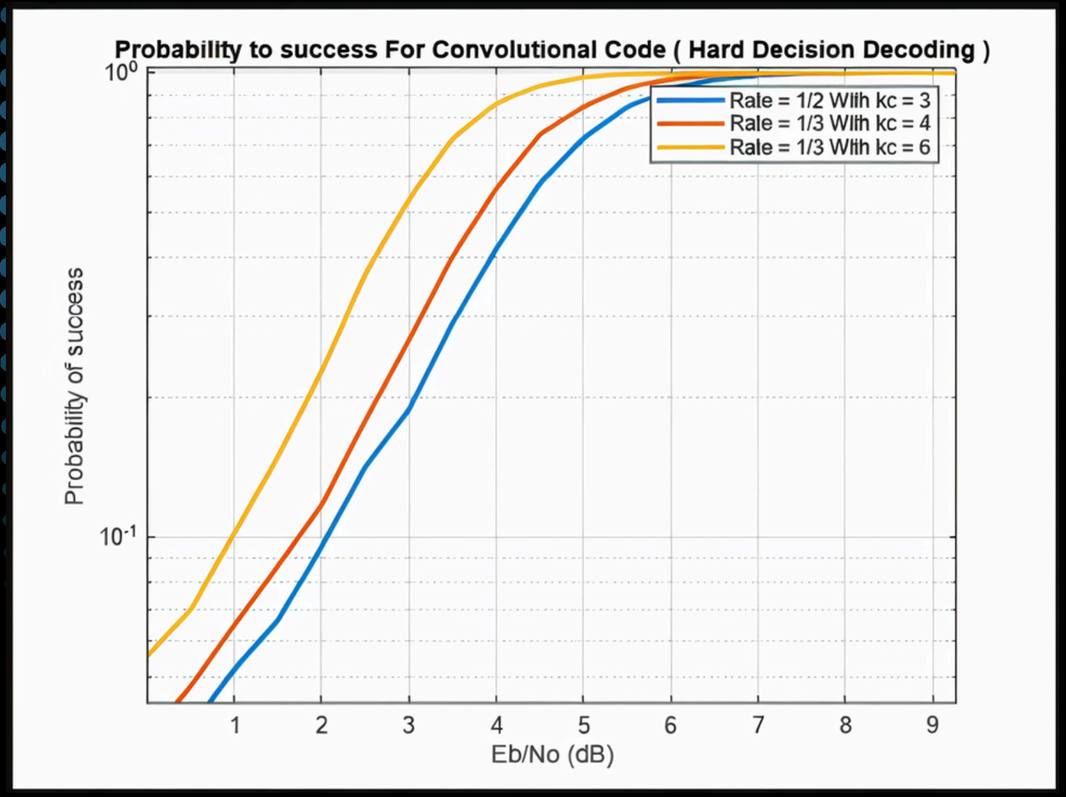
EUCLIDEAN DISTANCE IN DECODING

- **Euclidean Distance in Soft Decoding**
  - Input: Real-valued symbols received from the AWGN channel (e.g., values near +1 or -1 from BPSK)
  - Compared With: Expected modulated signals (e.g., +1 for 0, −1 for 1) from each possible encoder path
- **How It’s Calculated:**
  - Compute the sum of squared differences between the received and expected values:

