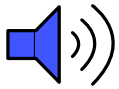


Channel Coding for IEEE 802.16e Mobile WiMAX

Matthew C. Valenti

Lane Department of Computer Science and Electrical Engineering
West Virginia University
U.S.A.

June 2009

Outline

Outline

IEEE 802.16

IEEE 802.16 is a family of standards for Wireless MAN's.

- Metropolitan area networks.
- Wireless at broadband speeds.

Applications of IEEE 802.16

- Wireless backhaul.
- Residential broadband.
- Cellular-like service.

Progression of standards

- 802.16
  - → December 2001.
  - → 10-66 GHz.
  - → Line-of-sight (LOS) only.
  - → Up to 134.4 Mbps operation using 28 MHz bandwidth.
- 802.16-2004
  - → June 2004.
  - → Added 2-11 GHz non-LOS operation.
  - → Up to 75 Mbps operation using 15 MHz bandwidth.
- 802.16e-2005
  - → December 2005.
  - → Added support for mobility.

Key Technologies

Advanced technologies supported by IEEE 802.16

- OFDM and OFDMA.
- Adaptive modulation: QPSK, 16-QAM, or 64-QAM.
- Adaptive turbo and LDPC codes.
- Hybrid-ARQ
- MIMO: Space-time codes and spatial multiplexing.
- Time-division duplexing.
- Multiuser diversity.
- Partial frequency reuse.

WiMAX Forum

The WiMAX forum is an consortium of over 500 companies whose purpose is to commercialize systems based on IEEE 802.16 technology.

The activities of the WiMAX forum include:

- Development of WiMAX system profiles.
- Certification of equipment.
- Standardization of higher-layer functionality.,

WiMAX vs. mobile WiMAX

WiMAX

- fixed system profile.
- OFDM PHY from IEEE 802.16-2004.
- 256 OFDM subcarriers (fixed).
- 3.5 MHz bandwidth.

mobile WiMAX

- mobility system profile.
- OFDMA PHY from IEEE 802.16e-2004.
- 128 to 2,048 subcarriers (scalable).
- 1.25 to 20 MHz bandwidths

Channel Codes Specified in IEEE 802.16e

Four codes are specified in IEEE 802.16e.

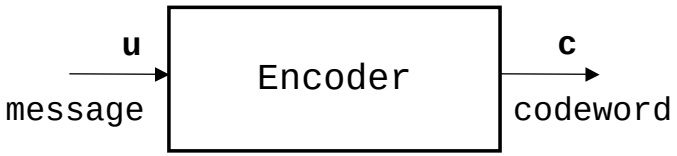
1. Tailbiting convolutional code.
2. Block turbo code (BTC).
3. Convolutional turbo code (CTC).
4. Low-density Parity-check (LDPC) code.

The goal of the remainder of this tutorial is to describe each of these codes in detail.

Outline

Code and Message Vectors

- A *binary code* $\mathcal{C}$ is a set of 2^k codewords $\mathbf{c}_i, 0 \leq i < 2^k$.
- Each *codeword* is represented by a length n binary vector.
- Each codeword is associated with a unique *message* $\mathbf{u}_i, 0 \leq i < 2^k$, which is a length k binary vector.
- The code must define the mapping from messages to codewords $\mathbf{u} \longrightarrow \mathbf{c}$.

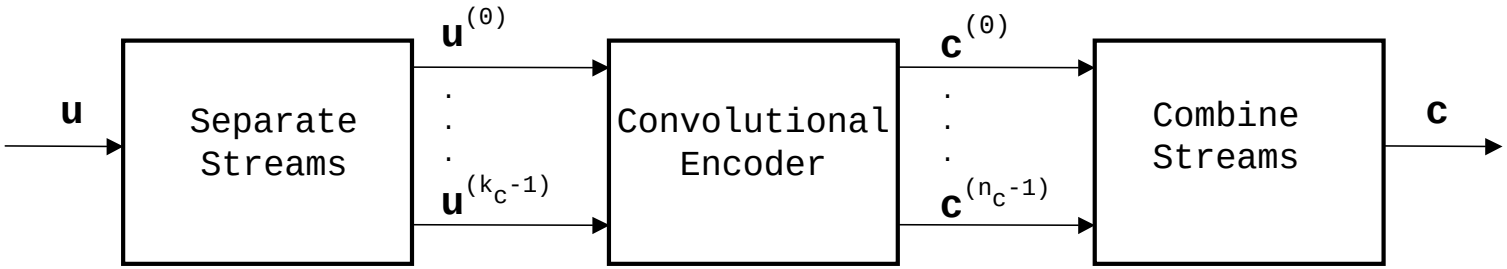


Linear codes.

- A code is *linear* if the modulo-2 sum of any two codewords is also a codeword.
  - Mathematically, if $\mathbf{c}_i \in \mathcal{C}$ and $\mathbf{c}_j \in \mathcal{C}$, then $\mathbf{c}_i + \mathbf{c}_j \in \mathcal{C}$.
  - Note that the addition is modulo-2.
- Because $\mathbf{c}_i + \mathbf{c}_i = \mathbf{0}$, it follows that all linear codes must contain the all-zeros codeword.
- All codes considered in this tutorial are linear.

Encoding

- A *convolutional encoder* is a device with k_c inputs and n_c outputs, where $n_c \geq k_c$.
- The input message $\mathbf{u}$ is split into k_c input *streams* $\mathbf{u}^{(i)}, 0 \leq i \leq k_c - 1$ each of length k/k_c.
- Similarly, the output codeword $\mathbf{c}$ is assembled from n_c output streams $\mathbf{c}^{(j)}, 0 \leq j \leq n_c - 1$ each of length n/n_c.
- In this tutorial, $1 \leq k_c \leq 2$ and $1 \leq n_c \leq 4$.



Convolutional Encoding when $k_c = 1$

- Suppose there is one input stream, $\mathbf{u}^{(0)} = \mathbf{u}$.
- Output stream $\mathbf{c}^{(j)}$ is found by convolving the input stream with a *generator sequence* $\mathbf{g}^{(j)}$ as follows:

$$\mathbf{c}^{(j)} = \mathbf{u} * \mathbf{g}^{(j)}$$

where the ℓ^{th} element of the output vector is

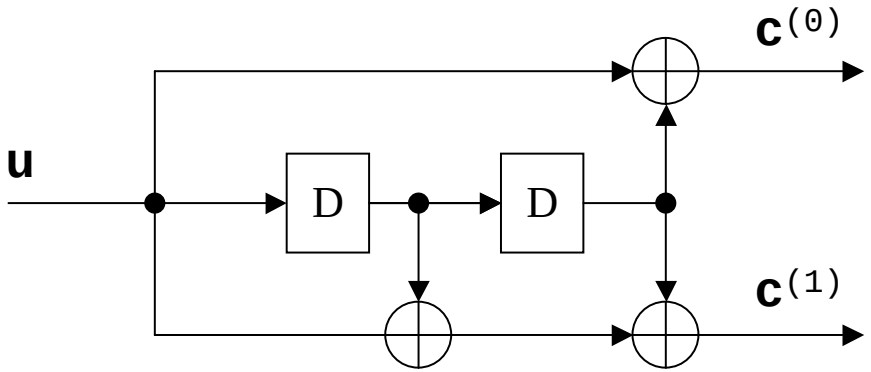
$$c_\ell^{(j)} = \sum_{k=0}^{m} u_{\ell-k} g_k^{(j)}$$

and the addition is modulo-2.

- m is the *memory* of the encoder.
- $\nu = m + 1$ is the *constraint length*.

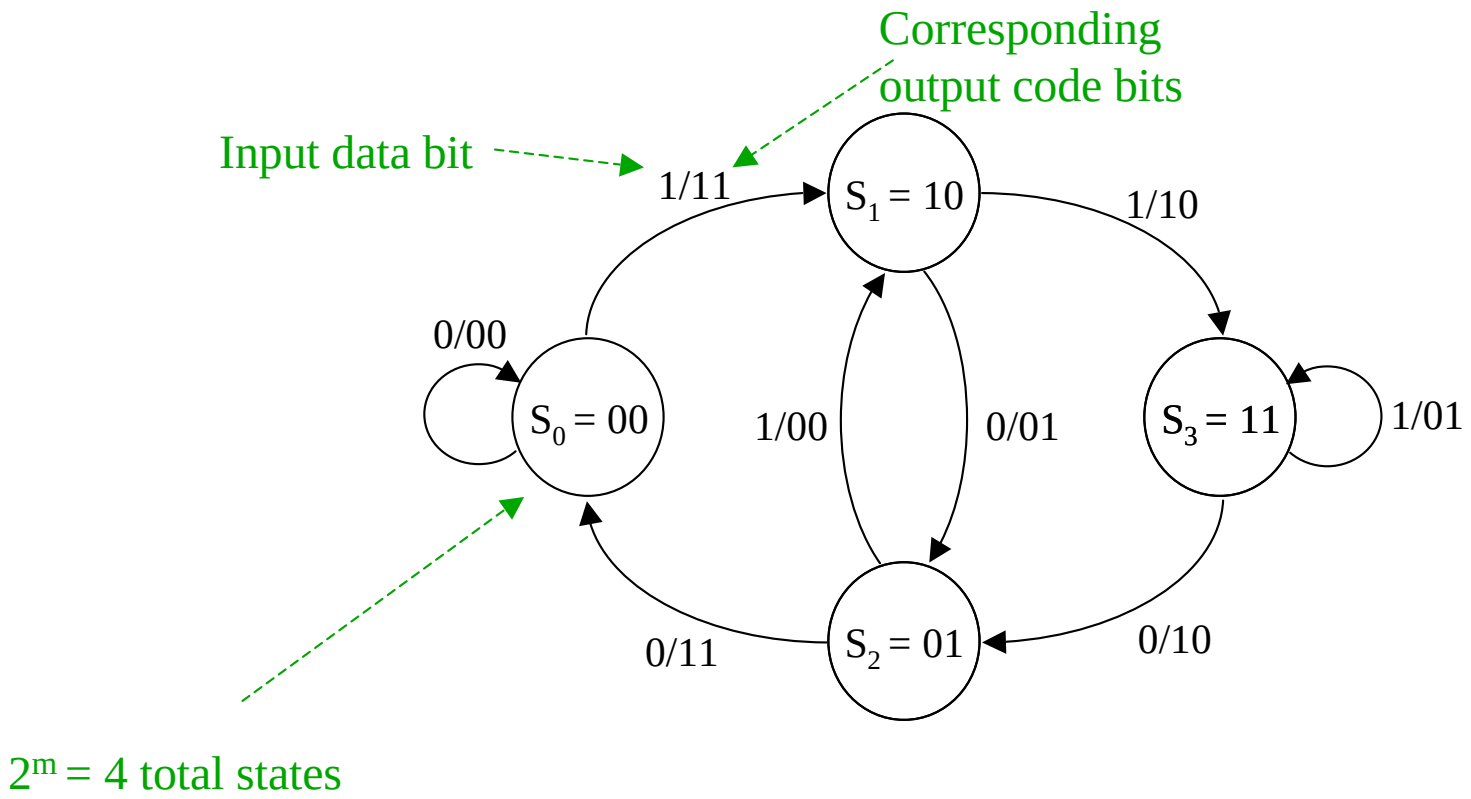
Encoder Diagram

- Let $\mathbf{g}^{(0)} = [101]$ and $\mathbf{g}^{(1)} = [111]$.
- The encoder may be realized with the following structure:



State diagram representation

A convolutional encoder is a finite state machine, and can be represented in terms of a state diagram.

