SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
V Semester - CSE 'A'
UCS1511 NETWORKS LAB

Exercise 6: Computing Hamming code for Error correction

AIM: To implement Hamming Code for Single Error Correction using C socket program.

Server should perform the following:

1. Read the input from a user (zero's and one's)
2. Encoding a message by Hamming Code
    a. Calculate the number of redundant bits.
    b. Position the redundant bits.
    c. Calculate the values of each redundant bit.
3. Introduce error (single bit error or no error)
4. Send the data to receiver

The receiver should do the following:

1. Receive the data from the sender and.
2. Check for any error by performing the following operations
    a. Calculation of the number of redundant bits.
    b. Positioning the redundant bits.
    c. Parity checking.
    d. If any error, correct the error and display the original message.

Sample Input and Output

Server
Sender
Input data: 1010101
Number of redundant bits needed is: 4 (your program should find this number)
Data with redundant bits: 10100101111
Introduce error in data: 10101101111

Receiver
Data received: 10101101111
Calculated redundant bits: 0111
Corrected data: 1010101

Server side:

Procedure for 2.a: Calculation of the number of redundant bits

1. If the message contains *m* number of data bits, *r* number of redundant bits; Solve r from $2^r \geq m + r + 1$
    - If the data is of 7 bits, *m = 7*, the minimum value of *r* that will satisfy the above equation is *4, ($2^4 \geq 7 + 4 + 1$).* The total number of bits in the encoded message, *(m + r) = 11*

Procedure for 2.b: Positioning the redundant bits

1. The *r* redundant bits placed at bit positions of powers of 2, i.e. 1, 2, 4, 8, 16 etc.
2. Referred to as *r1* (at position 1), *r2* (at position 2), *r3* (at position 4), *r4* (at position 8) and so on.
    - If, *m = 7, r* comes to 4, the positions of the redundant bits are as follows

11	10	9	8	7	6	5	4	3	2	1
d	*d*	*d*	r_4	*d*	*d*	*d*	r_3	*d*	r_2	r_1

Procedure for 2.c: Calculating the values of each redundant bit

The redundant bits are parity bits. A parity bit is an extra bit that makes the number of 1s either even or odd. The two types of parity are:

Even Parity − The total number of bits in the message is made even.

Odd Parity − The total number of bits in the message is made odd.

Each redundant bit, r_i, is calculated as the parity, assuming to maintain even parity, based upon its bit position.

It covers all bit positions whose binary representation includes a 1 in the i^{th} position except the position of r_i. Thus:

1. r_1 is the parity bit for all data bits in positions whose binary representation includes a 1 in the least significant position excluding 1 (3, 5, 7, 9, 11 and so on)

2. r_2 is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 2 from right except 2 (3, 6, 7, 10, 11 and so on)

3. r_3 is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 3 from right except 4 (5-7, 12-15, 20-23 and so on)

    - Suppose that the message **1100101** needs to be encoded using even parity Hamming code. Here, **m = 7** and r comes to **4**. The values of redundant bits will be as follows

11	10	9	8(r_4)	7	6	5	4(r_3)	3	2(r_2)	1(r_1)
1	1	0	0	0	1	0	1	1	0	0

    - Hence, the message sent will be **11000101100**.

Client side:

Procedure for 2.a: Calculation of the number of redundant bits

1. Using the same formula as in encoding, the number of redundant bits is ascertained. $2r \geq m + r + 1$

where *m* is the number of data bits and *r* is the number of redundant bits.

Procedure for 2.b: Positioning the redundant bits

1. The **r** redundant bits placed at bit positions of powers of 2, i.e. 1, 2, 4, 8, 16 etc.

Procedure for 2.c: Parity checking

1. Parity bits are calculated based upon the data bits and the redundant bits using the same rule as during generation of c1, c2, c3, c4 etc. Thus

   - c1 = parity(1, 3, 5, 7, 9, 11 and so on)
   - c2 = parity(2, 3, 6, 7, 10, 11 and so on)
   - c3 = parity(4-7, 12-15, 20-23 and so on)

Procedure for 2.d: Error detection and correction

1. The decimal equivalent of the parity bits binary values is calculated.
   a) If it is 0, there is no error.
   b) Otherwise, the decimal value gives the bit position which has error.

   - For example, if *c1c2c3c4 = 1001*, it implies that the data bit at position 9, decimal equivalent of 1001, has error.
   - The bit is flipped (converted from 0 to 1 or vice versa) to get the correct message.

Example 4 – Suppose that an incoming message 11110101101 is received.

2a. At first the number of redundant bits is calculated using the formula $2r \geq m + r + 1$. Here, $m + r + 1 = 11 + 1 = 12$. The minimum value of *r* such that $2r \geq 12$ *is 4*.

2b. The redundant bits are positioned as below –

11	10	9	$8(r_4)$	7	6	5	$4(r_3)$	3	$2(r_2)$	$1(r_1)$
1	1	1	1	0	1	0	1	1	0	1

2c. Even parity checking is done :

- c1 = even_parity(1, 3, 5, 7, 9, 11) = 0
- c2 = even_parity(2, 3, 6, 7, 10, 11) = 0
- c3 = even_parity (4, 5, 6, 7) = 0

- c4 = even_parity (8, 9, 10, 11) = 0

2d. Since the value of the check bits c1c2c3c4 = 0000 = 0, there are no errors in this message.

Reference: https://www.tutorialspoint.com/hamming-code-for-single-error-correction-double-error-detection