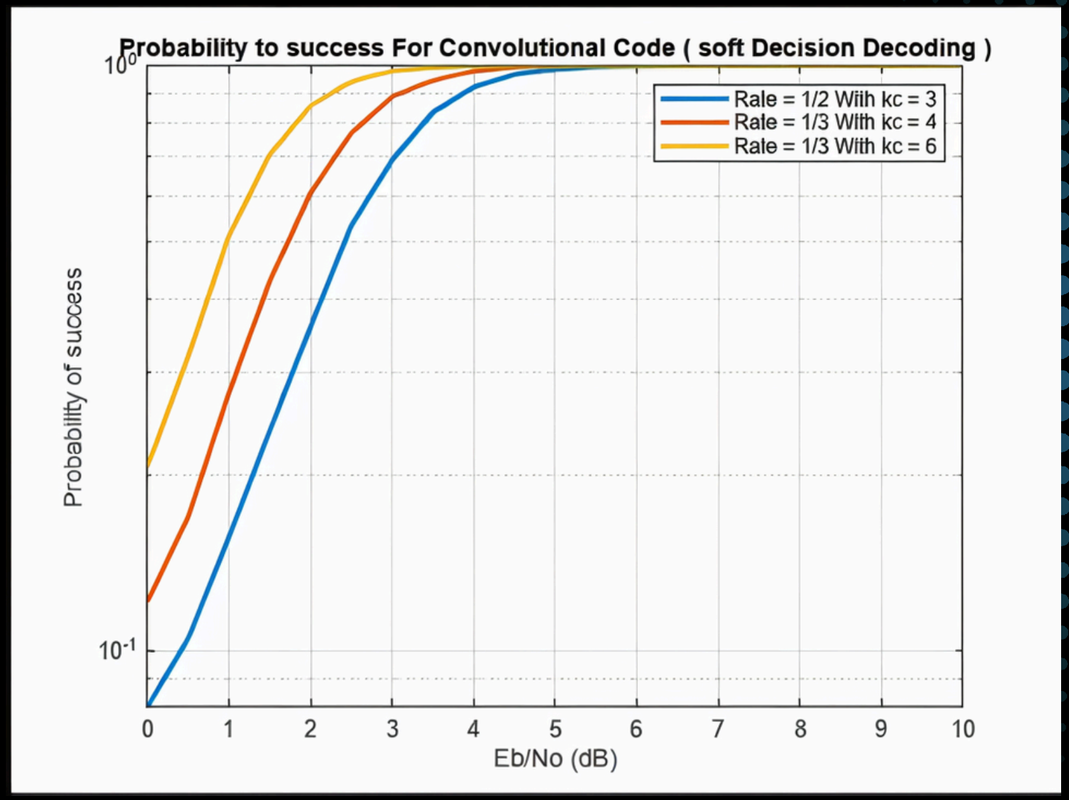
$$d = \Sigma(\text{received} - \text{expected})^2$$

ERROR PERFORMANCE COMPARISON

PROBABILITY OF SUCCESS (HDD)

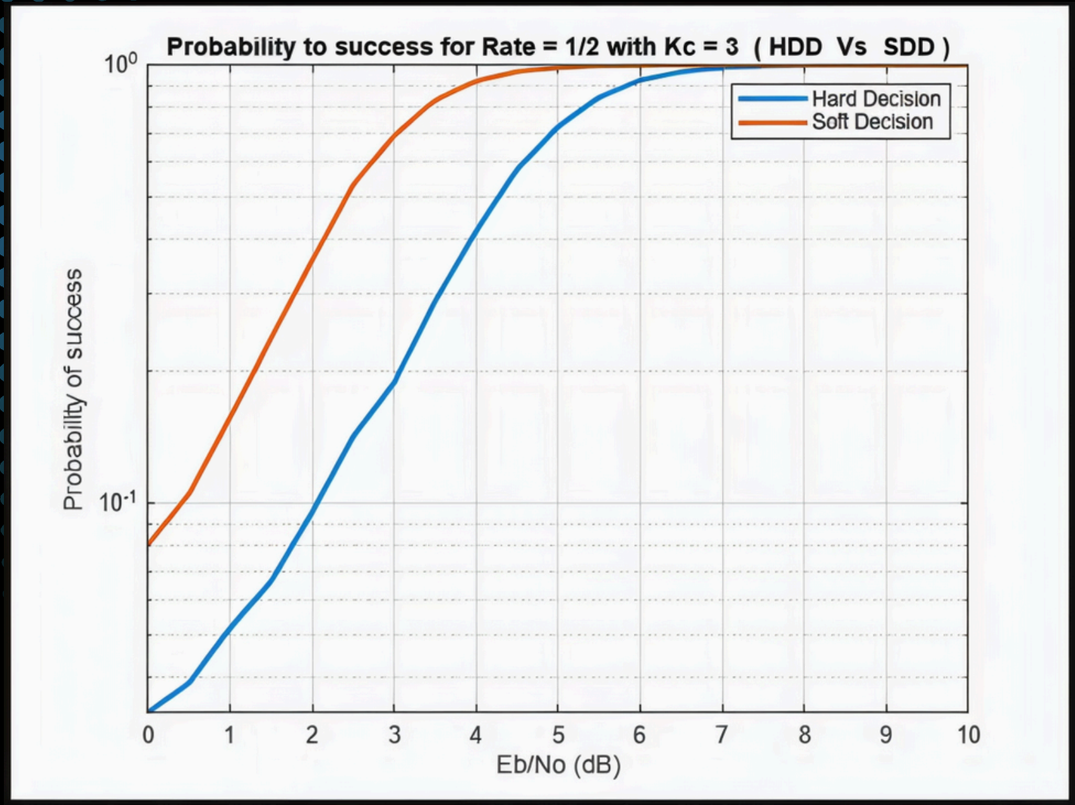


PROBABILITY OF SUCCESS (SDD)