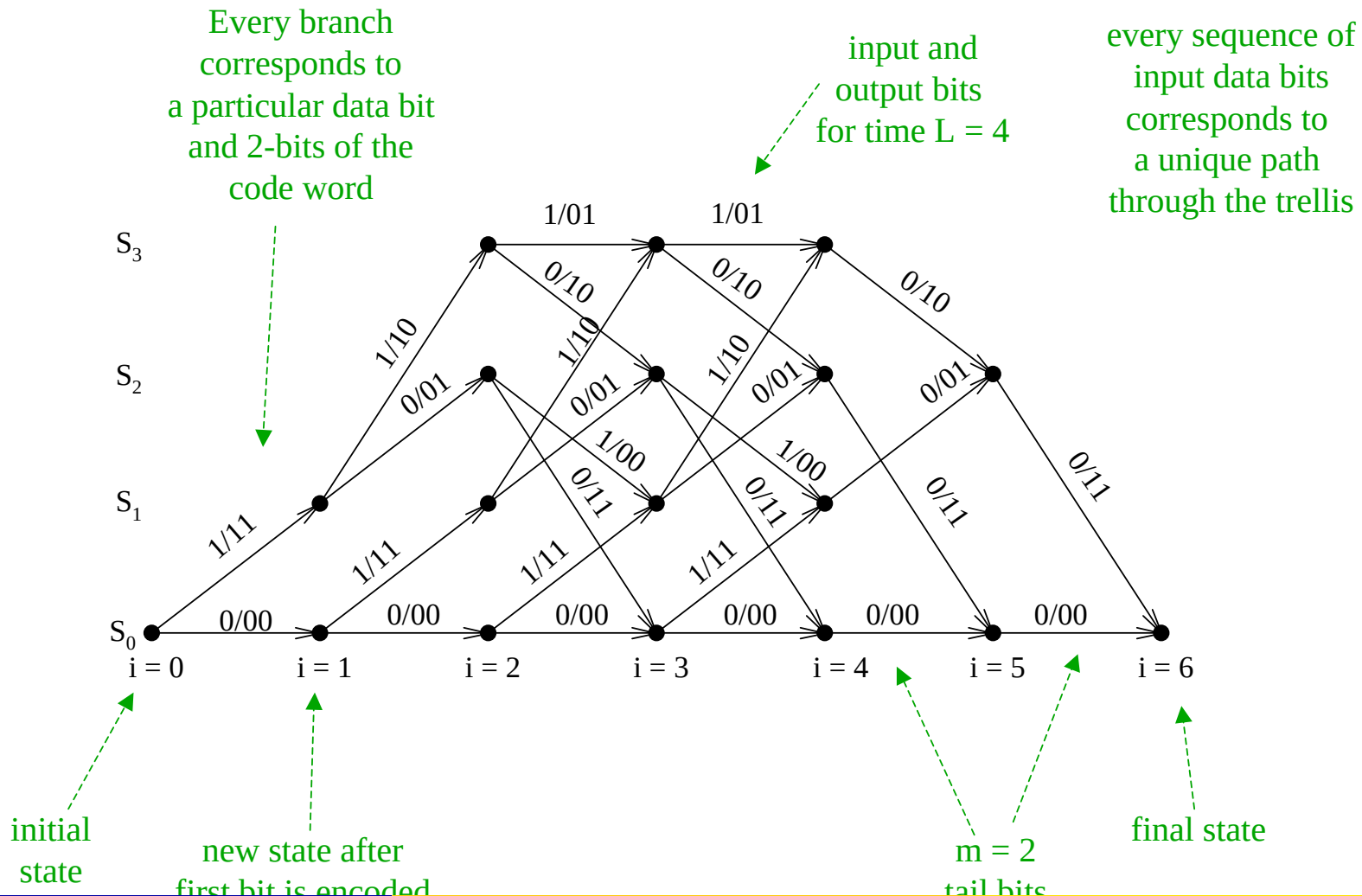


Initial and Terminating States

- The system needs a policy for choosing the initial and terminating states of the encoder.
- The usual convention is to start in the all-zeros state and then force the encoder to terminate in the all-zeros state.
- Termination in the all-zeros state requires a *tail* of m zeros.
- The tail results in a *fractional-rate loss*.
- *Tailbiting convolutional codes* operate such that the initial and terminating states are the same (but not necessarily all-zeros).
  - Tailbiting codes don't require a tail and have no fractional-rate loss.
  - *More on tailbiting codes later...*

Trellis representation

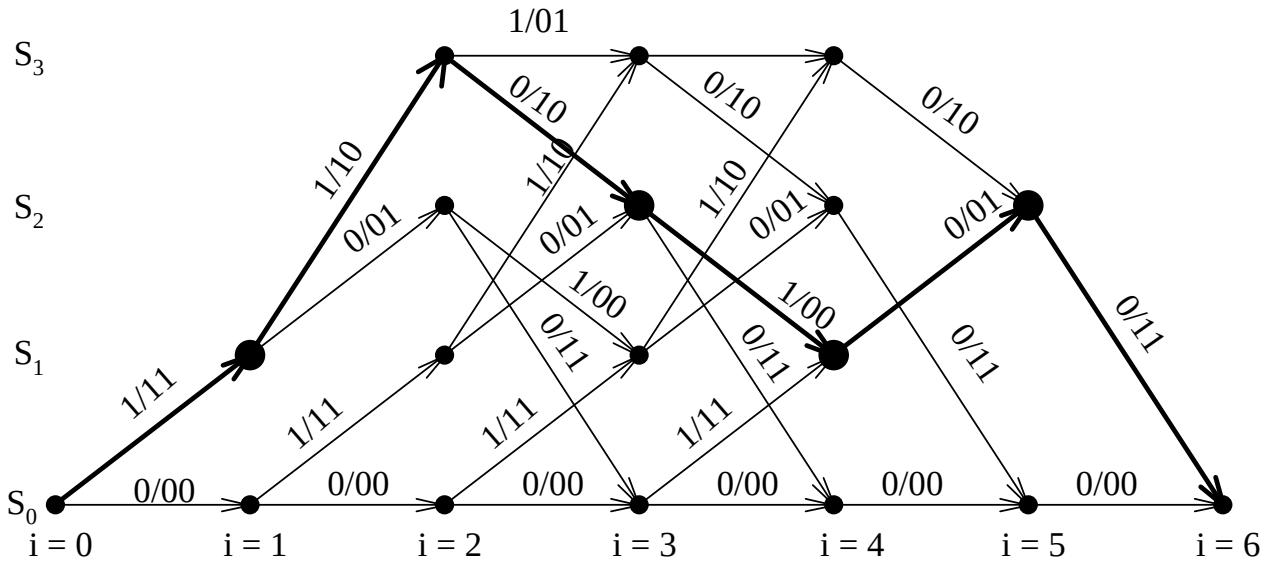
A *trellis* is an expansion of the state diagram which explicitly shows the passage of time.



Encoding Using the Trellis

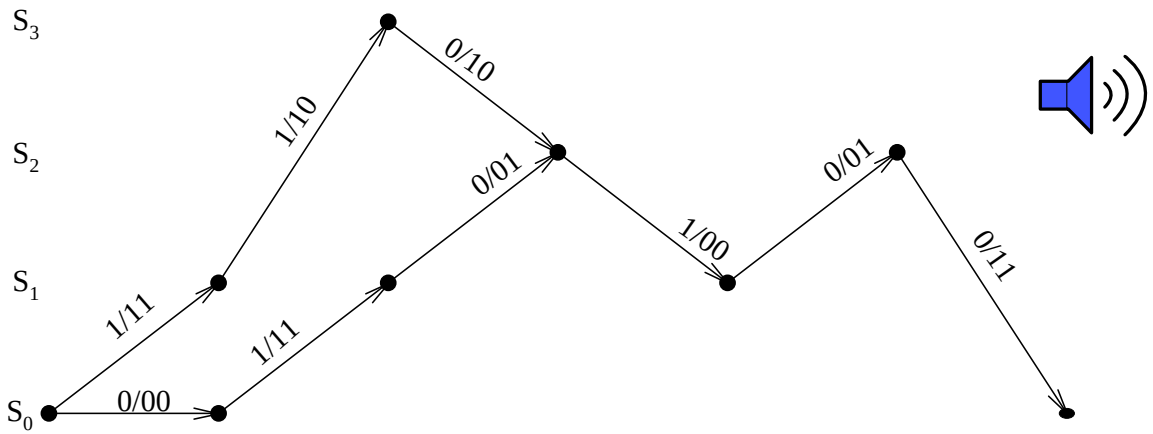
The trellis can be used to encode the message.
Use message bits to determine path, then read off the code bits.

`u = [1 1 0 1]`



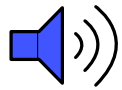
`c = [1 1 1 0 1 0 0 0 0 1 1 1]`

The Viterbi Algorithm



- The Viterbi algorithm is used for ML Decoding.
- Exploiting the recursive structure of the trellis minimizes complexity.
- Steps:
  - A *forward sweep* through the trellis is performed.
  - Each node holds a *partial path metric*.
  - A *branch metric* is computed for each branch in the trellis.
  - At each node, an *add-compare-select* operation is performed.
  - Once the end of the trellis is reached, a traceback operation determines the value of the data bits.

Viterbi Algorithm: Example



- Suppose that the input to the convolutional encoder is:

$$\mathbf{u} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

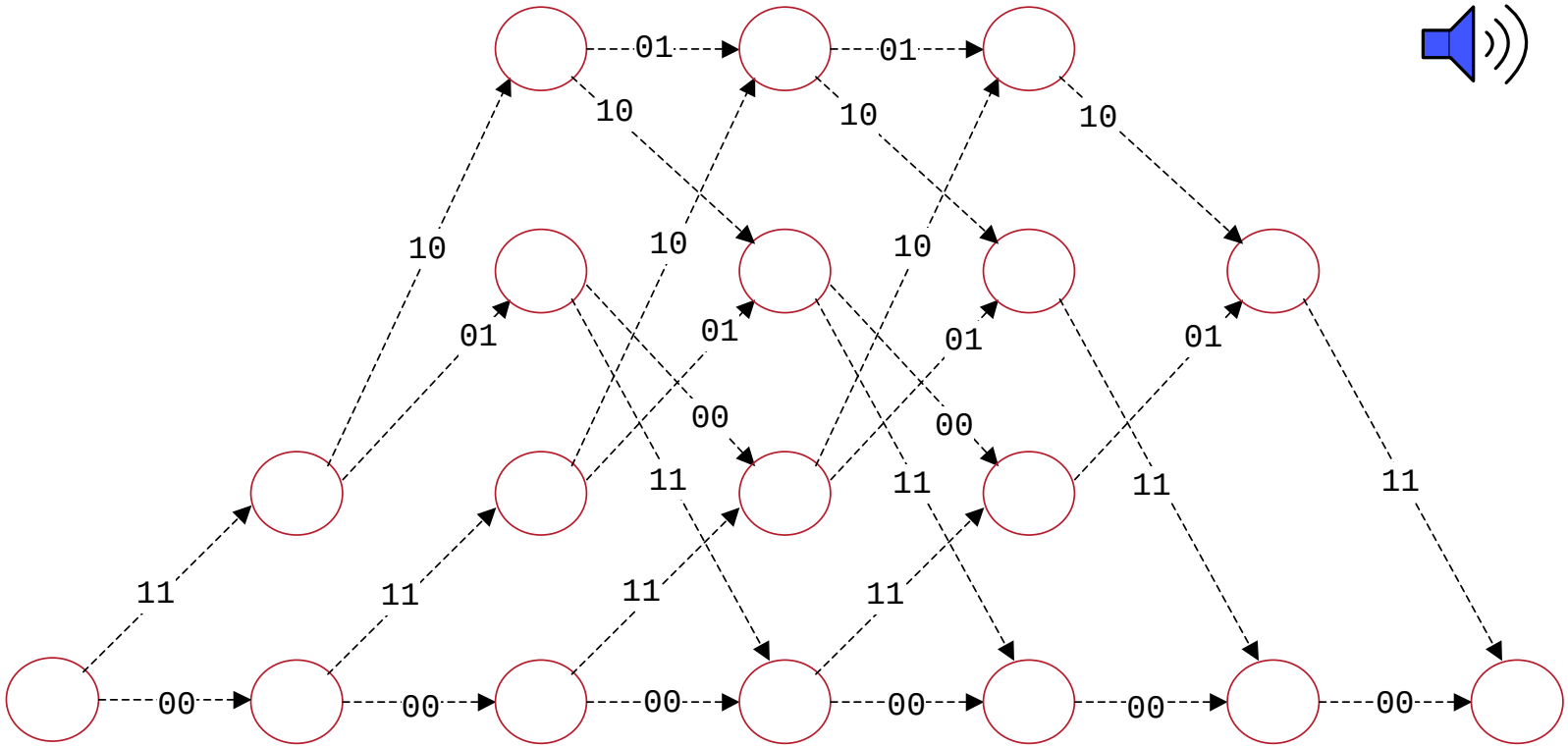
- Then the ouput of the encoder is:

