CS 425
Assignment 1

Name: Utsav Shah, Roll No.: 2203128
Name: Mithun P, Roll No.: 2203314

Hamming Ball Emulator

Overview:

The Hamming Ball Emulator is a web-based tool that helps visualize linear codes in coding theory. It allows users to input a generator matrix for a linear code over F_2 (binary field) and visualizes the codewords and Hamming balls.

Files and Their Functions:

1. **index.html:**
   This file creates the structure of the webpage with:
   - Input area for the generator matrix
   - Information display sections
   - Visualization area for Hamming balls
   - Control panels for adjusting the visualization
2. **styles.css:**

   This file handles the visual appearance of the webpage:

   - Colors, fonts, and spacing
   - Layout of components
   - Visual styles for elements (buttons, input areas, etc.)
3. **script.js:**

   This file contains all the logic and functionality:

   Main functions:

   1. processMatrix():

   - Purpose: This is the core function of the application that runs when the user clicks the "Process Matrix" button. It handles the entire workflow from taking input to displaying results.
   - Steps: The function takes the text input, clears any previous errors or results, converts the input text into a matrix, validates it, calculates various code properties, updates the information display, and finally creates the visual graph representation.

- Logic: Using a try/catch structure to handle potential errors, this function verifies if the matrix is valid. If valid, it calculates important properties like code dimensions [n,k], code rate (k/n), all possible codewords, minimum Hamming distance, and error detection/correction capabilities. Once calculations are complete, it calls the `displayNetwork()` function to create the visual representation.

2. parseMatrix(matrixText):

- Purpose: This function converts the user's text input into a proper matrix format that can be used for calculations.
- Steps: It splits the input text by line breaks to identify rows, then splits each row by spaces or commas to get individual elements. It checks if each element is valid (either 0 or 1) and ensures all rows have the same length before returning a usable 2D array.
- Logic: The function carefully processes the raw text input to transform it into a structured 2D binary matrix. It includes validation steps to ensure only binary values (0 and 1) are accepted and that the matrix has a consistent structure.

3. validateMatrix(matrix):

- Purpose: This function verifies that the matrix meets the requirements to be a valid generator matrix for a linear code.
- Steps: It checks basic requirements first, ensuring the matrix has proper dimensions and that the number of rows (k) is less than or equal to the number of columns (n). It then uses Gaussian elimination to check if the rows are linearly independent and calculates the rank of the matrix.
- Logic: The function confirms the matrix can properly generate a linear code by verifying its basic structure and then using more advanced linear algebra (Gaussian elimination) to ensure the rows are linearly independent. If the rank equals k, the matrix is considered valid.

4. generateCodewords(G):

- Purpose: This function creates all possible codewords that can be generated from the given generator matrix.
- Steps: For a code with dimensions [n,k], it generates all 2^k possible messages, multiplies each message with the generator matrix using modulo-2 arithmetic, and stores the resulting codewords.
- Logic: This implements the fundamental operation of linear codes (c = m×G), where each k-bit message is multiplied by the generator matrix to produce an n-bit codeword. It uses binary field operations to ensure all calculations are done in modulo-2 arithmetic.

5. hammingDistance(word1, word2):

- Purpose: This function calculates how many bit positions differ between two binary words.
- Steps: It compares each position in the two input words, counts the positions where bits differ, and returns the total count.
- Logic: The function performs a straightforward bit-by-bit comparison between two binary words. This distance metric is fundamental to understanding how "far apart" different codewords are in the code space.

6. minimumHammingDistance(words):

- Purpose: This function finds the smallest Hamming distance between any pair of codewords in the code.
- Steps: It compares every possible pair of codewords using a double loop, calculates the Hamming distance for each pair, and keeps track of the smallest distance found.
- Logic: The minimum distance is a critical parameter that determines the error correction capability of the code. The function systematically examines all pairs to find this crucial value.

7. errorDetectionAndCorrection(minHammingDistance):

- Purpose: This function calculates the error detection and correction capabilities of the code based on its minimum Hamming distance.
- Steps: It applies standard formulas to determine how many errors the code can detect and correct, specifically calculating detection capacity as (minimum distance - 1) and correction capacity as floor((minimum distance - 1) / 2).
- Logic: The function translates the abstract minimum distance value into practical error-handling capabilities, providing insight into how many bit errors the code can reliably detect and correct.

8. generateWords(n):

- Purpose: This function creates all possible binary words of a given length n.
- Steps: For n bits, it generates all 2^n possible binary combinations by converting each number from 0 to 2^n-1 to its binary representation and padding with zeros to maintain consistent length.
- Logic: This function explores the entire space of possible received words, which is essential for visualizing how the code partitions this space into regions around each codeword.