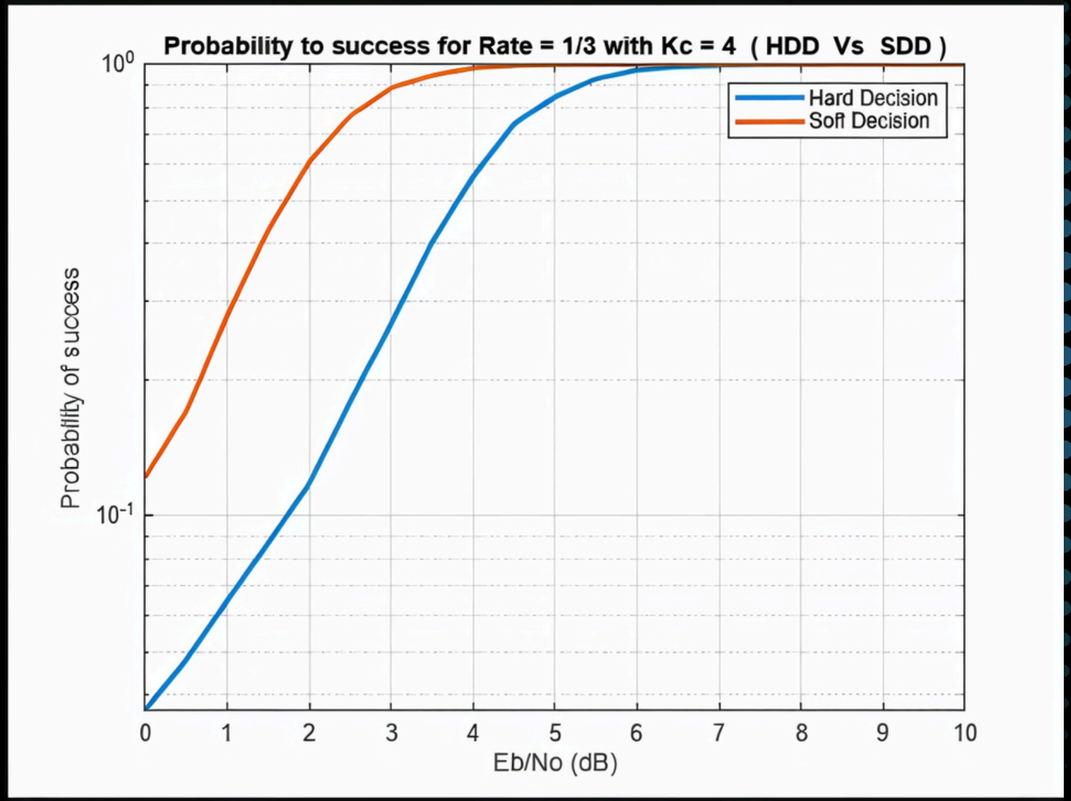


ERROR PERFORMANCE COMPARISON

PROBABILITY OF SUCCESS (HDD VS SDD) FOR KC=3 AND R=1/2