$$\mathbf{c} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

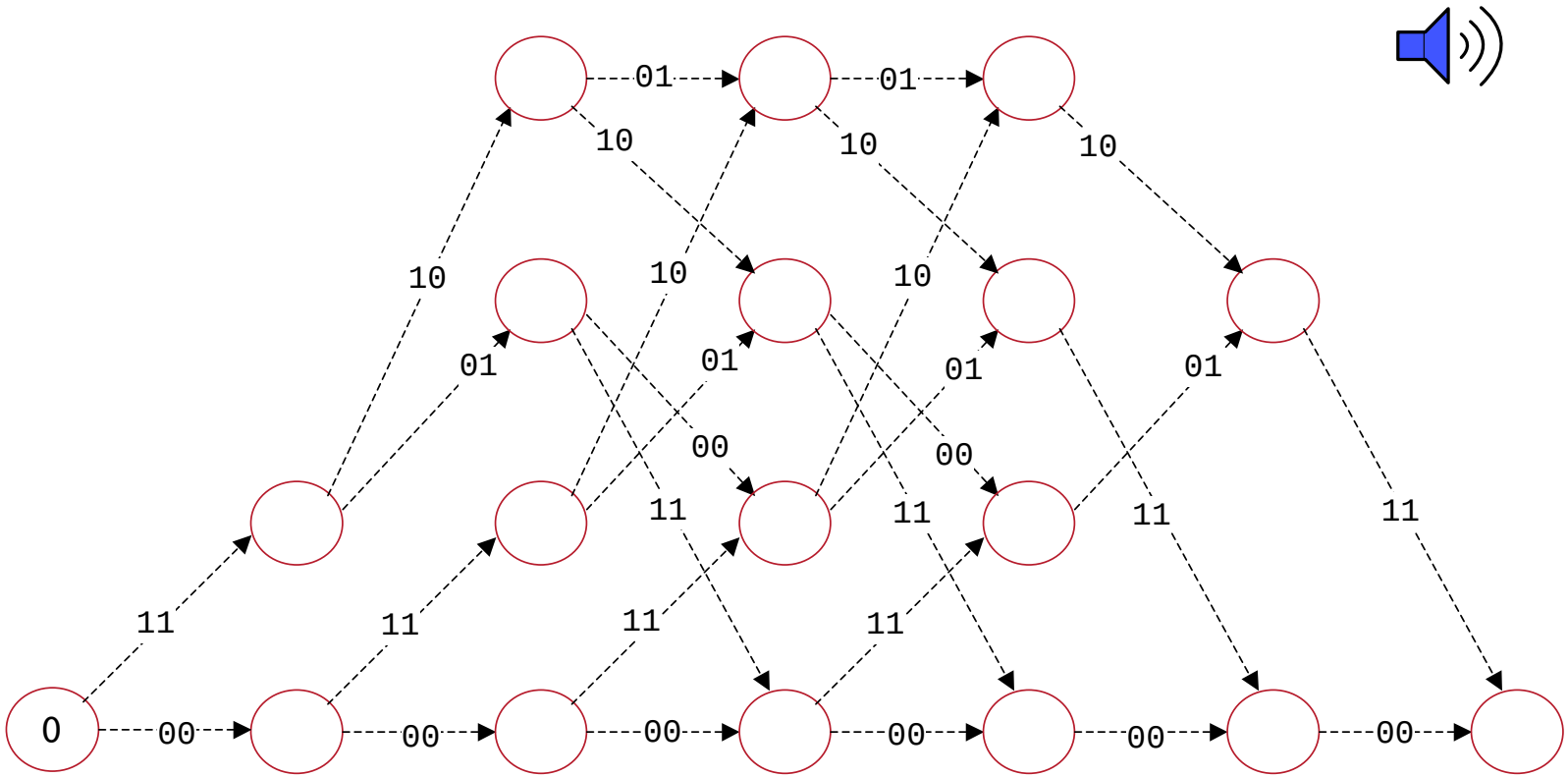
- Suppose every fourth bit is received in error:

$$\mathbf{r} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

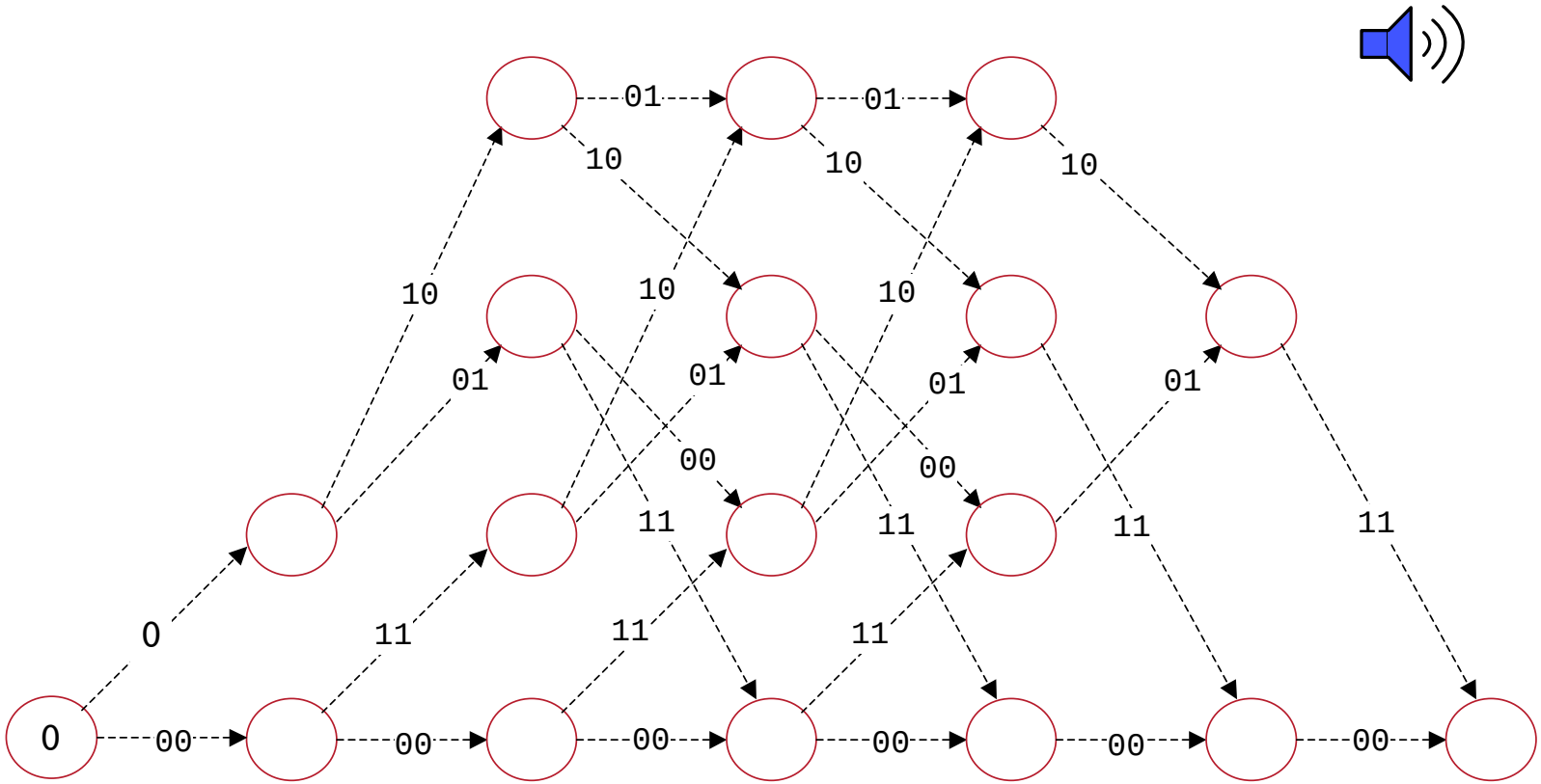
- Determine the most likely $\mathbf{u}$ given $\mathbf{r}$.
- For clarity, we will assume hard-decision decoding.



