**PSG COLLEGE OF TECHNOLOGY**

**DEPARTMENT OF APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCES**

**M. Sc. DATA SCIENCE - COMPUTER NETWORKS LAB – 5th Semester**

**PROBLEM SHEET - 2 – HAMMING CODE**

The purpose of this lab assignment is for you to gain familiarity with Hamming codes. Write socket programs in Python for Sender and Receiver that will implement Hamming code to accomplish single-bit error detection and correction.

**Tasks to be performed:**

1. The Sender program must generate parity bits for dataword got from the user as input, using even parity and embed it into the dataword to form the codeword and send it to the receiver.
2. The Receiver must receive the codeword and generate the parity bits using even parity. If one of these parity bits produces an odd number of bits, the receiver knows that a transmission error occurred. Display the bit position in which error has occurred.

**Note:** Manually modify a bit (including in those bits added by the Hamming code algorithm) in the codeword at the sender before sending it and check whether the receiver detects it. The working of Hamming code is given below for your reference.

**Hamming Code**

In 1950, Richard Hamming developed an innovative way of adding bits to a number in such a way that transmission errors involving no more than a single bit could be detected and corrected.

The number of parity bits depends on the number of data bits:

|  |
| --- |
| **Data Bits : 4 8 16 32 64 128** |
| **Parity Bits: 3 4 5 6 7 8** |
| **Codeword : 7 12 21 38 71 136** |

For a data of size 2n bits, n+1 parity bits are embedded to form the codeword. It's interesting to note that doubling the number of data bits results in the addition of only 1 more data bit.

**Placing the Parity Bits**

We will number the bits from left to right, beginning with 1. In other words, bit 1 is the most significant bit.

The parity bit positions are powers of 2: {1,2,4,8,16,32...}. All remaining positions hold data bits. Here is a table representing a 21-bit code word:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| P | P |  | P |  |  |  | P |  |  |  |  |  |  |  | P |  |  |  |  |  |

The 16-bit data value 1000111100110101 would be stored as follows:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| P | P | 1 | P | 0 | 0 | 0 | P | 1 | 1 | 1 | 1 | 0 | 0 | 1 | P | 1 | 0 | 1 | 0 | 1 |

**Calculating Parity**

The following table shows exactly which data bits are checked by each parity bit in a 21-bit code word:

|  |  |  |
| --- | --- | --- |
| **Parity Bit** | **Checks the following Data Bits** | **Hint\*** |
| 1 | 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21 | use 1, skip 1, use 1, skip 1, ... |
| 2 | 2, 3, 6, 7, 10, 11, 14, 15, 18, 19 | use 2, skip 2, use 2, skip 2, ... |
| 4 | 4, 5, 6, 7, 12, 13, 14, 15, 20, 21 | use 4, skip 4, use 4, ... |
| 8 | 8, 9, 10, 11, 12, 13, 14, 15 | use 8, skip 8, use 8, ... |
| 16 | 16, 17, 18, 19, 20, 21 | use 16, skip 16, ... |

**Encoding a Data Value (Sender):**

For the first example, let's use the 8-bit data value 1 1 0 0 1 1 1 1, which will produce a 12-bit code word. Let's start by filling in the data bits:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| P | P | **1** | P | **1** | **0** | **0** | P | **1** | **1** | **1** | **1** |

Next, we begin calculating and inserting each of the parity bits.

**P1:** To calculate the parity bit in position 1, we sum the bits in positions 3, 5, 7, 9, and 11: (1+1+0+1+1 = 4). This sum is even (indicating *even parity*), so parity bit 1 should be assigned a value of 0. By doing this, we allow the parity to remain even:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| **0** | P | **1** | P | **1** | **0** | **0** | P | **1** | **1** | **1** | **1** |

**P2:** To generate the parity bit in position 2, we sum the bits in positions 3, 6, 7, 10, and 11: (1+0+0+1+1 = 3). The sum is odd, so we assign a value of 1 to parity bit 2. This produces even parity for the combined group of bits 2, 3, 6, 7, 10, and 11:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 0 | **1** | **1** | P | **1** | **0** | **0** | P | **1** | **1** | **1** | **1** |

**P4:** To generate the parity bit in position 4, we sum the bits in positions 5, 6, 7, and 12: (1+0+0+1 = 2). This results in **even** parity, so we set parity bit 4 to zero, leaving the parity even:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 0 | 1 | **1** | **0** | **1** | **0** | **0** | P | **1** | **1** | **1** | **1** |

**P8:** To generate the parity bit in position 8, we sum the bits in positions 9, 10, 11 and 12: (1+1+1+1 = 4). This results in **even** parity, so we set parity bit 8 to zero, leaving the parity even:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 0 | 1 | **1** | 0 | **1** | **0** | **0** | **0** | **1** | **1** | **1** | **1** |

All parity bits have been created, and the resulting code word is: 011010001111.

**Detecting a Single Error (Receiver):**

When a code word is received, the receiver must verify the correctness of the data. This is accomplished by counting the 1 bit in each bit and verifying that each has even parity. Recall that we arbitrarily decided to use even parity when creating code words. Here are the bit groups for a 12-bit code value:

|  |  |
| --- | --- |
| **Parity Bit** | **Bit Group** |
| 1 | 1, 3, 5, 7, 9, 11 |
| 2 | 2, 3, 6, 7, 10, 11 |
| 4 | 4, 5, 6, 7, 12 |
| 8 | 8, 9, 10, 11, 12 |

If one of these groups produces an odd number of bits, the receiver knows that a transmission error occurred. As long as only a single bit was altered, it can be corrected.

Arrange the calculated parity bits in the below order at the receiver side to detect and correct single bit error.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P8** | **P4** | **P2** | **P1** | **Inference** |
| 0 | 0 | 0 | 0 | denotes no error |
| 0 | 1 | 0 | 1 | denotes error in 5th bit (0101 is the binary value for 5) |

If two errors were to occur, we could detect the presence of an error, but it would not be possible to correct the error by correctly identifying their bit positions.