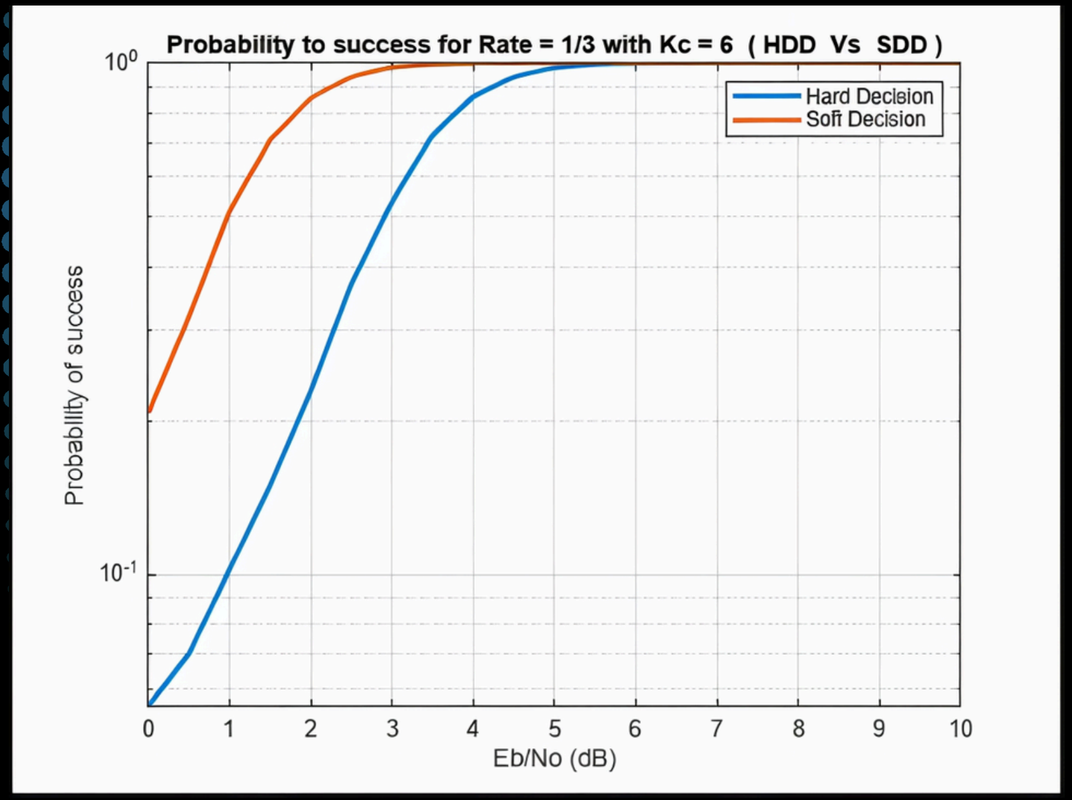


PROBABILITY OF SUCCESS (HDD VS SDD) FOR KC=4 AND R=1/3

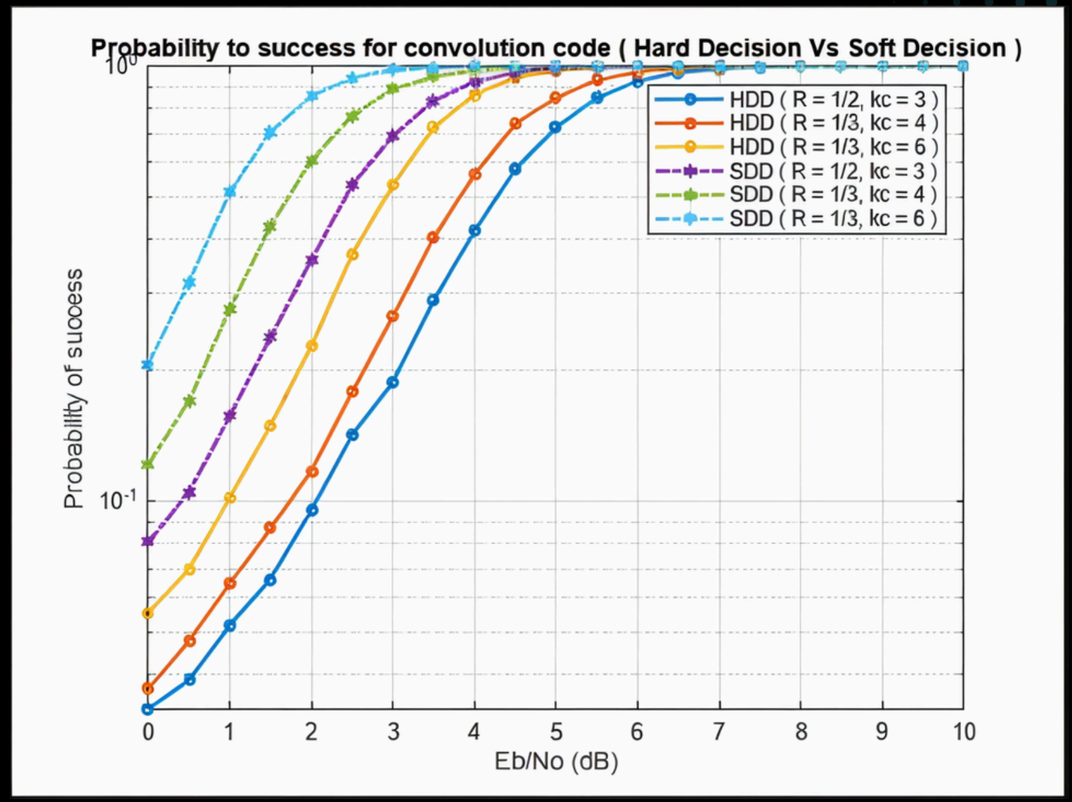


ERROR PERFORMANCE COMPARISON