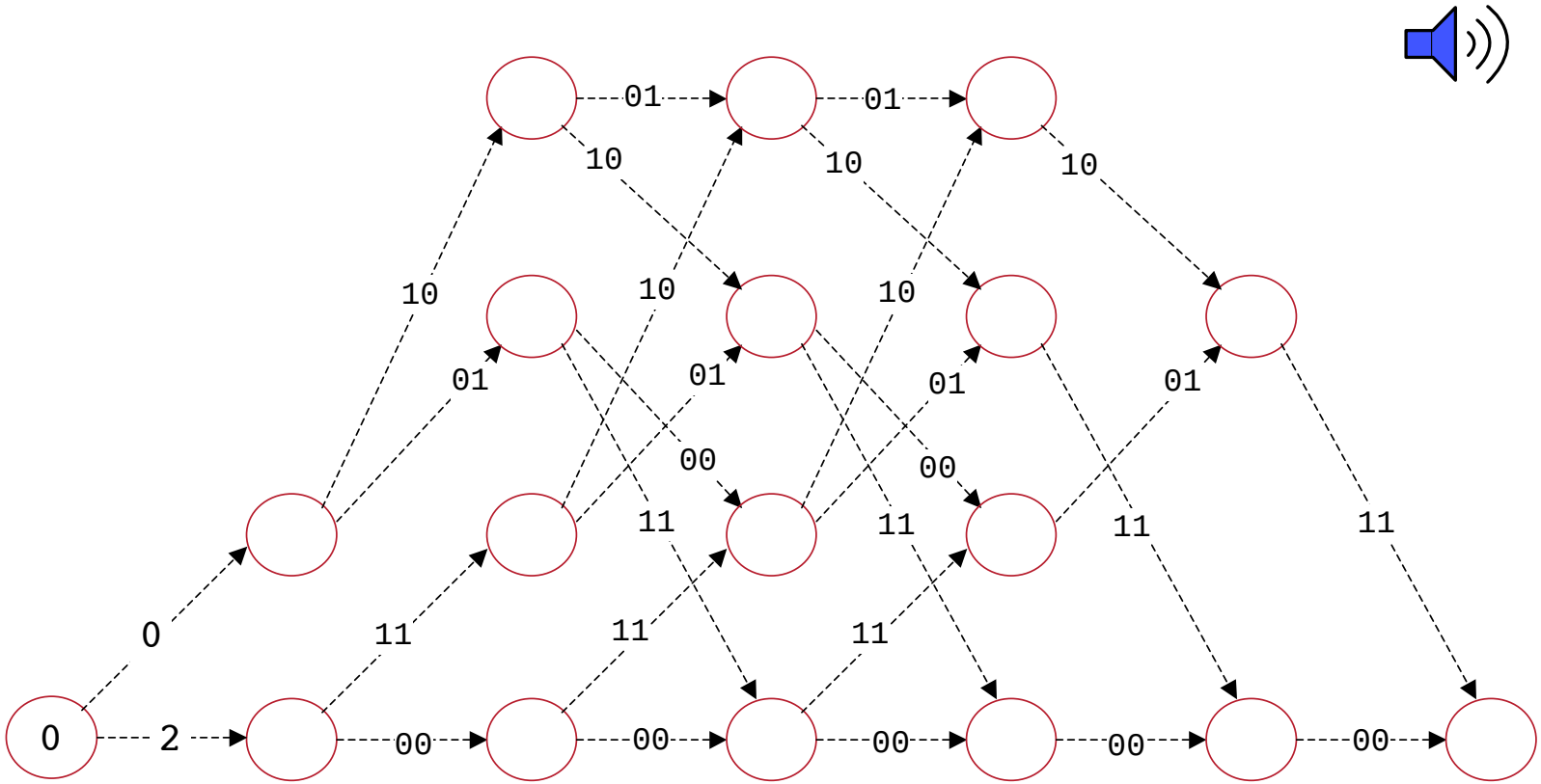
r = [1 1 1 1 1 0 0 1 0 1 1 0]



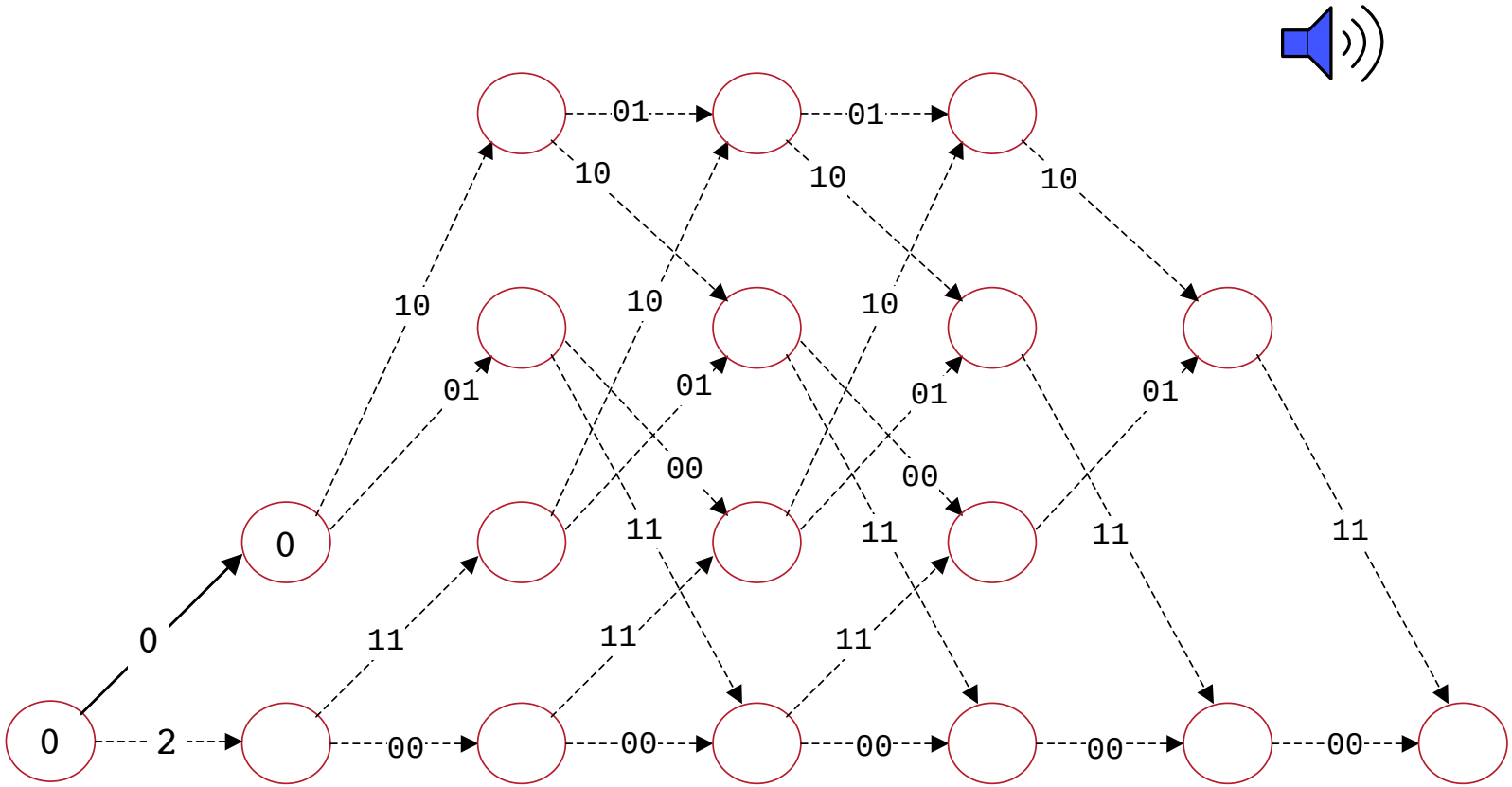
r = [1 1 1 1 1 0 0 1 0 1 1 0]



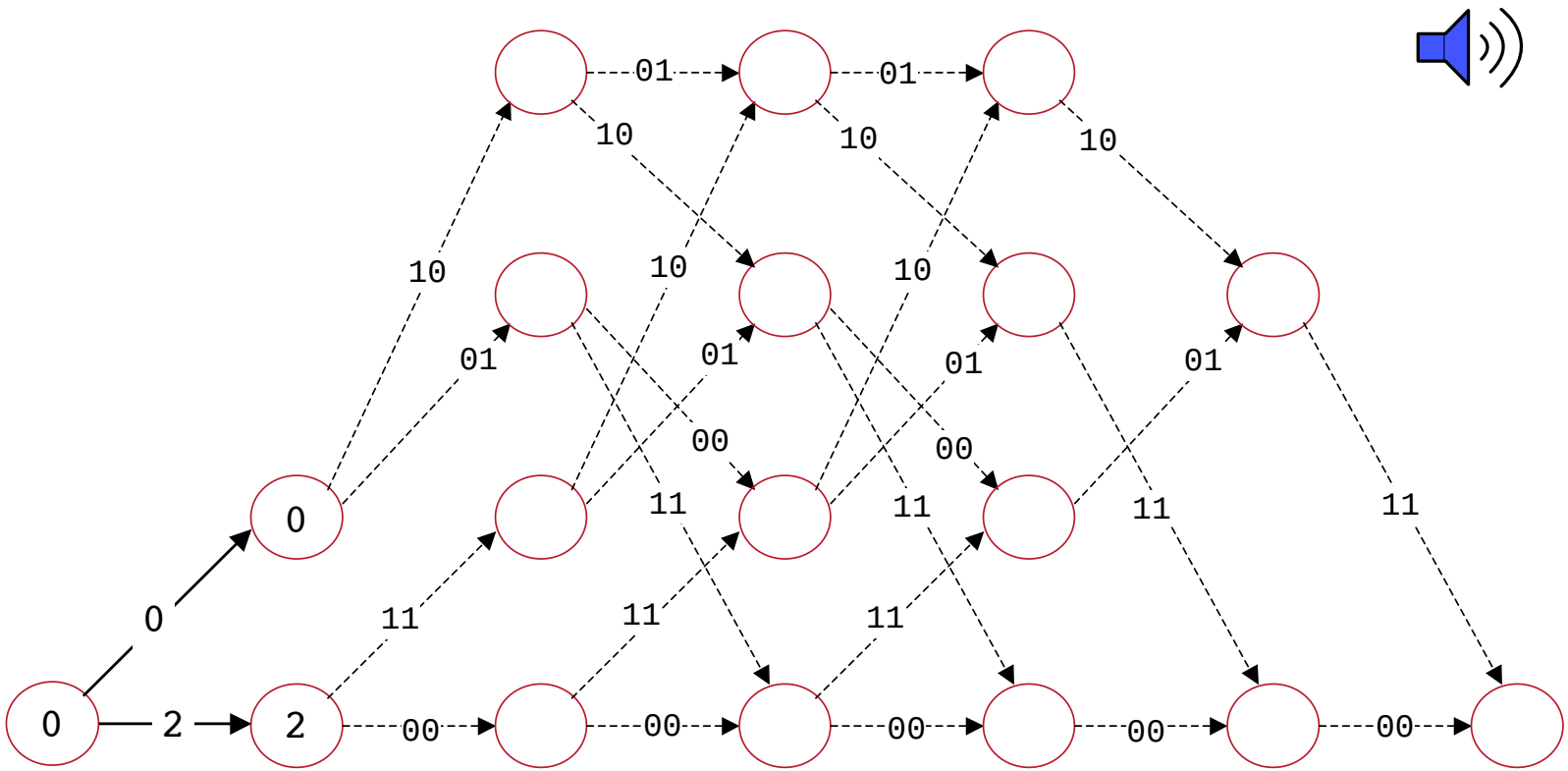
r = [1 1 1 1 1 0 0 1 0 1 1 0]



