Title: Write a program for error detection and correction for 7/8 bits ASCII codes using HammingCodes.

HammingCodes Technic

Hamming code is a set of error-correction codes that can be used to detect and correct the errors that can occur when the data is moved or stored from the sender to the receiver. It is a technique developed by R.W. Hamming for error correction. Redundant bits – Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer.

There are two types of parity bits:

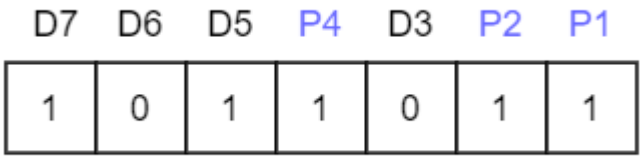
1. **Even parity bit:** In the case of even parity, for a given set of bits, the number of 1's are counted. If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1's an even number. If the total number of 1's in a given set of bits is already even, the parity bit's value is 0.
2. **Odd Parity bit:** In the case of odd parity, for a given set of bits, the number of 1's are counted. If that count is even, the parity bit value is set to 1, making the total count of occurrences of 1's an odd number. If the total number of 1's in a given set of bits is already odd, the parity bit's value is 0.

General Algorithm of Hamming code: Hamming Code is simply the use of extra parity bits to allow the identification of an error.

1. Write the bit positions starting from 1 in binary form (1, 10, 11, 100, etc).
2. All the bit positions that are a power of 2 are marked as parity bits (1, 2, 4, 8, etc).
3. All the other bit positions are marked as data bits.
4. Each data bit is included in a unique set of parity bits, as determined its bit position in binary form. **a.** Parity bit 1 covers all the bits positions whose binary representation includes a 1 in the least significant position (1, 3, 5, 7, 9, 11, etc). **b.** Parity bit 2 covers all the bits positions whose binary representation includes a 1 in the second

position from the least significant bit (2, 3, 6, 7, 10, 11, etc). **c.** Parity bit 4 covers all the bits positions whose binary representation includes a 1 in the third position from the least significant bit (4–7, 12–15, 20–23, etc). **d.** Parity bit 8 covers all the bits positions whose binary representation includes a 1 in the fourth position from the least significant bit bits (8–15, 24–31, 40–47, etc). **e.** In general, each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.

5. Since we check for even parity set a parity bit to 1 if the total number of ones in the positions it checks is odd.
6. Set a parity bit to 0 if the total number of ones in the positions it checks is even.



Code:-

```
import java.util.Scanner;

public class Hamming {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter 4 bits of data: ");
        String inputData = scanner.nextLine();


        if (inputData.length() != 4) {
            System.out.println("Please enter 4 bits of data.");
            return;
        }

        String encodedData = encodeHammingCode(inputData);
        System.out.println("Encoded Data: " + encodedData);

        System.out.print("Enter received data (with possible errors): ");
        String receivedData = scanner.nextLine();


        if (receivedData.length() != 7) {
            System.out.println("Please enter 7 bits of data.");
            return;
        }


        String decodedData = decodeHammingCode(receivedData);
        System.out.println("Decoded Data: " + decodedData);
    }

    private static String encodeHammingCode(String data) {

        int[][] hammingCodeMatrix = {
                {1, 0, 0, 0},
                {0, 1, 0, 0},
                {0, 0, 1, 0},
                {0, 0, 0, 1},
                {1, 1, 0, 1},
                {1, 0, 1, 1},
                {0, 1, 1, 1}
        };


        int[] inputDataArray = new int[4];
        for (int i = 0; i < 4; i++) {
            inputDataArray[i] =
Integer.parseInt(String.valueOf(data.charAt(i)));
```

```
        }


        int[] parityBits = new int[3];
        for (int i = 0; i < 3; i++) {
            parityBits[i] = (inputDataArray[0] * hammingCodeMatrix[4][i]) ^
                            (inputDataArray[1] * hammingCodeMatrix[5][i]) ^
                            (inputDataArray[2] * hammingCodeMatrix[6][i]);
        }


        StringBuilder encodedData = new StringBuilder();
        for (int bit : inputDataArray) {
            encodedData.append(bit);
        }
        for (int bit : parityBits) {
            encodedData.append(bit);
        }

        return encodedData.toString();
    }

    private static String decodeHammingCode(String receivedData) {

        int[][] hammingCodeMatrix = {
                {1, 0, 0, 0},
                {0, 1, 0, 0},
                {0, 0, 1, 0},
                {0, 0, 0, 1},
                {1, 1, 0, 1},
                {1, 0, 1, 1},
                {0, 1, 1, 1}
        };


        int[] receivedDataArray = new int[7];
        for (int i = 0; i < 7; i++) {
            receivedDataArray[i] =
Integer.parseInt(String.valueOf(receivedData.charAt(i)));
        }


        int[] syndrome = new int[3];
        for (int i = 0; i < 3; i++) {
            syndrome[i] = (receivedDataArray[0] * hammingCodeMatrix[4][i]) ^
                          (receivedDataArray[1] * hammingCodeMatrix[5][i]) ^
                          (receivedDataArray[2] * hammingCodeMatrix[6][i]);
        }


        int errorPosition = syndrome[0] * 1 + syndrome[1] * 2 + syndrome[2] * 4;

        if (errorPosition != 0) {
```

```
            System.out.println("Error at position: " + errorPosition);


            receivedDataArray[errorPosition - 1] =
(receivedDataArray[errorPosition - 1] + 1) % 2;
            System.out.print("Corrected data:- ");
            for (int i = 0; i < 7; i++) {
               System.out.print(receivedDataArray[i]);
            }
            System.out.println();

        }



        int[] decodedDataArray = new int[4];
        decodedDataArray[0] = receivedDataArray[2];
        decodedDataArray[1] = receivedDataArray[4];
        decodedDataArray[2] = receivedDataArray[5];
        decodedDataArray[3] = receivedDataArray[6];


        StringBuilder decodedData = new StringBuilder();
        for (int bit : decodedDataArray) {
            decodedData.append(bit);
        }

        return decodedData.toString();
    }
}
```

Output:

```
PS C:\Users\Admin\Documents\AIDS-B1-75-Rushikesh-Tanpure> cd "c:\Users\Admin\
Enter 4 bits of data: 1101
Encoded Data: 1101011
Enter received data (with possible errors): 1101111
Error at position: 6
Corrected data:- 1101101
Decoded Data: 0101
```