PROBABILITY OF SUCCESS (HDD VS SDD) FOR KC=6 AND R=1/3

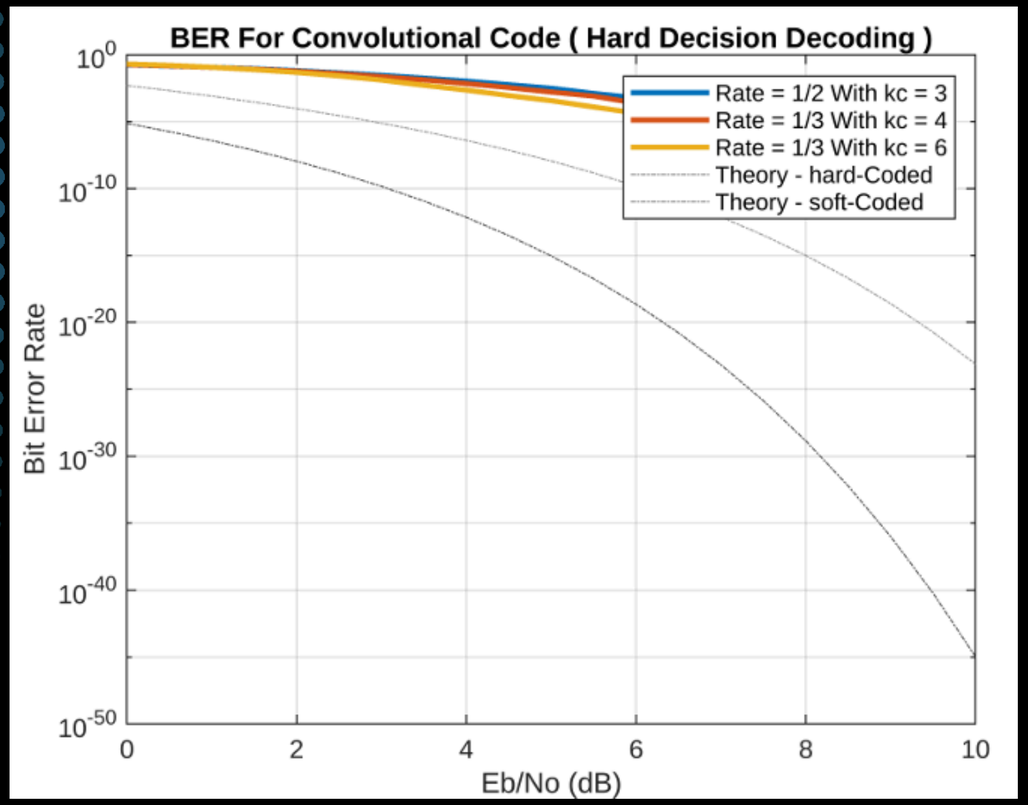


PROBABILITY OF SUCCESS (HDD VS SDD)