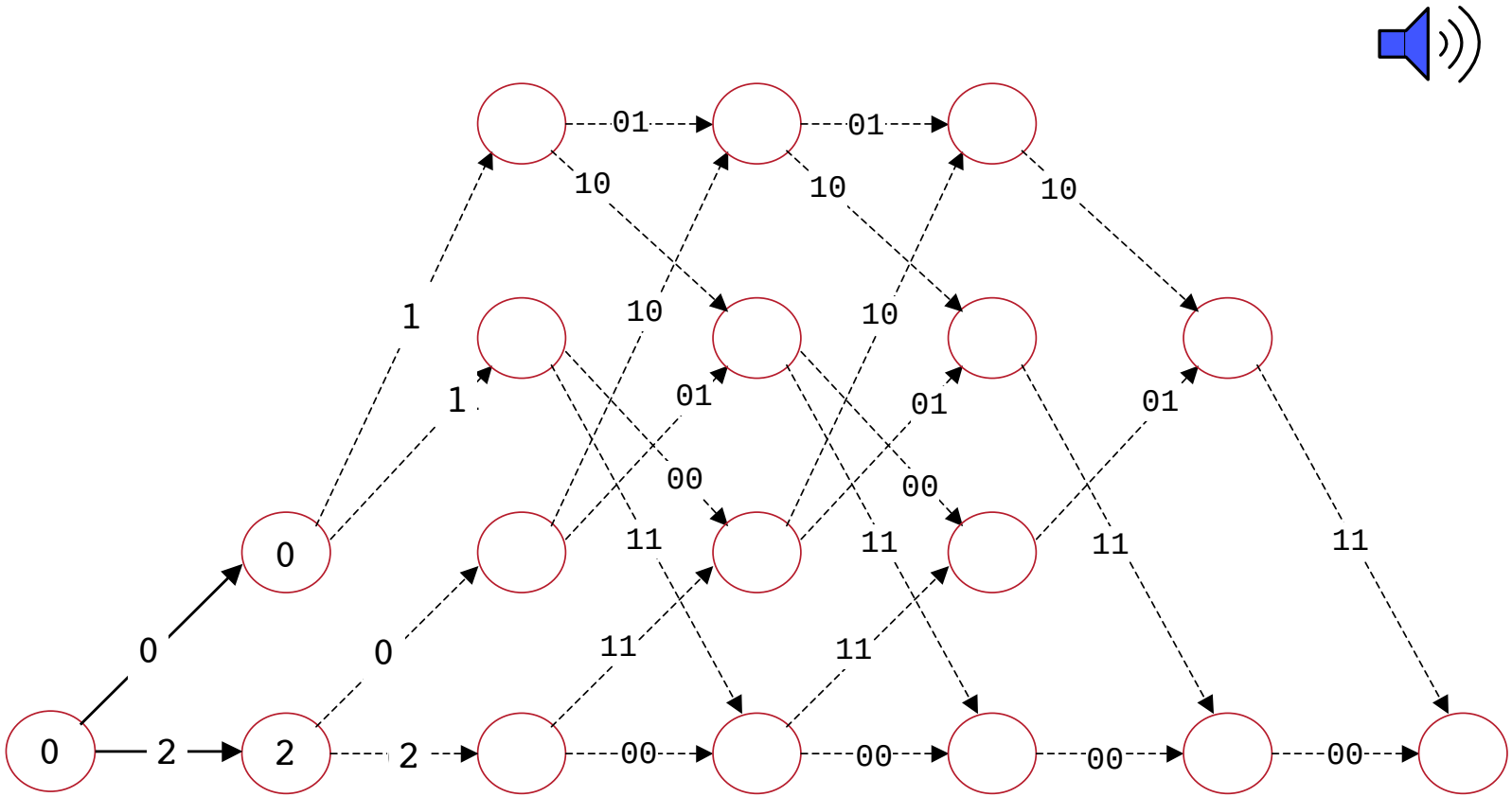
r = [1 1 1 1 1 0 0 1 0 1 1 0]



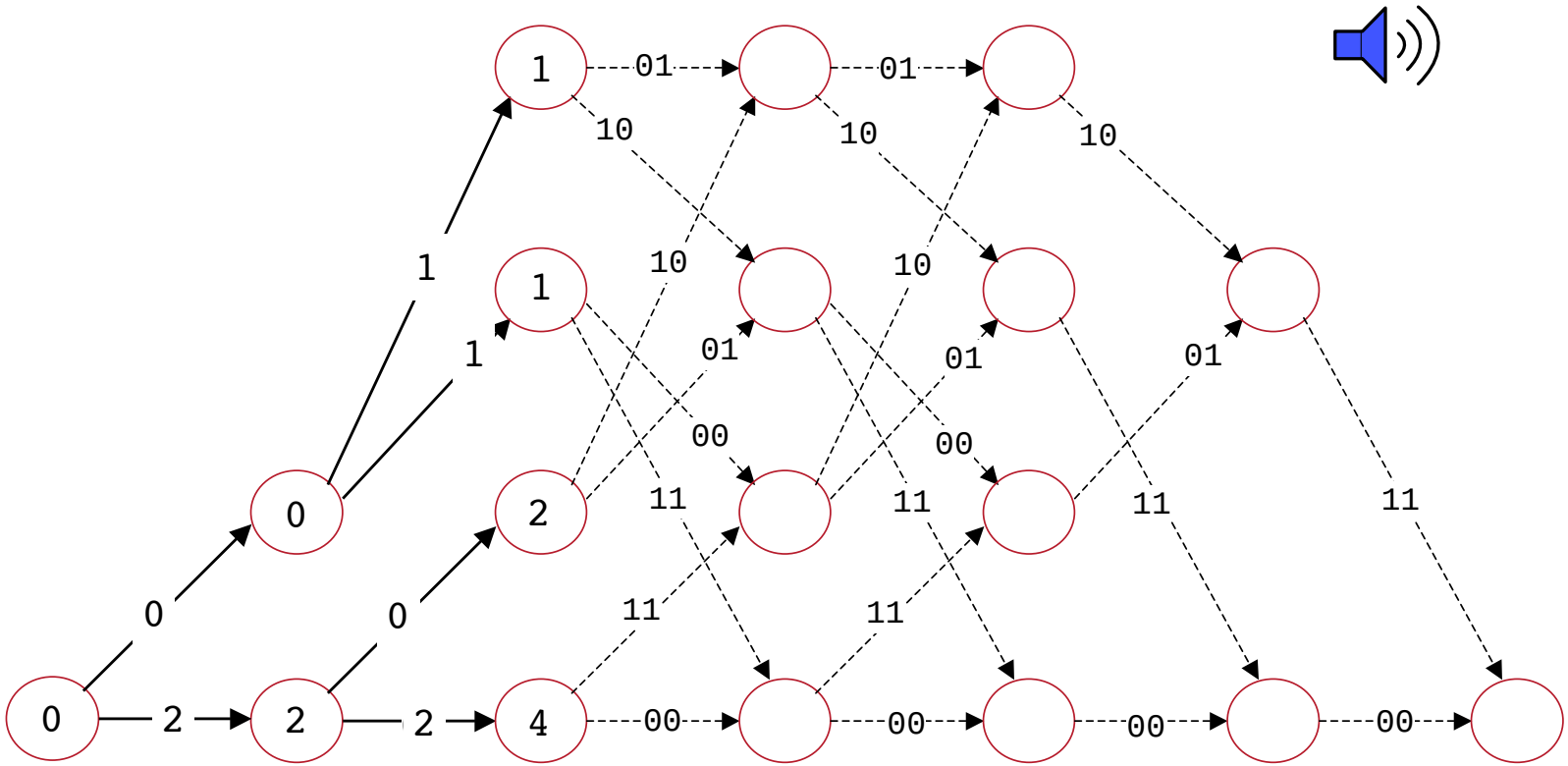
r = [1 1 1 1 1 0 0 1 0 1 1 0]



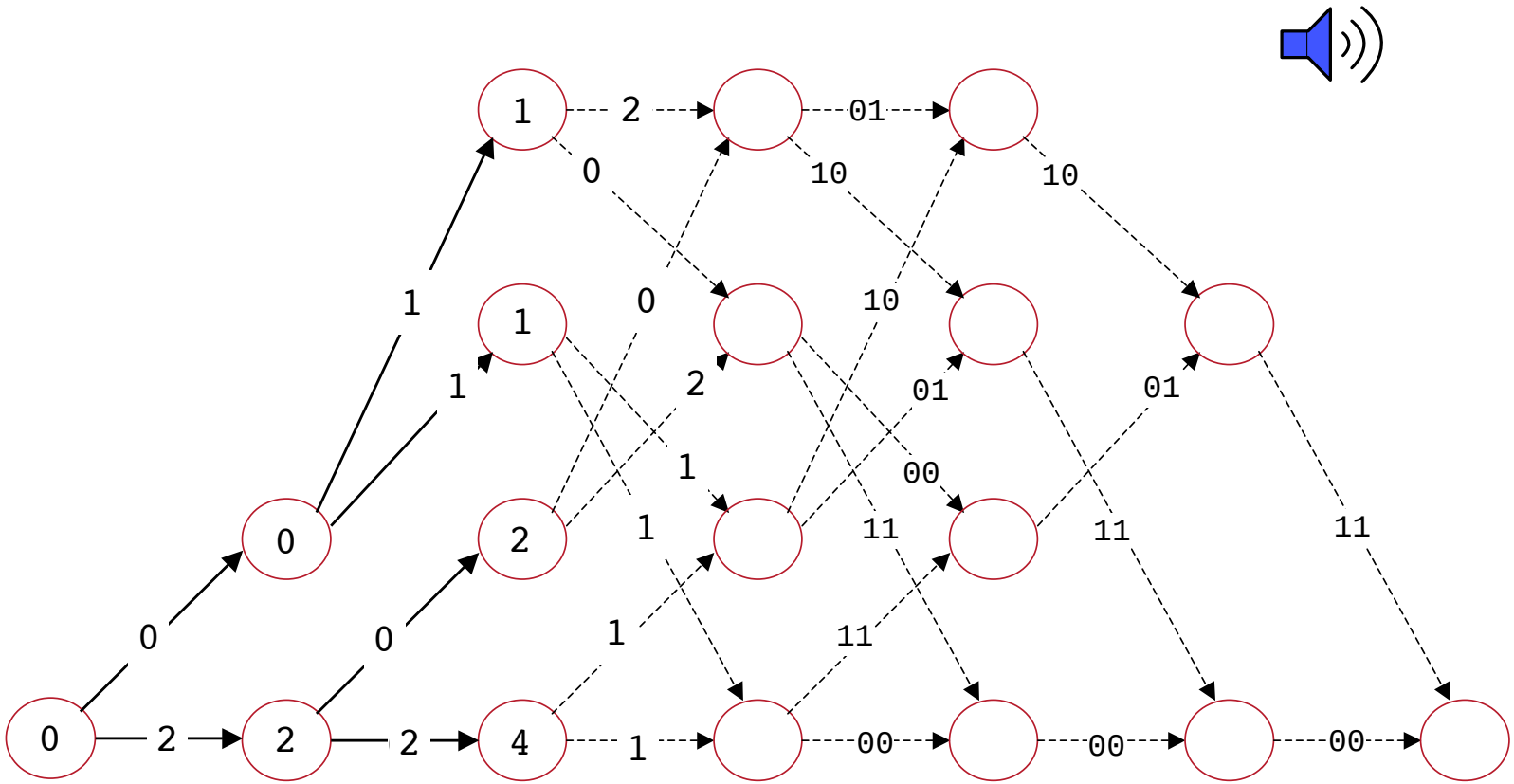
r = [1 1 1 1 1 0 0 1 0 1 1 0]



