MIT WORLD PEACE UNIVERSITY

Computer Networks
Second Year B. Tech, Semester 3

ERROR DETECTION AND CORRECTION WITH HAMMING CODE

PRACTICAL REPORT
ASSIGNMENT 4

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

November 20, 2022

Contents

1 Aim and Objectives

Aim

To write a program for error detection and correction using Hamming Codes

Objectives

1. To encode and decode original data bits with the help of parity bits
2. To demonstrate use of error control protocols

2 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Programs Used: Cisco Packet Tracer v8.2 **Compiler Used**: g++ on Linux for Compiling C++

3 Code

```
// You will be given a string as input and you have to find the resulting hamming
    code to be sent.
// Also check which bit if flipped after flipping it.

#include <iostream>
#include <cmath>

using namespace std;

unsigned long int m;
int r_array[20][20];
int r_val = 0;      // value of r, or number of bits of r that need to be put in.
int error_bit = 0; // the bit that was changed by the user and detected by program
    .

void display_r()
{
    for (int i = 0; i < 20; i++)
    {
        for (int j = 0; j < 20; j++)
        {
            cout << r_array[i][j] << " ";
        }
        cout << endl;
    }
}

// Returns the length of the resulting hamming code
int calc_length(string input)
{
    //  2^r_array > m + r_array + 1
    m = input.length();
    int r = 0;
    for (int i = 0; i < 10; i++)
    {
```

```
        if (pow(2, i) >= int(m) + i + 1)
        {
            r = i;
            break;
        }
    }
    ::r_val = r;
    return int(m) + r;
}

// Converts Binary array into decimal
void convert_binary_to_decimal(bool parity[])
{
    int decimal = 0;
    for (int i = 0; i < ::r_val; i++)
    {
        if (parity[i])
        {
            decimal += pow(2, i);
        }
    }
    if (decimal)
    {
        ::error_bit = decimal;
    }
}

// Fills the values with r_array in the 2d array
void fill_r_values(int hamming_len)
{
    int count;
    bool should_add;
    for (int k = 0; k < hamming_len; k++)
    {
        count = 0;
        should_add = false;
        for (int i = 0, j = 1; i <= hamming_len; i++)
        {
            if (count == pow(2, k))
            {
                count = 0;
                should_add = !should_add; // flips it
            }

            if (should_add)
            {
                r_array[k][j] = i;
                j++;
            }
            count++;
        }
    }
}

// Fills the first column of the r_array table, to 1 or 0 for maintaining even
   parity.
void fill_r_parity(int hamming_len, const bool hamming[])
{
    int count;
```

```
    bool parity;
    for (int i = 0; i < ::r_val; i++)
    {
        // check parity
        count = 0;
        for (int j = 2; j <= (hamming_len / 2) + 1; j++)
        {
            hamming[r_array[i][j] - 1] ? count++ : count;
        }
        parity = count % 2 != 0; // if number of 1's is even
        r_array[i][0] = parity;  // assign parity bit
    }
}

// Fills the hamming array by looking at the parity r_array bits from the r_array array.
void fill_hamming(int hamming_len, bool hamming[])
{
    int k = 0;
    for (int j = 0; j < hamming_len; j++)
    {
        if (j == pow(2, k) - 1)
        {
            hamming[j] = r_array[k][0];
            k++;
        }
    }
}

void display_hamming(int hamming_len, bool hamming[])
{
    for (int i = hamming_len - 1; i >= 0; i--)
    {
        cout << hamming[i];
    }
    cout << endl;
}

// This function does the entire error correction, and prints the process as well
void detect_errors(int hamming_len, bool hamming[50])
{
    int count;
    bool parity[::r_val];

    // Display new hamming code with flipped bit, and the old one as well.

    // Deduce values of r_array from the new hamming code

    // from the previous r_array table that we already have,
    for (int i = 0; i < ::r_val; i++)
    {
        // check parity
        count = 0;
        for (int j = 1; j <= hamming_len / 2 + 1; j++)
        {
            hamming[r_array[i][j] - 1] ? count++ : count;
        }
        parity[i] = count % 2 != 0; // if number of 1's is even
    }
```

```
    // converted parity bits to decimal, and then find the flipped bit
    convert_binary_to_decimal(parity);

    // Display the flipped bit and then the corrected hamming code, with the
   original hamming code.
    cout << "The Bit which was changed is: " << ::error_bit << endl;
    cout << "The Hamming code with the correction is: " << endl;
    hamming[::error_bit - 1] = !hamming[::error_bit - 1];
    display_hamming(hamming_len, hamming);
}

int main()
{
    string input;
    int hamming_len, flipped_bit = 0;
    // Input the value as a string, as we don't know how long it can be.
    cout << "Enter the Input : " << endl;
    cin >> input;

    // Edge Case
    if (input.length() == 0)
        return 0;
    else
        m = input.length();

    // Find the value of r_array and the length of the hamming code
    hamming_len = calc_length(input);

    // Declare an array to store the hamming code
    bool hamming[50] = {};

    // Store the bits
    for (int i = 0, j = 0, k = int(m); i < hamming_len; i++)
    {
        if (i != (pow(2, j) - 1))
        {
            hamming[i] = (input[k - 1] == '1');
            k--;
        }
        else
        {
            j++;
        }
    }

    // fill the values of r_array till hamming_len
    fill_r_values(hamming_len);

    // Fill r_array with even parity
    fill_r_parity(hamming_len, hamming);

    // Fill the hamming code
    fill_hamming(hamming_len, hamming);

    cout << "The Hamming code to be sent by the sender is: " << endl;
    display_hamming(hamming_len, hamming);

    // Implement Error Detection