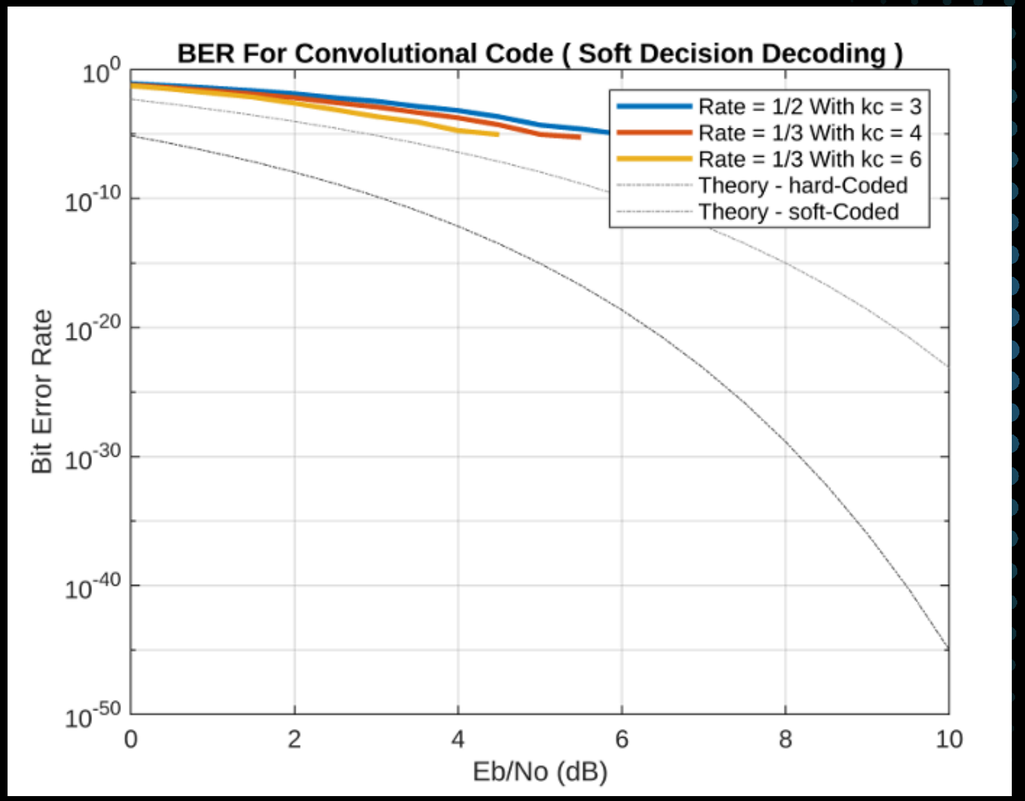


ERROR PERFORMANCE COMPARISON

BIT ERROR RATE (HDD)

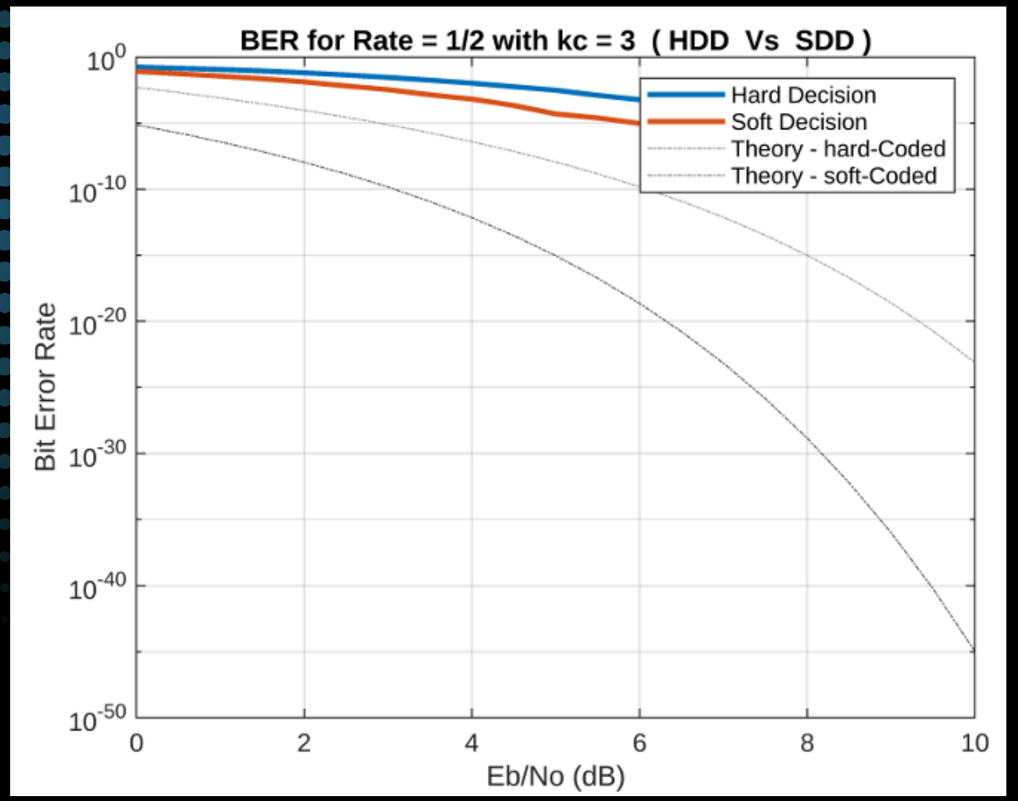


BIT ERROR RATE (SDD)