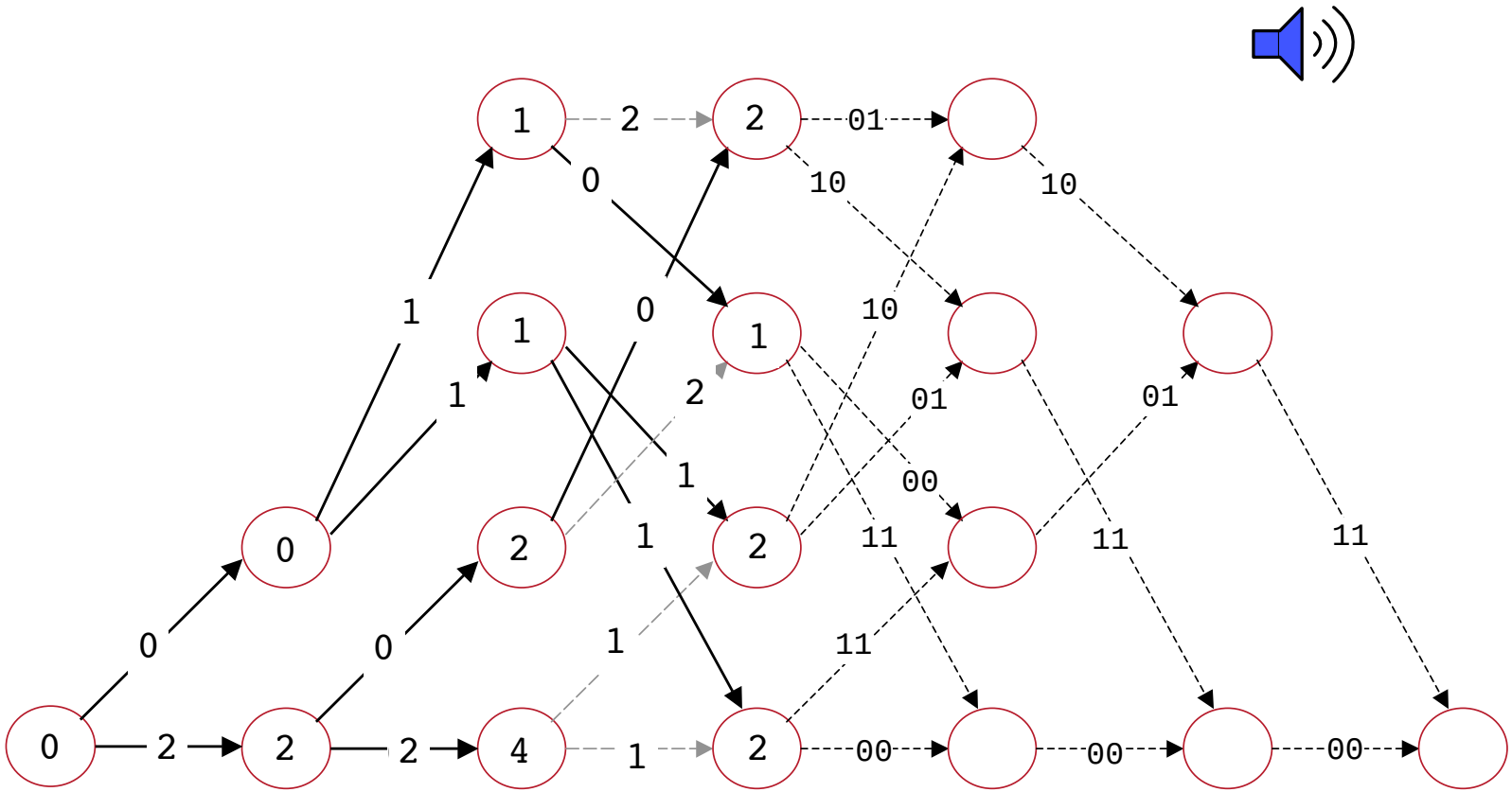
r = [1 1 1 1 1 0 0 1 0 1 1 0]

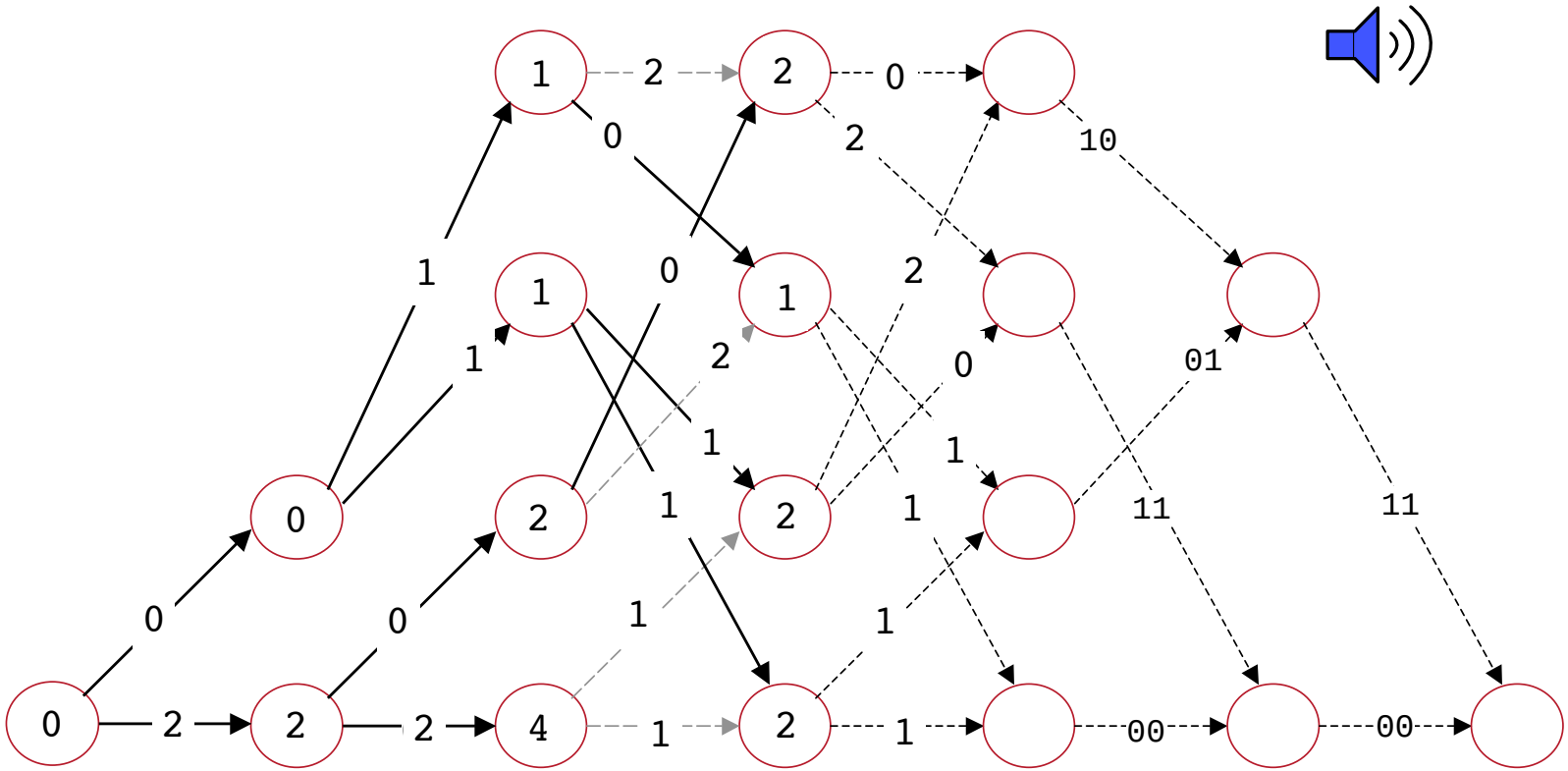


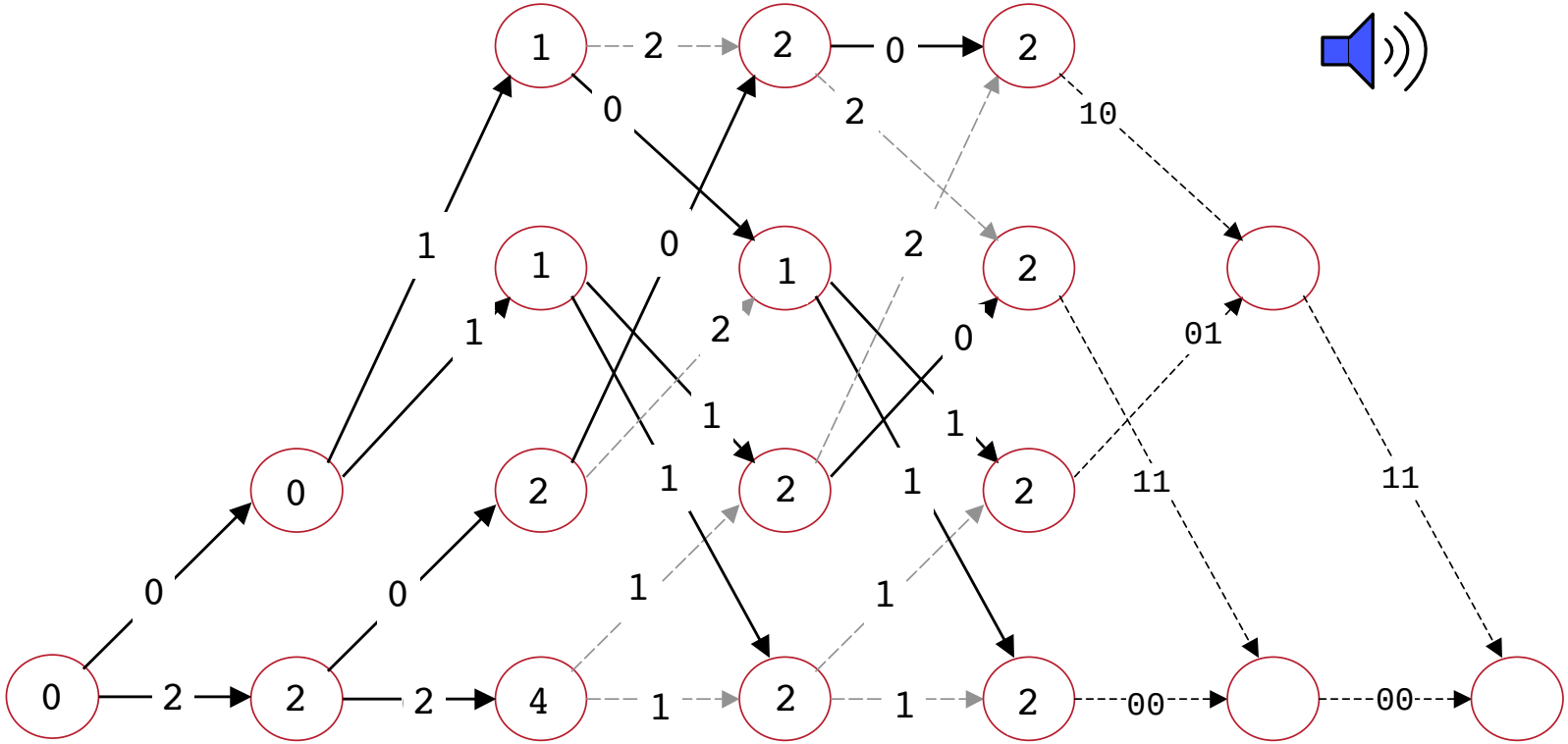
r = [1 1 1 1 1 0 0 1 0 1 1 0]