9. displayNetwork(codewords, allWords, selectedWord, customRadius):

- Purpose: This function creates the visual graph showing codewords and their relationships to other words in the space.
- Steps: It operates in two modes: a full view showing all codewords with their Hamming balls, or a selected view showing one codeword with words at exactly the specified distance. It creates nodes for all words (red for codewords, blue for others) and edges connecting words within specified Hamming distances.

Visualization Functions:

1. displayNetwork(codewords, allWords, selectedWord, customRadius):

- **Purpose**: This function creates the visual graph representation of the code structure, showing the relationships between codewords and other binary words in the space.
- **Steps**: The function first determines which view mode to use (full view or selected word view), then creates nodes for all relevant words and establishes connections between them based on Hamming distance criteria. It styles nodes differently based on whether they are codewords or non-codewords, and finally updates the user interface elements to match the current visualization.
- **Logic**: In full view, the function shows all codewords and their surrounding Hamming balls, creating a comprehensive picture of the code's structure. In selected view, it focuses on a single codeword and only shows words at exactly the specified Hamming distance, allowing for detailed exploration of specific error patterns. The function creates a network structure that is then laid out by the force-directed algorithm to visually represent the mathematical relationships between words.

2. forceAtlas2Layout(graph, iterations):

- **Purpose**: This function calculates optimal positions for all nodes in the graph to create a readable and meaningful visual layout.
- **Steps**: The function begins by assigning random initial positions to all nodes, then iteratively applies physical forces to determine better positions. For each iteration, it calculates repulsion forces between all pairs of nodes, attraction forces along edges, and gravity forces pulling toward the center. It then updates node positions based on these combined forces. Finally, it performs post-processing to scale and center the resulting layout.
- **Logic**: By simulating physical forces, this algorithm naturally groups related nodes and separates unrelated ones. Nodes that are connected by edges (indicating close Hamming distance) are pulled together by spring-like forces, while all nodes repel each other to prevent overlap. The gravity component ensures the layout doesn't spread infinitely. Through multiple iterations, this

process converges toward a stable arrangement that visually conveys the underlying structure of the code.

- **Reference: [** Link] This function is based on the Force Atlas 2 algorithm, which is a force-directed graph layout algorithm commonly used for network visualization. We have implemented a more straightforward version of this algorithm.

3. updateHammingBallInfo(selectedWord, currentRadius):

- **Purpose**: This function updates the information panel with relevant details about the Hamming balls currently being displayed in the visualization.
- **Steps**: The function determines which view mode is active and gathers appropriate statistics. In the selected view, it calculates how many words are at exactly the specified distance from the chosen codeword. In full view, it provides general information about all Hamming balls. It then updates the user interface with this information in a readable format.
- **Logic**: This function serves as a bridge between the visual representation and the mathematical concepts, helping users understand what they're seeing. It dynamically updates as users interact with the visualization, providing context about Hamming distances, sphere sizes, and error correction capabilities. This contextual information enhances the educational value of the visualization by connecting the visual patterns to the underlying coding theory concepts.

Interactive Elements:

1. Reset Button:
    - Logic: Resets to the full view showing all codewords and their relationships
    - Called when user clicks "Reset to Full View"
2. Apply Radius Button:
    - Logic: Updates the visualization to show words at exactly the selected distance
    - Called when user changes the radius and clicks "Apply Radius"
3. Node Click Event:
    - Logic: Detects when a codeword node is clicked and switches to selected view
    - Only triggers for codeword nodes (red nodes)
    - Sets the initial radius to 1 when a codeword is first selected

How to Use the Tool:

1. Enter a generator matrix in the text area (rows of 0s and 1s).
2. Click "Process Matrix" to analyze the code.
3. View the information about your code (parameters, distance, etc...
4. Explore the visualization:
    - Red nodes are codewords (centers of Hamming balls).
    - Blue nodes are other words.
    - Lines show Hamming distance connections.
5. Click on any codeword (red node) to focus on its Hamming ball.
6. Adjust the radius using the dropdown menu when focused on a codeword.
7. Click "Reset to Full View" to return to the complete visualization.

Some Technical Details:

- The tool uses Sigma.js and Graphology libraries for graph visualization.
- The layout uses a force-directed algorithm to position nodes.
- Hamming balls are calculated using Hamming distance measurements.
- The tool handles binary linear codes of reasonable size (up to n=10 recommended).