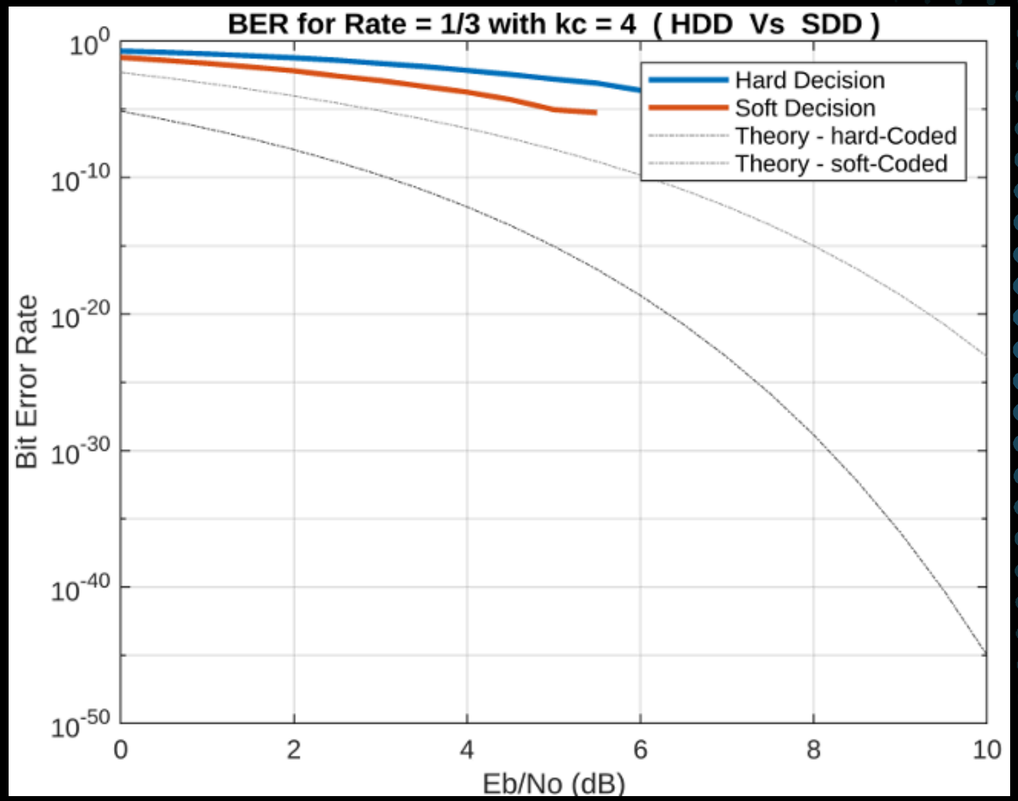


ERROR PERFORMANCE COMPARISON

BIT ERROR RATE (HDD VS SDD) FOR KC=3 AND R=1/2

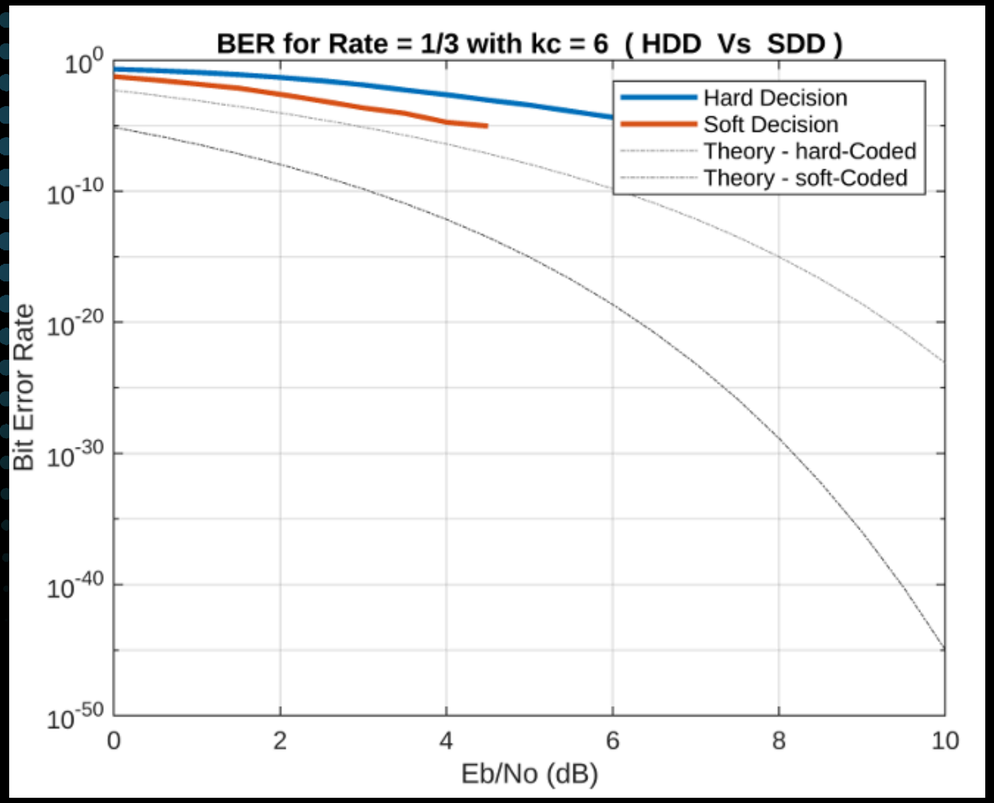