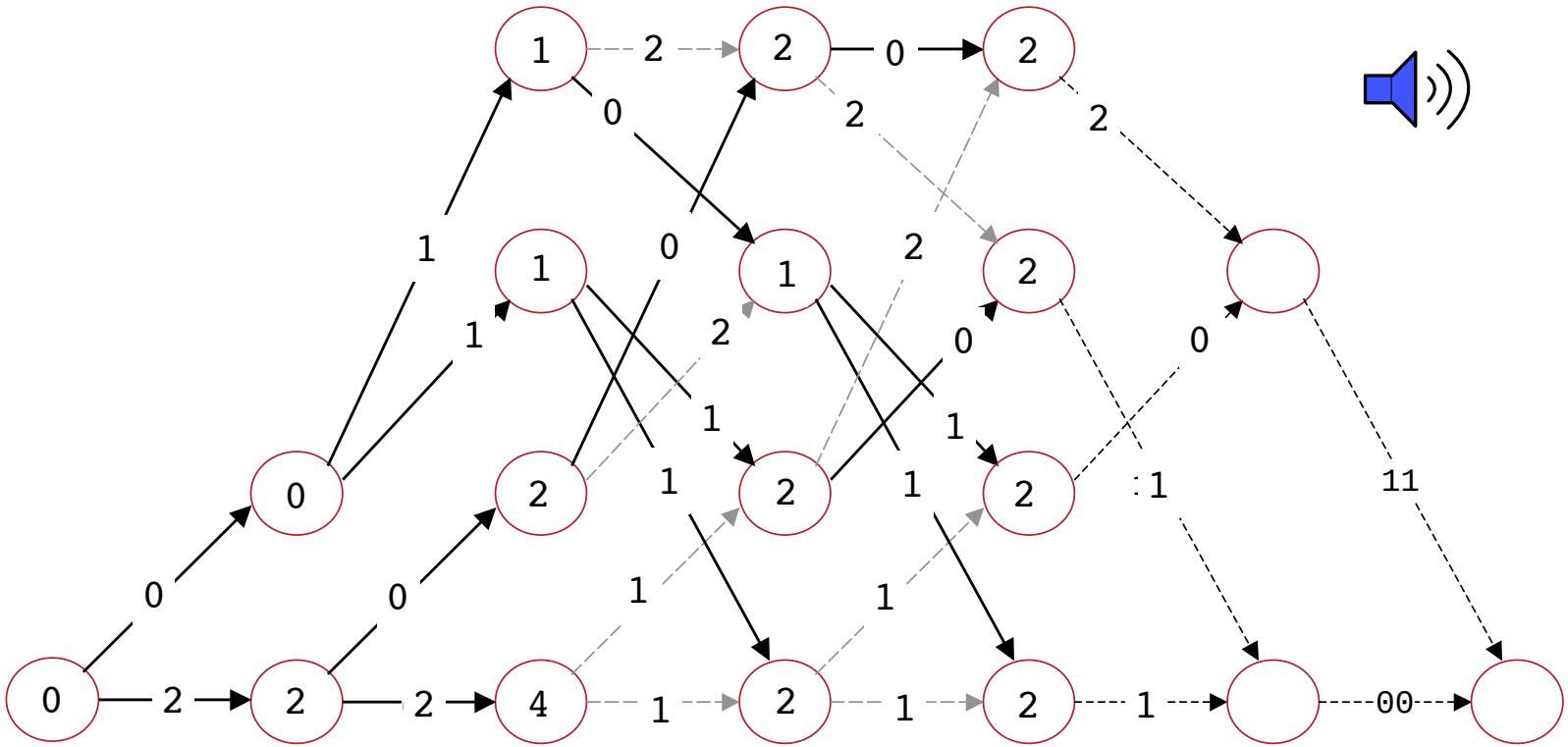


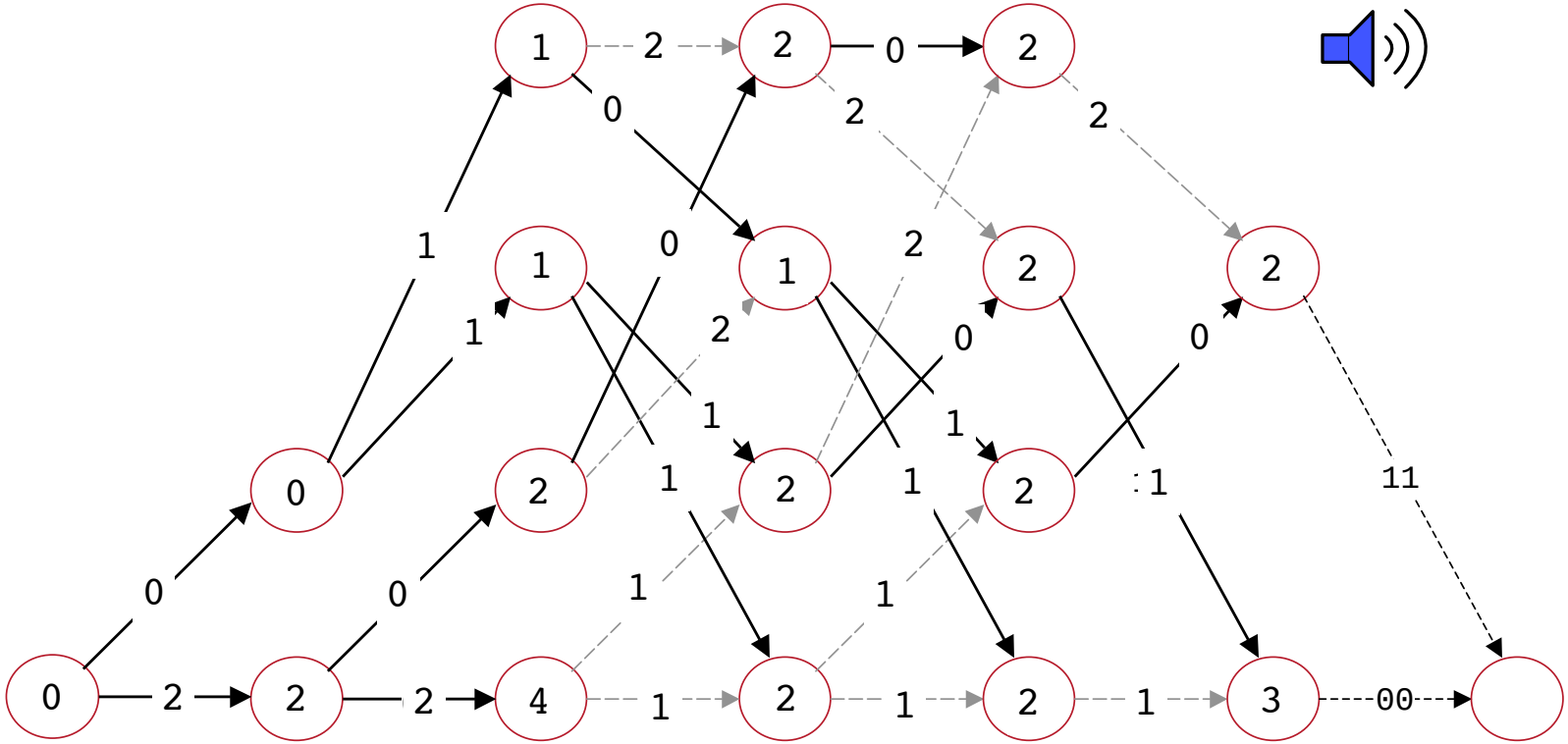
r = [1 1 1 1 1 0 0 1 0 1 1 0]


r = [1 1 1 1 1 0 0 1 0 1 1 0]

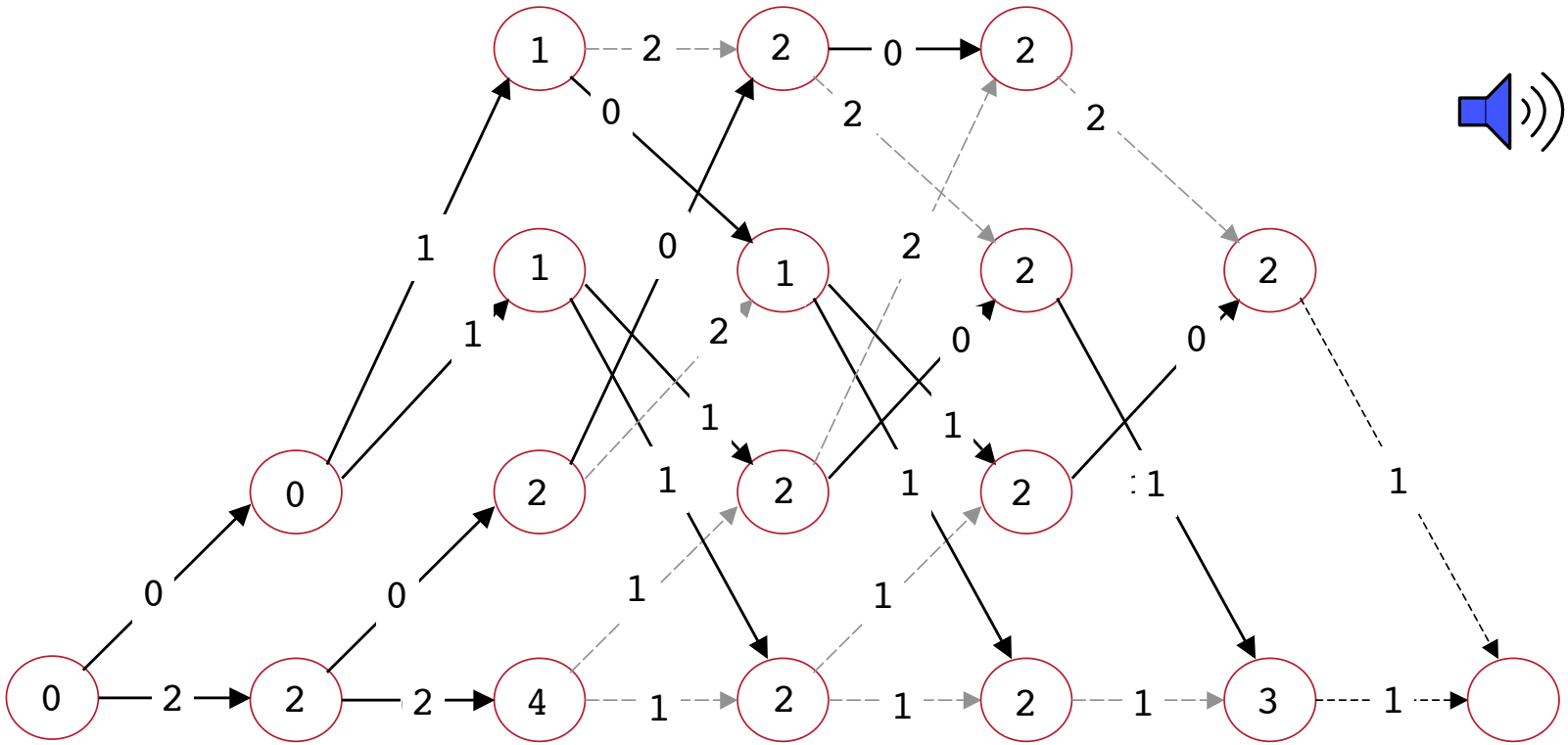

r = [1 1 1 1 1 0 0 1 0 1 1 0]


r = [1 1 1 1 1 0 0 1 0 1 1 0]