```

```
    cout << "What bit would you like to flip? (Starting from 1, from right)" <<
    endl;
    cin >> flipped_bit;
    // Changing the Hamming code
    hamming[flipped_bit - 1] = !hamming[flipped_bit - 1];
    cout << "The Hamming code after the error is: " << endl;
    display_hamming(hamming_len, hamming);

    cout << "Now Calculating Error" << endl;
    detect_errors(hamming_len, hamming);

    return 0;
}
```

Listing 1: Hamming Code.cpp

4 Output

```
Enter the Input :
11011011011
The Hamming code to be sent by the sender is:
110110111011110
What bit would you like to flip? (Starting from 1, from right)
3
The Hamming code after the error is:
110110111011010
Now Calculating Error
The Bit which was changed is: 3
The Hamming code with the correction is:
110110111011110
```

Listing 2: Output for Hamming Codes

5 Conclusion

Thus learnt how error correction works, and implemented a simple program using Hamming Codes. Hamming Codes were understood in detail along with the logic behind error correction.

20/11/22

Assignment - 4

Krishnaraj P
Batch A1, 20
1032210985

Hamming Code

Theory

→ Types of Errors

1. Single Bit Errors

10110101
↓ transported
10010101

1 bit gets changed by mistake

2. Multiple Bit Errors - many bits are changed during transport.

3. Burst Errors : Consecutive Bits end up corrupted.

0110101 ⟶ 0000101

(*) Concept of Parity Bits

Parity is done by adding an extra bit called parity to the data to count the number of 1's and or 0's.

it is of 2 types:

(a) <u>even</u>: If no. of 1's == even,
parity bit = 0
else : parity bit = 1.

(b) odd : If no. of 1's == odd ;
parity bit = 0
else : parity bit = 1.

(*) Hamming Code example

eg. 1000001

data bits = 7 = m. ; r = parity bits

$2^r \geq m + r + 1 \Rightarrow r = 4$

total no. of bits to be considered = 11.

$r = 4. = R_1, R_2, R_3, R_4$

$2^0 \quad 2^1 \quad 2^2 \quad 2^3$

$1 \quad 2 \quad 4 \quad 8$

→

D_7	D_6	D_5	R_4	D_4	D_3	D_2	R_3	D_1	R_2	R_1
1	0	0	x	0	0	0	x	1	x	x

To find parity: (even)

R_1 = Pos: $1 + 3 + 5 + 7 + 11 =$

$1 + 0 + 0 + 0 + 1 \Rightarrow 0$

R_2 = pos: $3 + 6 + 7 + 10 + 11$

$\Rightarrow 1 + 0 + 0 + 0 + 1 \Rightarrow 0$

R_3 = pos: $5 + 6 + 7$

$\Rightarrow 0 + 0 + 0 \Rightarrow 0$

R_4 = pos: $9 + 10 + 11$

$\Rightarrow 0 + 1 + 1 \Rightarrow 1$

So code is:

1	0	0	1	0	0	1	0	1	0	0

(*) FAQs.

(Q.1) Flow vs. Error control

Flow Control	Error control
— Only ~~meant~~ meant for transmission of data	Meant for transmission of error free data.
— There are 2 approaches feedback based and rate based.	Approaches are : Hamming code, CRC, checksum. etc.
— prevents loss of data.	Used to detect and correct data.

(Q.2) Explain in brief 2 Error control Mechanisms.

→ Error Detection : It involves checking if any error has occured or not. The number of error bits and type of error does not matter here.

→ Error Correction : It involves ascertaining the exact no. of bits that have been corrupted and finding the location of those bits.