BIT ERROR RATE (HDD VS SDD) FOR KC=4 AND R=1/3

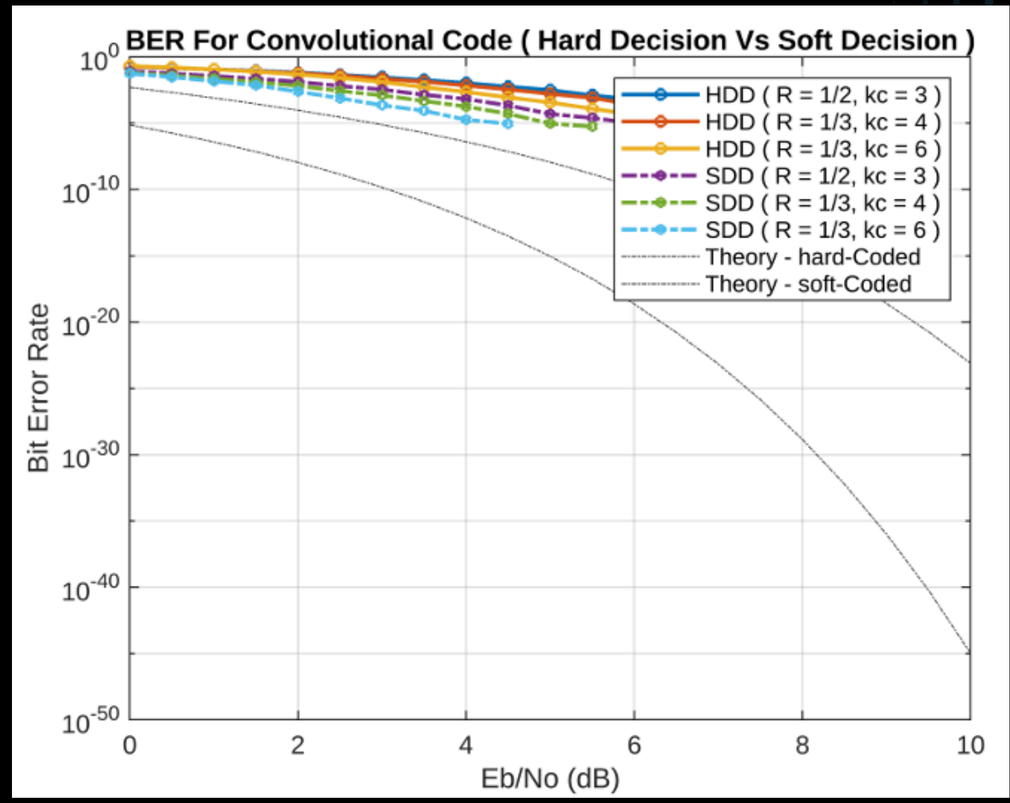


ERROR PERFORMANCE COMPARISON

BIT ERROR RATE (HDD VS SDD) FOR KC=6 AND R=1/3



BIT ERROR RATE (HDD VS SDD)



THANK YOU