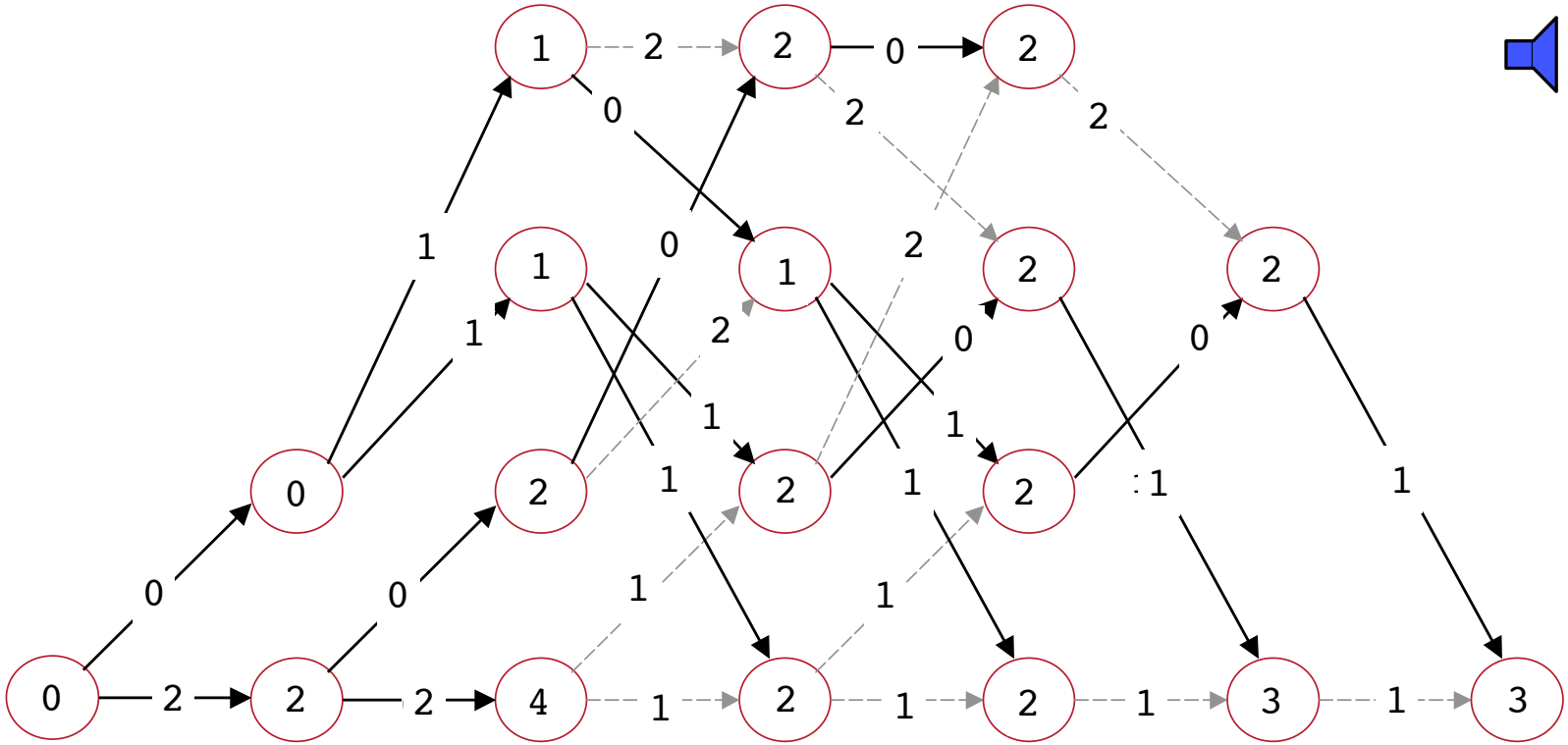

r = [1 1 1 1 1 0 0 1 0 1 1 0]

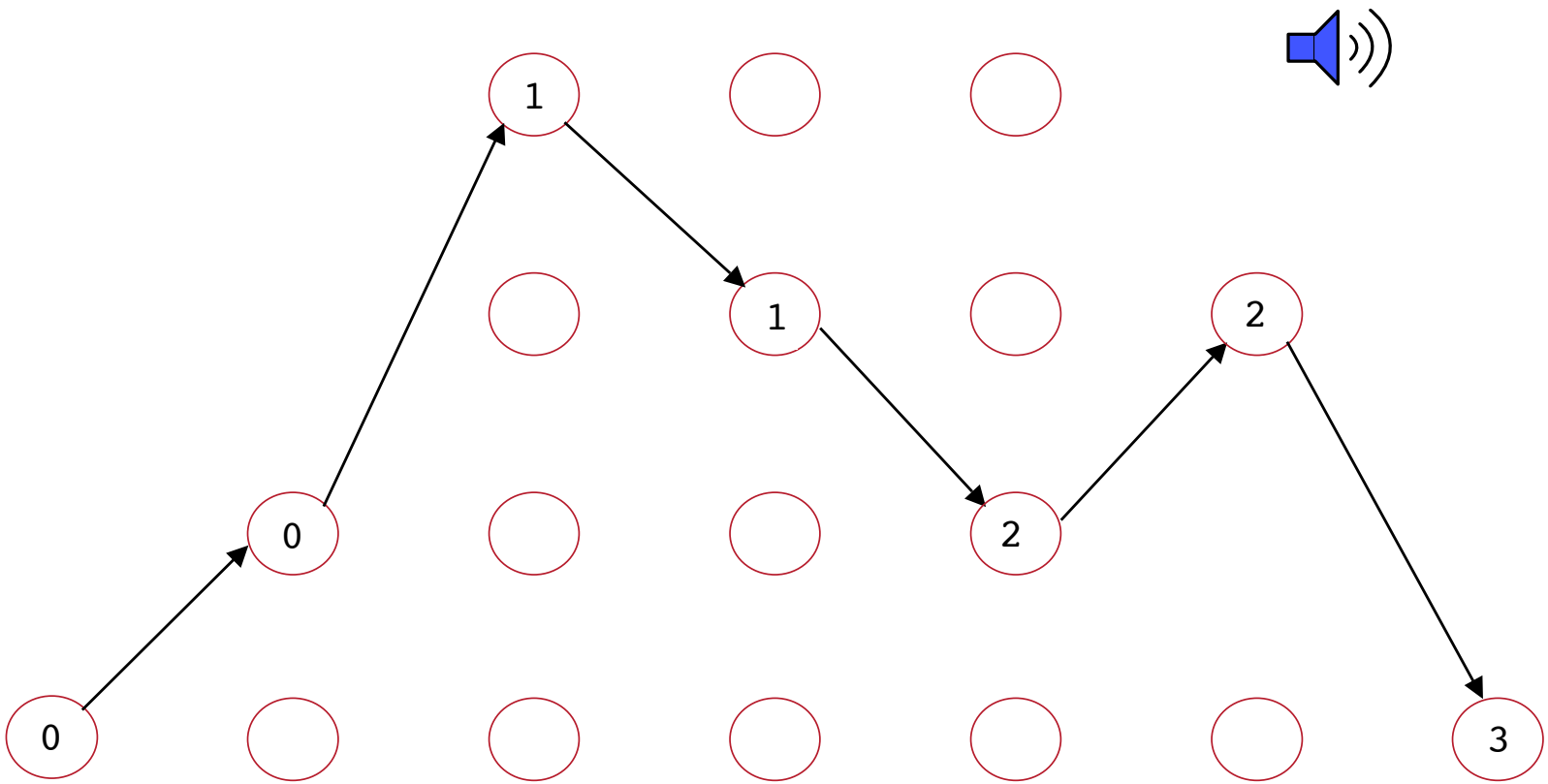


r = [1 1 1 1 1 0 0 1 0 1 1 0]

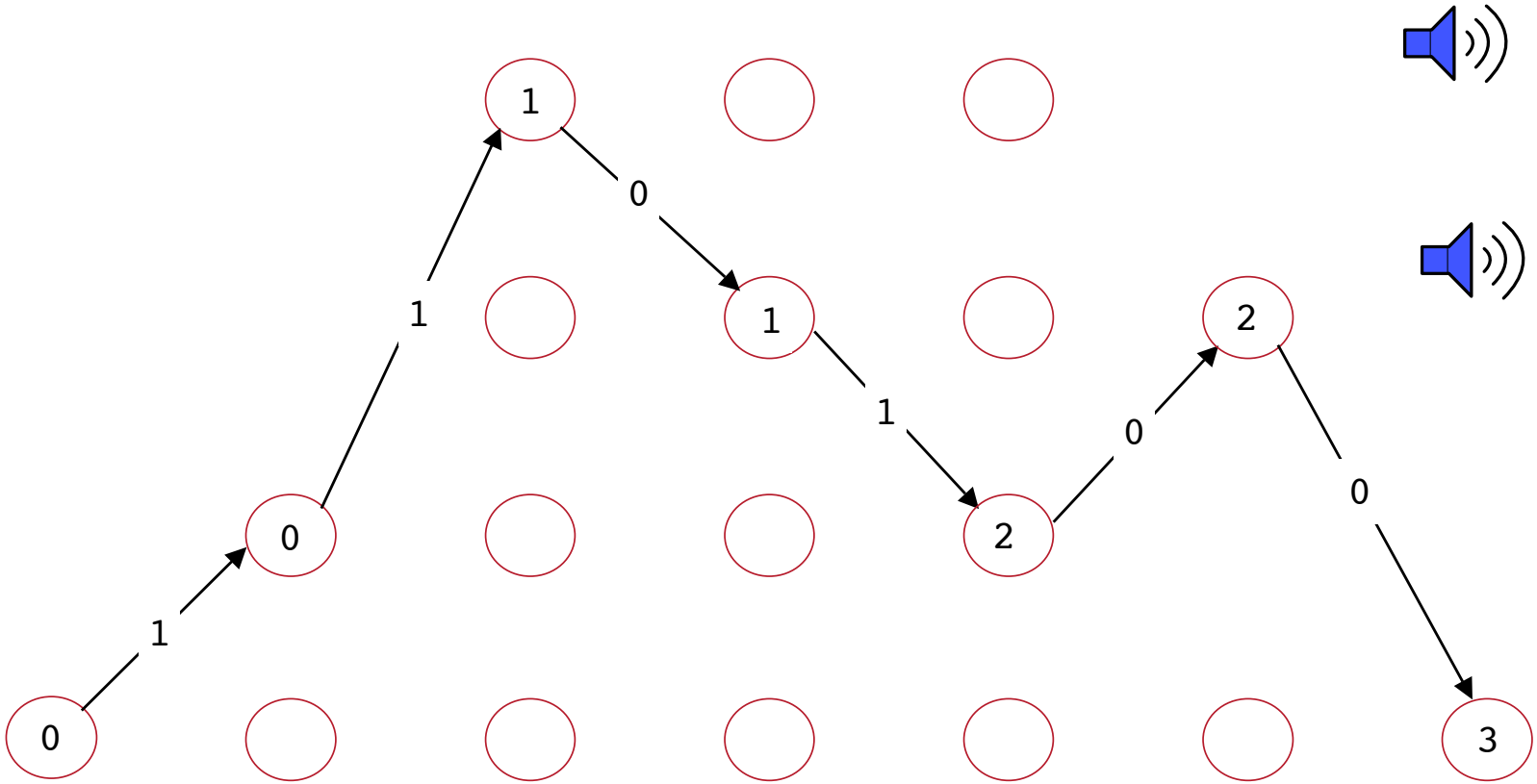


r = [1 1 1 1 1 0 0 1 0 1 1 0]


0 2 2 2 4 1 2 1 2 1 2 1 3 1 3
0 0 1 1 2 1 2 1 2 1 2 1
1 1 0 2 1 2 0 2 1 2 0 2
r = [1 1 1 1 1 0 0 1 0 1 1 0]



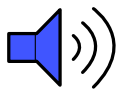
r = [1 1 1 1 1 0 0 1 0 1 1 0]


1
0
1
1
0
1
1
2
0
2
0
3
0
1
1
r = [1 1 1 1 1 0 0 1 0 1 1 0]

Soft-decision Decoding

- Instead of using hard-decisions on the bits, soft-decisions could be used.
- Requires that the input to the decoder be a *log-likelihood ratio* (LLR) in the form:

$$\lambda_i = \log \frac{P[c_i = 1|r_i]}{P[c_i = 0|r_i]}$$



- For BPSK modulation in AWGN, the LLR is

$$\lambda_i = \frac{2}{\sigma^2} r_i$$

- The branch metric for a particular state transition $S_j \rightarrow S_\ell$ is:

$$\gamma_{j,\ell} = \sum_{i=0}^{n_c-1} c_i \lambda_i$$

- Goal is to *maximize* the metric, rather than minimize it.
- Thus, the ACS will select the *larger* branch instead of the *smaller* one.